UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA

Relatório de Estágio Supervisionado

Ferramenta de *Hardware* e *Software* de Aquisição de Dados para Implementação de um Sistema de Localização

Aluno
Antonio Agripino da Costa Filho

Orientador
Prof. D.Sc. José Sérgio da Rocha Neto

Campina Grande, Fevereiro de 2010

ANTONIO AGRIPINO DA COSTA FILHO

Ferramenta de *Hardware* e *Software* de Aquisição de Dados para Implementação de um Sistema de Localização

Relatório de Estágio Supervisionado apresentado à Universidade Federal de Campina Grande como requesito parcial para a obtenção do título de Engenheiro Eletricista.

Orientador: Prof. D.Sc. José Sérgio da Rocha Neto

Campina Grande, Fevereiro de 2010

Dedicatória

Sumário

1 Introdução 4

2 O Padrão IEEE 802.15.4 5

2.1 Rápida Visão Sobre Redes de Sensores Sem Fio 5
2.2 IEEE 802.15.4 7
2.3 Topologias de Rede 8
2.4 Camada Física (PHY) 9
2.4.1 Bandas de Frequência de Trabalho 9
2.4.2 Detecção de Energia 11
2.4.3 Percepção da Portadora (CS) 11
2.4.4 Indicador de Qualidade de *Link* (LQI) 11
2.4.5 Avaliação de Canal Livre (CCA) 11
2.5 Camada de Acesso ao Meio (MAC) 12
2.5.1 Associação/Dissociação da PAN 12
2.5.2 CSMA-CA 12
2.5.3 Redes com/sem *Beacon* 13
2.6 Endereçamento 14

3 Introdução aos Sistemas de Estimação de Localização 15

4 Materiais 17

4.1 XBee/XBee-Pro 17
4.2 ADuC 842 19

5 Sistema de Aquisição de Dados 20

5.1 *Hardware* 20
5.2 Interface com o Usuário 23

6 Resultados 26

7 Conclusões 28

8 Bibliografia 29

9 Anexos 30

9.1 Programa do ADuC para os nós móveis 30
9.2 Programa do ADuC para os nós âncoras 33
9.3 Programa desenvolvido no BCB 36

1 Introdução

As redes de sensores sem fio tem revolucionado o projeto de sistemas embarcados e despertado um novo conjunto de potenciais aplicações. Uma dessas aplicações encontra-se na estimação da localização de um objeto/pessoa/animal, seja em locais abertos ou fechados, já que os sensores sem fio têm-se tornados mais presentes, o que barateia seu custo de aplicação frente a sistemas que usam GPS (*Global Position System*). Um exemplo de seu uso é em sítios/fazendas para determinar o posicionamento dos animais que se encontram soltos no pasto.

Os estudos neste trabalho utilizaram as redes de sensores sem fio baseadas no padrão IEEE (*Institute of Electrical and Electronics Engineers*) 802.15.4 para o desenvolvimento de uma plataforma de aquisição de dados para coleta de informações que serão posteriormente utilizadas em um processo de caracterização elétrica de um ambiente onde um sistema de estimação de localização será instalado.

Os trabalhos foram realizados no Laboratório de Instrumentação Eletrônica e Controle (LIEC) do curso de Engenharia Elétrica na Universidade Federal de Campina Grande (UFCG) – Campus Campina Grande, sob supervisão do orientador e próprio coordenador do LIEC, o professor D.Sc. José Sérgio da Rocha Neto, e do aluno de mestrado José Maurício Ramos de Souza Neto.

No LIEC são desenvolvidas atividades de pesquisa nas seguintes áreas:

- Estabilidade de sistemas de controle distribuídos operando em tempo real;
- Automação e controle de processos industriais utilizando redes de sensores/atuadores sem fio;
- Confiabilidade dos Sistemas Instrumentados de Segurança;
- Implementação e melhorias no *BR-Tuning*: Sistema para avaliação e sintonia de Controladores PID;
- Automação de procedimentos, identificação de usuários e controle de acesso em unidades industriais utilizando tecnologia RFID e dispositivos móveis;
- Interfaces Interativas com uma *Network Camera* utilizando C# para Aplicações em Automação Industrial.

2 O Padrão IEEE 802.15.4

2.1 Rápida Visão Sobre Redes de Sensores Sem Fio

As redes de sensores sem fio (RSSF) podem ser encontradas em várias aplicações, tais como automação residencial, automação industrial e sistemas de saúde, por exemplo. Isso se deve as RSSF tornassem cada vez mais usadas/presentes graças à compactação e barateamento dos seus componentes.

Uma RSSF é composta basicamente por quatro elementos. São eles:

- Um dispositivo sensor;
- A interconexão de rede, no caso sem fio;
- Um dispositivo central que coordena os sensores;
- Um equipamento capaz de coletar os dados e processá-los.

Normalmente apresentam grandes quantidades de dispositivos sensores, conhecidos como nós sensores. Assim, o processo de comunicação e gerenciamento da rede tende a ser mais complexo à medida que a quantidade de nós sensores aumenta.

Os métodos de redes sem fio de curto alcance são divididos em duas categorias principais: redes de área local sem fio (WLAN – *Wireless Local Area Networks*) e redes de área privada sem fio (WPAN – *Wireless Personal Area Networks*). Uma WLAN é uma substituição ou extensão para redes de área local com fio, tais como a *ethernet*. Por outro lado, as WPAN's não foram desenvolvidas para substituir qualquer rede com fio. Elas foram criadas para fornecer os recursos para uma comunicação sem fio eficiente em energia dentro de um espaço operacional privado (POS – *Personal Operating Space*) sem a necessidade de qualquer infraestrutura. Um POS é uma região esférica que circunda um dispositivo sem fio e tem um raio de 10 metros.

Para que todos os dispositivos consigam utilizar o meio de comunicação para enviar e receber dados, é necessário que se estabeleça o protocolo de comunicação. No início da utilização das RSSF, cada fabricante idealizou seu próprio protocolo de comunicação, inviabilizando a utilização de sensores de vários fabricantes e tornando o gerenciamento desses dispositivos muito mais difíceis. Essas interfaces não seguiram adiante, sendo substituídas por padrões definidos pelo IEEE para viabilização das

RSSF, visando principalmente a interoperabilidade, ou seja, a capacidade de haver troca de dados e realizações de operações entre dispositivos de diferentes fabricantes.

A padronização surgiu por meio dos grupos de trabalho que formam o IEEE 802, que propõe discutir e padronizar protocolos de redes. Cada grupo procura um consenso para a padronização dentro das características de cada rede. No Grupo 11 discute-se sobre as redes WLAN. Já no Grupo 15 são discutidas as redes WPAN.

No caso dos padrões IEEE 802.15, existe mais de uma implementação, já que dentro do Grupo 15 existem sete grandes grupos de trabalho. Três desses grupos de trabalho são exemplificados a seguir:

- IEEE 802.15.1: tem produzido um padrão baseado nas especificações *Bluetooh*;
- IEEE 802.15.3: são RSSF de alta taxa de transferência de dados que utilizam UWB (*Ultra-Wide Band*);
- IEEE 802.15.4: são RSSF de baixa taxa de transferência de dados, porém com longa vida de bateria; a implementação desse padrão é a base e origem para outro protocolo, conhecido como *ZigBee*.

Além da classificação que toma como base a área de cobertura, as redes sem fio podem ser classificadas de acordo com a taxa de transmissão, como visto na Figura 1.

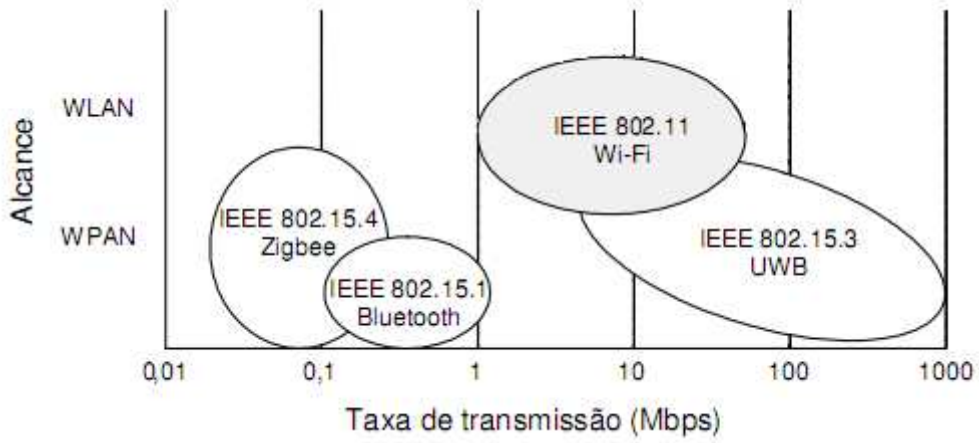

Figura 1 – Comparação das redes sem fio com relação à taxa de transmissão

Da observação da Figura 1 juntamente com as características do protocolo visando o trabalho a ser desenvolvido, escolheu-se o padrão IEEE 802.15.4 para ser usado.

2.2 IEEE 802.15.4

O padrão IEEE 802.15.4 define as características de construção da camada física (PHY – *Physical*) e da camada de acesso ao meio (MAC – *Media Access Control*) para redes sem fio com baixas taxas de transmissão, definidas como LR-WPAN (*Low-Rate Wireless Personal Area Networks*). É voltado, principalmente, para aplicações onde a baixa taxa de dados, o baixo custo e a longa vida da bateria (baixo consumo de energia) são os principais requerimentos. Em muitas aplicações, o tempo total em uso dos dispositivos sem fio empregado em qualquer tipo de atividade é muito limitado, ou seja, os dispositivos gastam a maior parte do tempo em um modo de economia de energia, conhecido como modo *sleep*. Como resultado, os dispositivos são capazes de serem operacionais por um longo tempo antes de suas baterias serem substituídas.

Há certa confusão ao se falar/pensar no padrão 802.15.4 relacionando-o diretamente à tecnologia *ZigBee*. A tecnologia *ZigBee* foi desenvolvida por uma associação formada por várias empresas (tais como *Motorola*, *Texas Instruments* e *Philips*, por exemplo) conhecida como *ZigBee Alliance* e que tem como objetivo desenvolver padrões e produtos sem fio de monitoramento e controle que sejam confiáveis, de baixo custo, baixo consumo e conectados em redes sem fio como um padrão global aberto. A tecnologia *ZigBee* define as camada de rede (NWK – *Network*) e de aplicação (APL – *Applicattion*), sendo implementadas sobre o padrão IEEE 802.15.4 que, relembrando, define as camadas física e de acesso ao meio. As camadas do protocolo *ZigBee* são observadas na Figura 2.

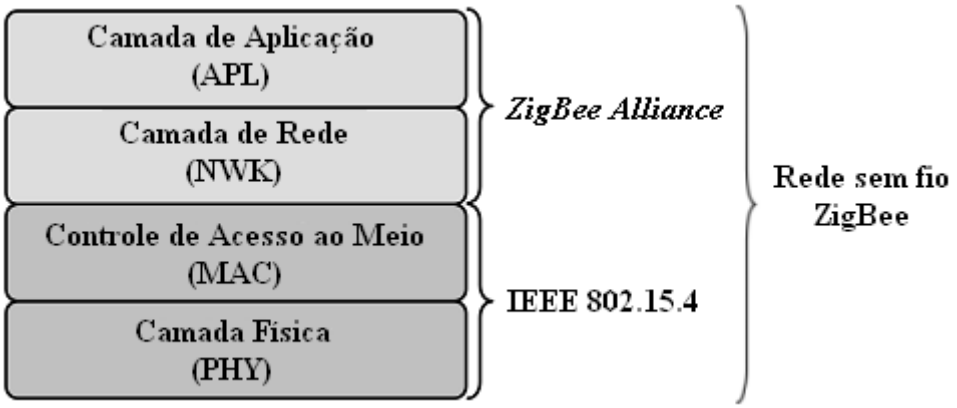

Figura 2 – Camadas da pilha de protocolo da tecnologia *ZigBee*

Há dois tipos de dispositivos em uma rede sem fio IEEE 802.15.4, os dispositivos de função completa (FFD – *Full-Function Devices*) e os dispositivos de função reduzida (RFD – *Reduced-Function Devices*). Um FFD é capaz de executar todas as tarefas descritas no padrão IEEE 802.15.4. Por outro lado, um RFD tem capacidades limitadas e cuja intenção é para aplicações que são extremamente simples, tais como liga/desligar uma lâmpada, por exemplo. Um FFD pode comunicar com qualquer outro dispositivo dentro da rede, enquanto que um RFD pode somente comunicar com um FFD.

Um FFD pode operar de três modos: como um coordenador de rede de área privada (PAN – *Personal Area Networks*), um coordenador ou um dispositivo. Um coordenador é um FFD capaz de rotear as mensagens dentro da rede. Se o coordenador é também o principal controlador da PAN, ele é dito ser o coordenador PAN. E se o dispositivo não atua como coordenador, ele é dito ser simplesmente um dispositivo. O RFD só pode sem, dentro da rede IEEE 802.15.4, um dispositivo. Os modos de operação são mostrados na Figura 3.

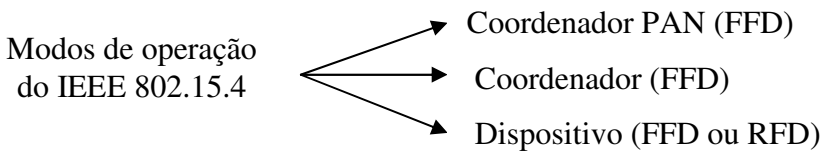

Figura 3 – Modos de operação no padrão IEEE 802.15.4

2.3 Topologias de Rede

No padrão IEEE 802.15.4 são definidos dois tipos de topologias de rede. A primeira delas é a topologia estrela (*star*) – Figura 4(a), onde há um elemento central sendo o coordenador PAN e vários elementos comuns comunicando somente com o ele. Um exemplo de uma topologia estrela seria um ambiente de uma sala com alguns sensores de temperatura e um coordenador posicionado no meio dessa sala. Se a temperatura aumentar, os elementos comuns devem comunicar ao coordenador PAN e esse deve decidir se vai ou não atuar junto ao ar condicionado para manter a temperatura determinada.

A segunda topologia é ponto a ponto (*peer-to-peer*) – Figura 4(b), onde existe ainda um elemento central como sendo o coordenador PAN, mas os vários

elementos comuns que formam a rede podem, além de comunicar com o coordenador PAN, comunicar com outros sensores comuns. Nessa topologia, os elementos comuns vão atuar também como roteadores. Esse papel de roteador é realizado pelos elementos comuns FFD, roteando pacotes de outros elementos RFD até o coordenador.

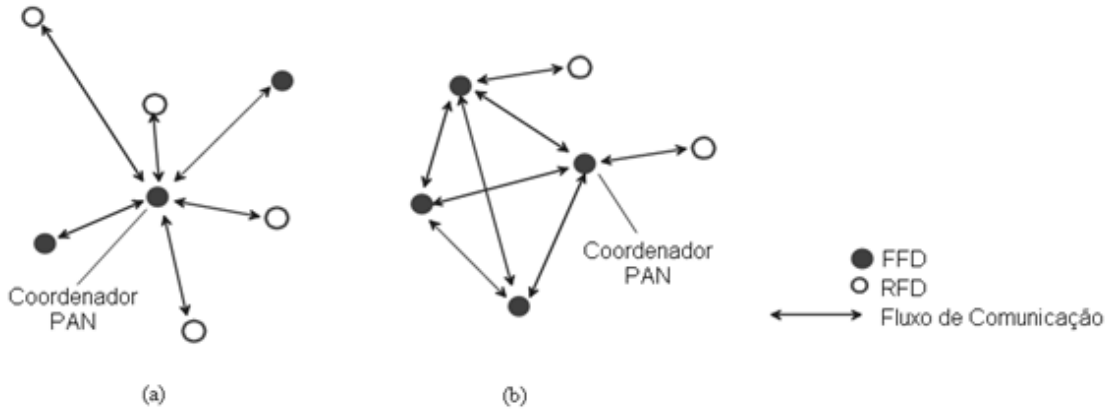

Figura 4 – Exemplos de topologia (a) Estrela (b) Ponto a ponto

2.4 Camada Física (PHY)

É responsável pela comunicação propriamente dita entre os vários dispositivos contidos na rede. As seguintes tarefas são executadas nessa camada:

- Ativação/desativação do transceptor de rádio;
- Detecção de energia (ED – *Energy Detection*) dentro do canal corrente;
- Indicador de qualidade do *link* (LQI – *Link Quality Indicator*) para pacotes recebidos;
- Avaliação de canal livre (CCA – *Clear Channel Assessment*) para CSMA-CA (*Carrier Sense Multiple Access With Collision Avoidance*);
- Seleção da frequência de canal;
- Transmissão e recepção de dados.

2.4.1 Bandas de Frequência de Trabalho

A camada física suporta três bandas de frequência:

- 868-868,6 MHz (banda de 868 MHz) com apenas um canal (Canal 0);
- 902-928 MHz (banda de 915 MHz) com dez canais (Canal 1-10);
- 2400-2483,5 MHZ (banda de 2,4 GHz) com dezesseis canais (Canal 11-26).

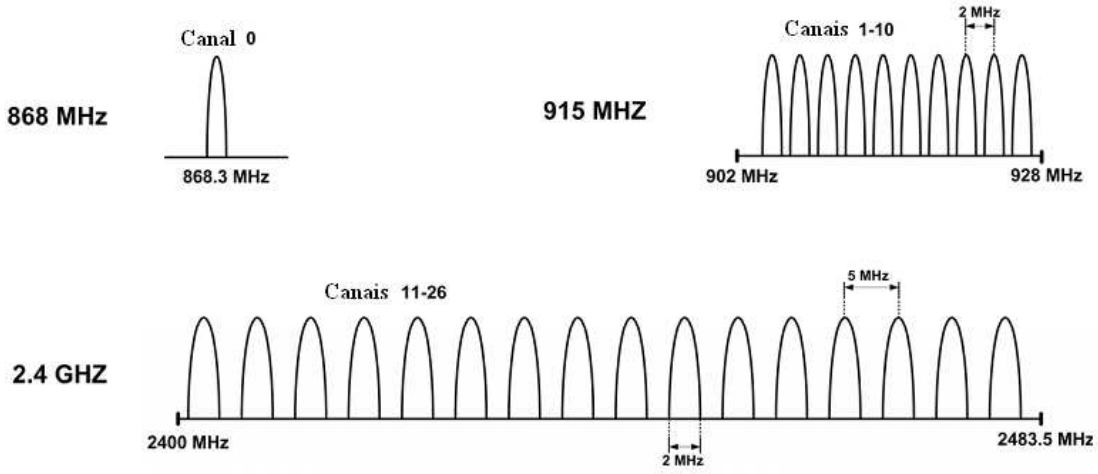

Figura 5 – Bandas de frequência

A banda de 868 MHz é usada na Europa. As outras duas bandas fazem parte das bandas ISM (*Industrial, Scientific and Medical*). A banda de 915 MHz é usada principalmente na América do Norte, enquanto a banda de 2,4 GHz é usada mundialmente.

Outras informações com relação às bandas de frequência encontram-se na Tabela 1.

Tabela 1 – Bandas de frequência e taxas de transmissão

PHY (MHz)	Banda de Frequência (MHz)	Canais	Modulação	Taxa de Transmissão (kbps)
868	868–868,6	1	BPKS	20
915	902–928	10	BPKS	40
2450	2400–2483,5	16	O-QPSK	250

2.4.2 Detecção de Energia

Quando um dispositivo planeja transmitir uma mensagem, ele primeiro entra no modo de recepção para detectar e estimar o nível de energia do sinal no canal desejado. Essa tarefa é conhecida como detecção de energia. Na ED, o receptor não tenta detectar o tipo de sinal, ele apenas estima o nível de energia do sinal. Em outras palavras, se um sinal está ocupando a banda de frequência de interesse, executar uma ED não revelará se esse sinal é um padrão IEEE 802.15.4 ou não.

2.4.3 Percepção da Portadora (CS)

Similar à detecção de energia, a percepção da portadora (CS – *Carrier Sense*) é uma maneira de verificar se um canal de frequência está disponível para uso. Ele realiza a mesma operação da ED, porém o sinal é demodulado para verificar se a modulação e espalhamento do sinal estão de acordo com as características da camada PHY que está correntemente em uso pelo dispositivo, ou seja, se o sinal é um sinal do padrão IEEE 802.15.4.

2.4.4 Indicador de Qualidade de *Link* (LQI)

O LQI é uma indicação da qualidade do pacote de dados recebido pelo receptor. A intensidade do sinal recebido (RSS – *Received Signal Strength*) pode ser usada como uma medida da qualidade do sinal. A RSS é uma medida da energia total do sinal recebido. A razão da energia do sinal desejado para a energia de ruído total na banda de frequência, ou seja, a razão sinal-ruído (SNR – *Signal-to-Noise Ratio*) é outra maneira de julgar a qualidade do sinal. As medições do LQI são executadas para cada pacote recebido.

2.4.5 Avaliação de Canal Livre (CCA)

Na primeira etapa do mecanismo de acesso ao canal CSMA-CA, a camada MAC requererá a camada PHY realizar uma CCA para garantir que o canal não está sendo em

uso por qualquer outro dispositivo. Em um CCA, os resultados da ED ou da CS podem ser usados para decidir se um canal de frequência deve ser considerado disponível ou ocupado.

2.5 Camada de Acesso ao Meio (MAC)

A camada MAC trata de todos os acessos ao canal físico e é responsável pelas seguintes tarefas:

- Geração de *beacons* de rede se o dispositivo é um coordenador PAN;
- Sincronização para *beacons* de rede (numa rede com *beacon*);
- Suportar associação/dissociação de PAN;
- Suportar proteção de dados na comunicação;
- Empregar o mecanismo CSMA-CA para acesso de canal;
- Tratar e manter o mecanismo de *slots* de tempo garantido (GTS – *Guaranteed Time Slots*);
- Fornecer um *link* confiável entre duas entidades pares de camada MAC.

2.5.1 Associação/Dissociação da PAN

Associação e dissociação são serviços fornecidos pelo IEEE 802.15.4 que podem ser usados para permitir dispositivos juntar ou deixar a PAN. Por exemplo, quando um dispositivo deseja juntar-se a uma PAN, ele envia um uma requisição de associação ao coordenador PAN. O coordenador PAN pode aceitar ou rejeitar a solicitação. O dispositivo usa a dissociação para notificar ao coordenador de sua intenção de deixar a rede.

2.5.2 CSMA-CA

O IEEE 802.15.4 implementa um método simples para permitir múltiplos dispositivos usarem o mesmo canal para comunicarem-se. O mecanismo de acesso ao canal usado é o CSMA-CA (*Carrier Sense Multiple Access With Collision Avoidance*).

No CSMA-CA, a qualquer instante um dispositivo que desejar transmitir primeiro deve executar o CCA para garantir que o canal não está sendo usado por qualquer outro dispositivo. Então, o dispositivo inicia a transmitir seu próprio sinal. A decisão para declarar um canal livre ou não pode ser baseada na medição da energia espectral na frequência do canal de interesse ou detectando o tipo de sinal que ocupa o canal.

Quando um dispositivo planeja transmitir um sinal, ele primeiro entra no modo ED para verificar o canal desejado está livre. Outra maneira é executando o CS, já que esse verifica, além do canal está ocupado ou não, se o sinal é um sinal do padrão IEEE 802.15.4. Se o canal está sendo usado, o dispositivo retira-se por um período aleatório de tempo e tenta novamente. Esse processo é repetido até ou o canal tornar-se livre ou o dispositivo atingir seu número máximo de tentativas definido pelo usuário.

2.5.3 Redes com/sem *Beacon*

Há dois métodos para acessar o canal: baseado em competição ou livre de competição. No acesso ao canal baseado em competição, todos os dispositivos que desejarem transmitir ao mesmo tempo no canal usam o mecanismo CSMA-CA e o primeiro que o canal livre inicia a transmissão. No método livre de competição, o coordenador PAN dedica um *slot* específico de tempo a um dispositivo em particular. Isso é chamado de *slot* de tempo garantido (GTS – *Guaranteed Time Slot*). Dessa forma, um dispositivo com um GTS alocado iniciará a transmissão durante aquele GTS sem usar o mecanismo CSMA-CA.

Para garantir um GTS, o coordenador PAN necessita garantir que todos os dispositivos na rede estejam sincronizados. *Beacon* é uma mensagem com formato específico que é usado para sincronizar os relógios dos nós da rede. Assim, um coordenador tem a opção para transmitir sinais *beacons* para sincronizar os dispositivos conectados a ele.

Uma rede na qual o coordenador PAN não transmite *beacons* é conhecido como uma rede sem *beacon*. Uma rede sem *beacon* não pode ter GTS porque os dispositivos não podem ser sincronizados de outra maneira.

2.6 Endereçamento

Cada dispositivo em uma rede necessita de um endereço único. O IEEE 802.15.4 usa dois métodos de endereçamento: endereçamento curto (16 bits) ou endereçamento estendido (64 bits).

Os endereços curtos permitem a comunicação dentro de uma rede simples. Permite também a redução no comprimento das mensagens e economiza no espaço de memória requerido que é alocado para armazenar os endereços.

A disponibilidade de endereçamento de 64 bits significa que o número máximo de dispositivos em uma rede pode ser de 2^{64}.

3 Introdução aos Sistemas de Estimação de Localização

Uma das aplicações de rede sem fio para curta distância é a determinação aproximada da localização física de um objeto em um dado instante de tempo. O conhecimento em tempo real de pessoas, propriedades e instrumentos portáteis pode tornar o gerenciamento mais eficiente.

A estimação de localização é o processo de obtenção da informação de localização de um nó em relação a um conjunto de posições de referência conhecidos.

Os sistemas de localização desenvolvidos que usam redes sem fio de curta distância também são chamados de sistemas de posicionamento local (LPS - *Local Positioning Systems*) para diferenciar do chamado sistema de posicionamento global (GPS - *Global Positioning System*). O sistema LPS, ao contrario de um sistema GPS, não usa informação proveniente de um satélite ou transmissor de longa distância.

A escolha do algoritmo de estimação de localização depende do cenário no qual o sistema é implementado. Os métodos de estimação de localização são comparados baseados na sua complexidade e desempenho. A precisão na localização, que relaciona a posição atual com a posição estimada, é talvez o mais intuitivo parâmetro de desempenho. Se o objetivo da estimação da localização é para o rastreamento de pessoas, uma precisão grosseira de alguns metros pode ser suficiente, por exemplo.

Escalabilidade é outro parâmetro que determina quão bem o algoritmo de localização se adapta quando o número de nós e a área de cobertura aumentam. Em aplicações de baixo custo, com nós associados a baterias, a capacidade computacional e o espaço na memória são limitados. Deste modo, estes nós não são apitos a implementação de algoritmos de maior complexidade computacional.

A estimação da localização usualmente envolve dois grupos de nós. O primeiro grupo consiste de nós fixos com localização conhecida. Estes nós fixos são referenciados como nós âncora (*Anchor Node*), são usados como referência para estimação de localização. A localização do nó âncora pode ser pode ser determinada pelo instalador, ou, esse pode ser equipado com um GPS para determinar sua própria localização.

O segundo grupo são os nós com posições desconhecidas, chamados de nós móveis (*Tracked Nodes*). O objetivo principal do sistema de localização é determinar a posição dos nós móveis com a ajuda dos nós âncoras.

A ideia básica de um sistema LPS pode ser resumida da seguinte forma. Um nó móvel com posição desconhecida emite um sinal que é recebido por seus vizinhos nós âncoras. O nó âncora mede o RSS, o tempo de viajem (TOA - *Time of Arrival*), ou o ângulo de chegada (AOA - *Angle of Arrival*) do sinal recebido. Estes valores medidos são usados como entradas para o algoritmo de localização utilizado que determina aproximadamente a localização do nó móvel. O algoritmo usa apenas uma dessas três entradas. A medição de RSS é bastante simples e os nós que usam o padrão IEEE 802.15.4 são capazes de medir o RSS para cada pacote recebido. A determinação do tempo de viajem requer um *clock* de alta precisão. Encontrar o ângulo de chegada requer modificações de *hardware* o que pode aumentar os custos. Assim, a maioria dos algoritmos de localização baseada em links RF usa apenas o RSS para estimar a localização devido a sua simplicidade e pouca ou nenhuma mudança de *hardware*.

4 Materiais

4.1 XBee/XBee-Pro

Os módulos RF (rádio frequência) XBee/XBee-Pro, Figura 6, fabricados pela *Digi International*®, foram projetados para satisfazer os padrões IEEE 802.15.4 e *ZigBee*, formando uma rede do tipo sem *beacon*, além de dá suporte a necessidade de obter uma RSSF de baixo custo e baixo consumo de energia. Os módulos operam dentro da faixa de frequência de 2,4 *GHz* e são compatíveis pino a pino com cada outro módulo. Permitem o controle do nível de potência de transmissão. Há três opções de antenas:

- Externa: um pedaço de fio de, aproximadamente, 2,5 cm soldada diretamente no módulo;
- Conector: a uma entrada para conectar uma antena externa ao módulo;
- Chip: é uma antena integrada que está soldada diretamente no módulo.

Com os dois primeiros tipos de antenas é possível direcionar o feixe de sinal, podendo-se melhorar, com isso, o desempenho da rede.

Figura 6 – Módulos RF XBee e XBee-Pro com diferentes antenas

A versão XBee possui uma sensibilidade de recepção de -92 dBm; máxima potência de transmissão de 0 dBm; consumo de transmissão de 45 mA; consumo de recepção de 50 mA; consumo no modo *sleep* menor de 10 µA; alcance do sinal de até 30 m para ambientes internos/urbanos e de até 100 m para ambientes externos com linha de visão.

A versão XBee-Pro possui sensibilidade de recepção de -100 dBm; máxima potência de transmissão de 20 dBm; consumo de transmissão de 215 mA; consumo de recepção de 55 mA; consumo no modo *sleep* menor de 10 µA; alcance do sinal de até 100 m para ambientes internos/urbanos e de até 1500 m para ambientes externos com linha de visão.

Para conectar os módulos ao computador, seja para a atualização do *firmware*, para transferências de dados ou para controle, foram utilizados as placas CON-USBBEE projetadas pela RogerCom® – Figura 7. Essas placas usam um chip conversor USB/Serial, regulador de tensão de baixa queda de tensão, comparador de tensão conectado aos LEDs (RSSI) que simulam a força do sinal de RF; LEDs indicadores de TX, RX, módulo ligado (ASS), e um botão para "resetar" o módulo conectado a placa.

Figura 7 – Placa CON-USBBEE

Outra maneira de comunicar-se com os mósdulos XBee/XBee-Pro é usando um microcontrolador (uC) por meio da comunicação serial assíncrona (UART) – Figura 8.

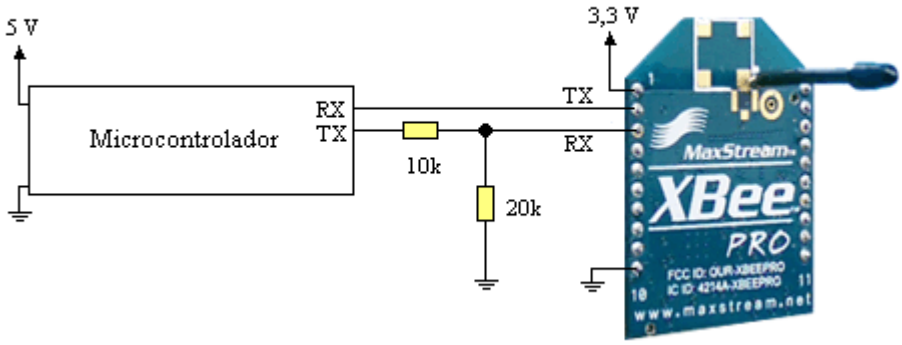

Figura 8 – Comunicação entre um microcontrolador e o módulo XBee/XBee-Pro

Para atualização/configuração do *firmware* dos módulos XBee/XBee-Pro, é utilizado o programa X-CTU, que é fornecido gratuitamente pela própria fabricante dos módulos RF.

Há dois modos de operação dos módulos: a operação transparente e a operação API (*Application Programming Interface*). O módulos XBee e XBee-Pro trabalham, por *default*, no modo de operação transparente. Nesse modo, os módulos agem como um substituto da linha serial, ou seja, todos os dados recebidos através do pino de entrada (DI) são enfileirados até a transmissão RF. Quando dados são recebidos por RF, os dados são enviados ao pino de saída (DO).

Já o modo de operação API é uma alternativa ao modo transparente. Os pacotes API estendem o nível ao qual o microcontrolador, ou até mesmo um computador, pode interagir com os recursos de rede do módulo. Quando no modo API, todos os dados que entram ou deixam o módulo estão contidos em pacotes que define operações ou eventos dentro do módulo. Enfim, o modo API facilita várias operações tais como: transmitir dados para vários destinos sem entrar no modo de comando, identificar o endereço fonte de cada pacote recebido, saber o valor do RSS do sinal recebido, etc.

4.2 ADuC 842

O ADuC, desenvolvido pela *Analog Devices*®, é uma série de microcontroladores destinados a sistemas de instrumentação e aquisição de dados. Seu núcleo é baseado na arquitetura do 8052, sendo que mais recentemente a *Analog Devices*® tem desenvolvido seus microcontroladores ADuC com um núcleo ARM7 para atingir taxas maiores de MIPS (Mega Instruções Por Segundo) e, consequentemente, aumentar o desempenho.

A programação é feita em linguagem C utilizando-se o compilador *KEIL*^{TM}, que é compatível e dispões de recursos para programação, depuração e simulação não só para a família ADuC, mas para vários outro microcontroladores.

O microcontrolador utilizado foi o ADuC 842. Esse microcontrolador possui um núcleo otimizado do 8052 de até 20 MHz, único ciclo e 8 bits, oferecendo um desempenho máximo de 20 MIPS. Três tipos de memórias internas são disponíveis: memória de programa não volátil, que oferece até 62 kB; memória de dados não volátil, que oferece até 4 kB; e, uma memória RAM de 256 bytes. Integra um conversor analógico-digital (CAD) multicanal, autocalibrado e de alto desempenho, dois conversores digital-analógico, duas saídas PWM de 16 bits, um *watchdog timer*, três temporizadores/contadores e três portas seriais (SPI, I²C e UART).

5 Sistema de Aquisição de Dados

O sistema realizado foi utilizado para o levantamento das características elétricas do campo de futebol, de 90 x 56 m, localizado no complexo poliesportivo da UFCG – Campus Campina Grande, a ser usado na dissertação do aluno José Maurício Ramos de Souza Neto durante o desenvolvimento de um algoritmo de estimação de localização baseado no RSS.

Para isso, duas configurações de experimento foram feitas. A primeira refere-se a um nó móvel que se desloca de 10 em 10 metros em linha reta, percorrendo o campo – Figura 9. Nessa configuração, o nó móvel envia uma mensagem (composta da letra "A"), via *broadcast*, a cada 100 ms – Figura 9(a). A exceção do nó coordenador PAN, os nós âncoras verificam o RSSI da mensagem recebida e repassam-no ao nó coordenador PAN, que está ligado a um PC, para que esses valores sejam armazenados – Figura 9(b).

Já na segunda configuração, devem-se coletar o RSSI de um nó móvel recebido pelo nó coordenador com um deslocamento de 25 em 25 cm em linha reta, atravessando o campo. Porém, esse deslocamento é muito curto e, com isso, um suporte para conter quatro módulos de uma vez foi feito – Figura 11. Assim, foi feita a coleta de dados com relação aos quatro nós de uma única vez e o deslocamento necessário passou a ser de 1 em 1 m – Figura 10.

O sistema de aquisição é composto por uma parte em *hardware*, que compõe os nós móveis e âncoras, e por uma interface que coleta e armazena os dados conforme a configuração utilizada no momento. Todos os nós encontram-se a uma altura de 1 m do solo.

5.1 Hardware

Os nós são constituídos pelo módulo XBee-Pro e pelo ADuC 842 – Figura 12 . A escolha do XBee-pro deve-se as características do terreno em termos de suas dimensões. Esse conjunto é alimentado por baterias de 12 V, desde que há reguladores de tensão para obter os níveis adequados para o ADuC e o módulo XBee-Pro.

A comunicação entre o ADuC e o módulo XBee-Pro foi feita utilizando-se a UART à uma taxa de 19200 kbps.

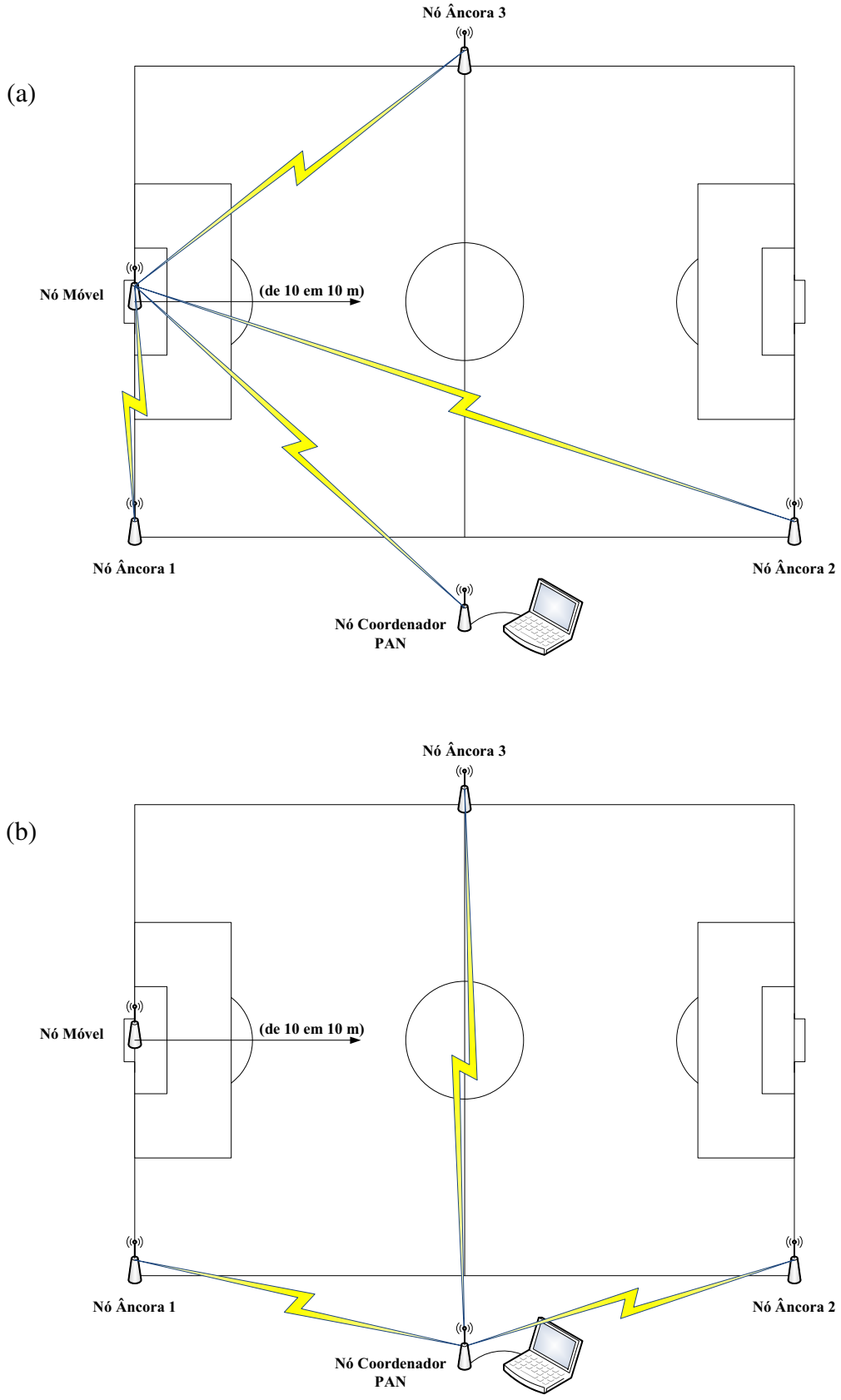

Figura 9 – Primeira configuração (a) Envio da mensagem *broadcast* do nó móvel (b) Transmissão do RSSI para o nó coordenador PAN

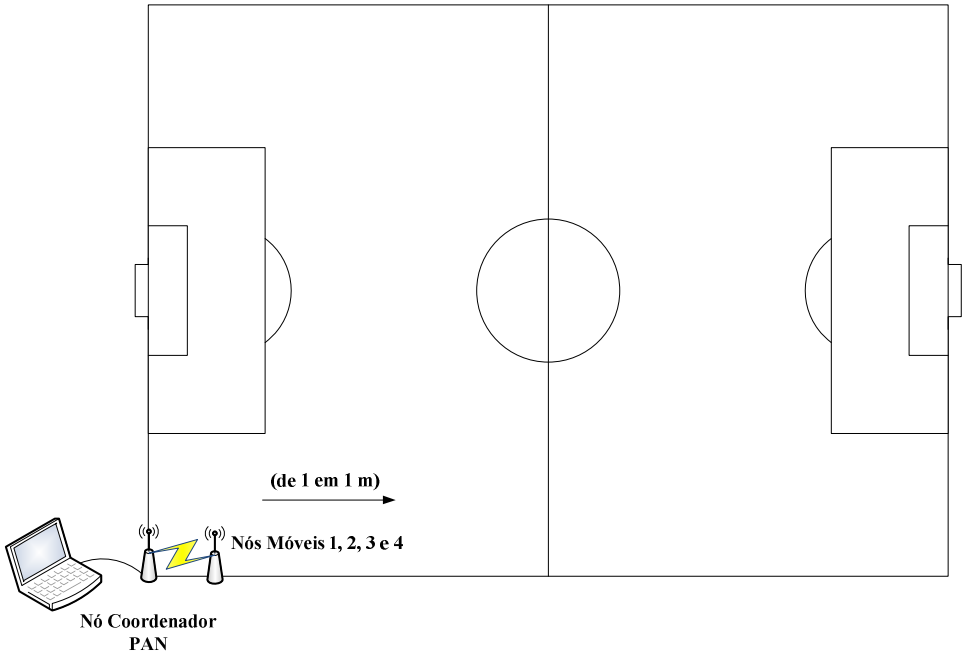

Figura 10 – Segunda configuração

Figura 11 – Suporte contendo quatro nós móveis

Figura 12 – Nó formado pelo ADuc e o XBee-Pro

Para gravar o *firmware* (contendo o padrão IEEE 802.15.4 mais as características de trabalho do módulo) no módulo XBee-Pro, utilizou-se o programa X-CTU. A versão do *firmware* usado foi o 10E6, disponibilizada no site da fabricante do módulo[1] e que é a versão mais atual até o momento. Ao gravar o *firmware*, somente um parâmetro das configurações do módulo era mudado, que é a taxa de comunicação serial, para que fosse possível a comunicação entre o ADuC e o XBee-Pro na taxa de comunicação desejada, já que o valor *default* de comunicação do módulo é de 9600 kbps.

Todas as outras configurações necessárias para o funcionamento do módulo XBee-Pro são feitas por meio do ADuC. Além de o ADuC ser responsável pela configuração dos módulos, ele também é responsável pelo envio das mensagens ou da recepção e repasse do valor RSSI para o nó coordenador, dependendo de como o ADuC foi programado para se comportar na rede formada (nó móvel ou nó âncora). Os códigos dos programas desenvolvidos para o ADuC encontram-se nos Anexos 9.1 e 9.2.

5.2 Interface com o Usuário

Para a coleta dos dados no computador, uma interface foi desenvolvida usando o *Borland C++ Builder* (BCB), que é um ambiente de desenvolvimento integrado (IDE – *Integrated Development Environment*) produzido pela *CodeGear*® para a escrita de programas na linguagem C++. O aplicativo inclui ferramentas que permitem

[1] http://www.digi.com/support/productdetl.jsp?pid=3257&osvid=0&s=268&tp=2

desenvolvimento visual através de "arrastar e soltar", tornando a programação mais simples.

Uma visão completa da interface do programa desenvolvido é vista na Figura 13.

Na seção “Configuração da Serial”, o usuário escolhe a porta de comunicação em que a placa CON-USBBEE (que conterá o nó coordenador PAN) está conectada e a taxa de comunicação entre a placa e o PC. Em “Porta” é mostrada a lista de portas seriais em que há uma conexão ativa. Em “Taxa” é mostrada uma lista contendo algumas taxas de comunicação serial, que por *default* é 19200 kbps. O botão “Abrir” abre/fecha a porta serial, estabelecendo/encerando a conexão. O botão “Atualizar”, que consta na opção do menu na parte superior da janela e que tem como tecla de atalho “F5”, atualiza a lista de portas para o caso de o usuário trocar a placa CON-USBBEE de porta. A conexão da porta serial será automaticamente encerrada, caso o usuário não faça isso, quando o programa for encerrado, evitando da porta de comunicação permanecer ativa sem que nenhum programa use-a.

Outra seção do programa é “Opções”. Nela define-se qual configuração de experimento é utilizada para que os dados sejam devidamente coletados, escolhendo-se ou “3 Aquisições” (que corresponde a primeira configuração) ou “4 Aquisições” (que corresponde a segunda configuração). Para o caso do usuário escolher “3 Aquisições” – Figura 13(a), as seções à esquerda “Valores do RSSI” e “Quantidade de Amostras” será formada por três janelas, uma para cada nó âncora, para que aja o acompanhamento dos valores adquiridos e da quantidade adquirida até o momento, respectivamente. O mesmo acontecerá caso seja escolhida a opção “4 Aquisições”, com a diferença que haverá quatro janelas, uma para cada nó móvel – Figura 13(b). A quantidade de amostras a serem obtidas é definida pelo usuário em “Quantidade de Amostras” dentro de “Opções”. Em “Ponto das Amostras”, é indicado o ponto onde está localizado o(s) nó(s) móvel(is), dependendo da configuração.

Ainda na seção “Opções”, há dois botões. O botão “Salvar”, que consta na opção do menu na parte superior da janela e que tem como tecla de atalho “CTRL+S”, salva os dados coletados em arquivos *txt*. O nome do arquivo será composto da seguinte forma: “**x**Aq_No**y**_Ponto_**wz**.txt”, onde **x** indica o tipo de aquisição; **y** indica qual o nó; **w** indica o ponto da amostra; e, **z** sinaliza a unidade (metros ou centímetros). O botão “Iniciar Aquisição” dá inicio a aquisição dos dados.

O código do programa encontra-se no Anexo 9.3.

(a)

(b)

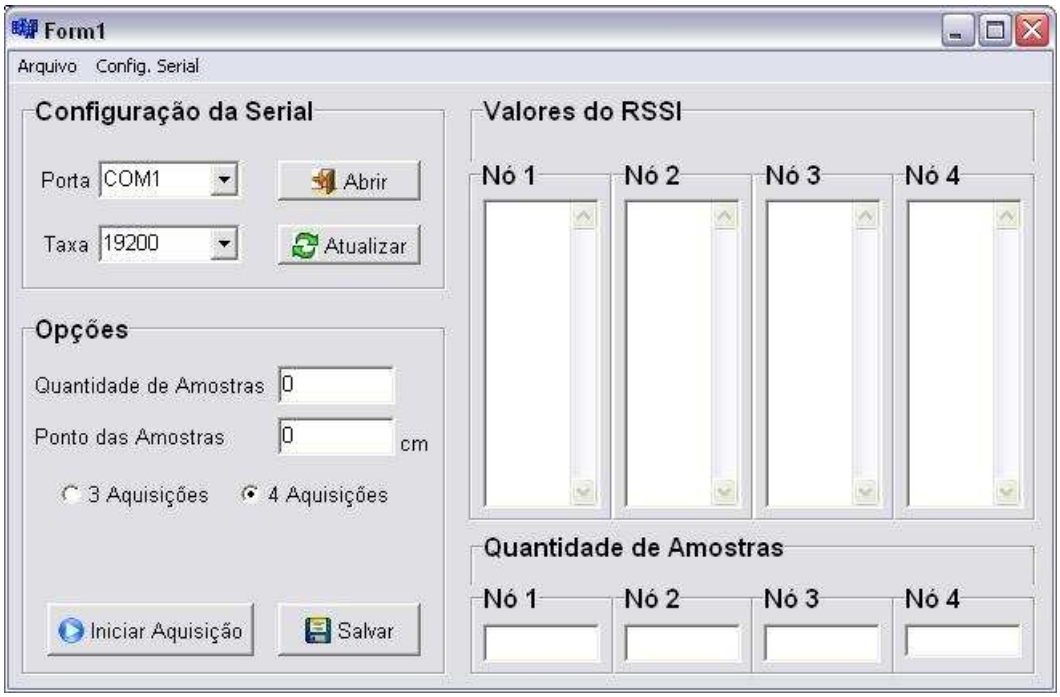

Figura 13 – Interface do programa (a) Primeira configuração (b) Segunda configuração

6 Resultados

Os resultados a seguir foram gerados e obtidos pelo aluno José Maurício para sua dissertação. Esses são mostrados aqui para mostrar e validar o sistema desenvolvido para a aquisição dos dados.

Para a primeira configuração, uma quantidade de 500 amostras foi coletada, enquanto que na segunda configuração foram 100 amostras.

O primeiro conjunto de dados serviu para validar e verificar a precisão do algoritmo de estimação de localização utilizado.

O segundo conjunto de amostras são utilizados para o levantamento das características elétricas do campo. Esse conjunto é visto na Figura 14.

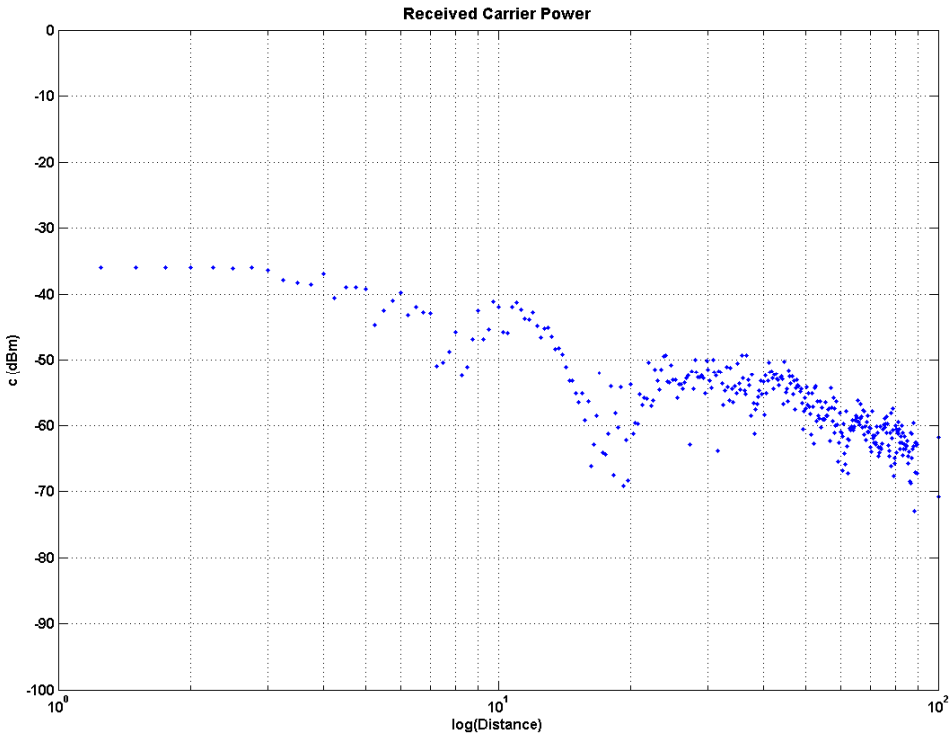

Figura 14 – Conjunto de dados obtidos da segunda configuração

Porém, como a potência de saída dos quatro nós móveis foi configurada para seu valor máximo, ou seja, 18 dBm. Assim, o efeito de pequenas diferenças na potência irradiada devido a desvios em relação ao valor nominal previsto pelo fabricante foi minimizado, por meio de uma rotina, de modo a normalizar a potência de saída de cada

módulo com um mesmo valor médio. Esse conjunto de dados, após o tratamento, é visualizado na Figura 15.

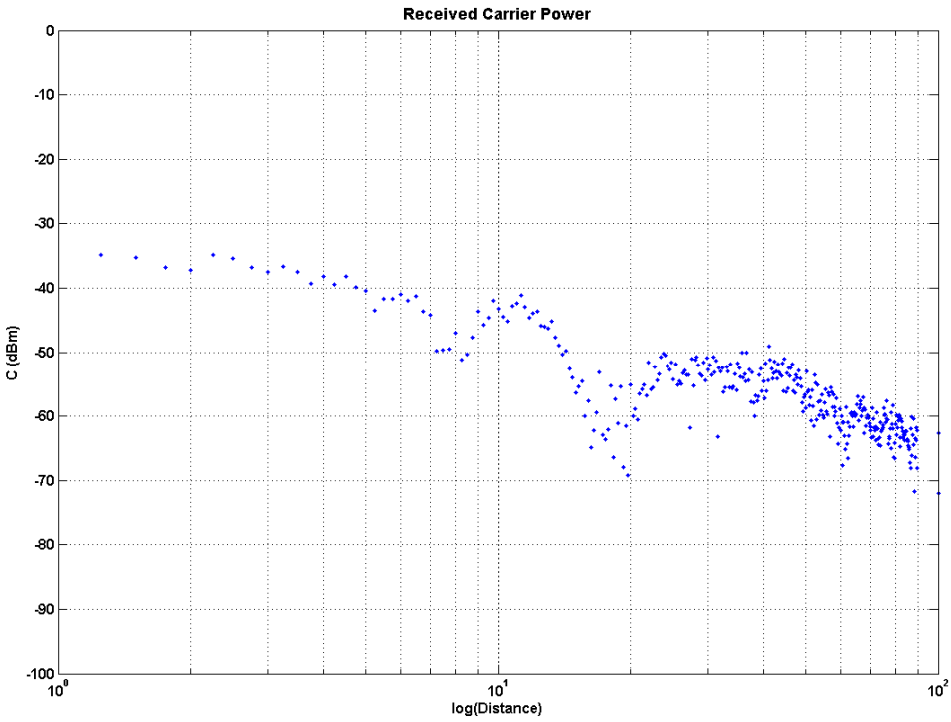

Figura 15 – Conjunto de dados após o tratamento das amostras

Os parâmetros elétricos encontrados foram: $\varepsilon_R = 1,0337$ e σ = -0,9794 S/m.

7 Conclusões

Nas primeiras tentativas de comunicar o ADuC com o módulo XBee/XBee-Pro, falhas ocorreriam na configuração do módulo. O problema foi devido ao não envio dos comandos respeitando o tempo necessário tanto para entrar no modo de comando como entre os comandos. Assim, uma rotina de atraso foi feita para o ADuC, já que o compilador não oferece nenhuma instrução que gerasse um atraso. Outro problema enfrentado no desenvolvimento do *hardware* foi com relação ao funcionamento do módulo XBee/XBee-Pro com relação ao *firmware* para a formação de uma rede sem fio. De início, usou-se uma versão antiga em que nem todos os parâmetros, tal como a quantidade de tentativas de envio do pacote, era fornecida. Ao acessar o site da fabricante, verificou-se uma versão mais nova onde *bugs*/novas características eram resolvidas/acrescentadas. Ao gravar os módulo XBee/XBee-Pro com esse novo *firmware* e configurando-o adequadamente, conseguiu estabelecer uma rede sem fio.

Com relação ao programa que fica hospedado no PC, um problema foi notado na primeira vez da coleta das amostras no campo. Após os dados serem coletados, observou-se que os dados não eram coerentes, já que os RSSI's não variavam da maneira esperada. Após alguns testes feitos no LIEC, percebeu-se que o erro era devido ao não esvaziamento do *buffer* da entrada serial do computador. Assim, quando era mandado coletar "novas amostras", na verdade o programa pegava amostras anteriores. Esse problema foi sanado com a introdução da instrução necessária para a limpeza do *buffer* de entrada no momento em que era requisitada uma nova aquisição.

Após as análises feitas pelo aluno José Maurício, conforme visto na seção Resultados, verificou-se que o sistema de aquisição de dados não apresentava problemas.

Tal sistema pode ainda ser melhorado com relação à estrutura do *hardware*, podendo-se desenvolver uma placa de circuito impresso que seja alimentada por pilha, tirando a necessidade de usar baterias e, com isso, diminuir os esforços realizados durante o experimento.

8 Bibliografia

1. **Farahani, Shahin.** *ZigBee Wireless Networks And Transceivers.* s.l. : Newnes, 2008. ISBN 978-0-7506-8393-7.

2. **LAN/MAN Standards Committee Of The IEEE Computer Society.** *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANS).* New York : IEEE, 2006. ISBN 0-7381-4997-7.

3. **Koubâa, Anis, Alves, Mário e Tovar, Eduardo.** *IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview.* Porto : IPP Hurray!, 2005.

4. **Ergen, Sinem Coleri.** *ZigBee/IEEE 802.15.4 Summary.* 2004.

5. **Silva, Luís Fabiano da.** *Investigação do Impacto da Mobilidade de Elementos na Rede IEEE 802.15.4 Através do Desenvolvimento de uma Plataforma de Simulação.* Campinas : PUC Campinas, 2008.

6. **Andrighetto, Eduardo.** *Sistema de Processamento de Sinais Biomédicos: Rede Wireless ZigBee com Aplicação do Padrão IEEE 802.15.4.* Florianópolis : UFSC, 2008.

7. **Digi International.** *XBee/XBee-Pro OEM RF Modules.* 2008.

8. **Analog Devices.** *Datasheet ADuC841/ADuC842/ADuC843.* Norwood : s.n., 2003.

9. **FunctionX.** Borland C++ Builder. [Online] [Citado em: 01 de Fevereiro de 2010.] http://www.functionx.com/bcb/index.htm.

10. **Neto, José Maurício Ramos de Souza.** *Sistema de Localização Baseado em Redes de Sensores se Fio.* Campina Grande : UFCG, 2010.

11. **IEEE 802.15.** IEEE 802.15 WPAN Low Rate Alternative PHY Task Group 4a (TG4a). [Online] [Citado em: 01 de Fevereiro de 2010.] http://grouper.ieee.org/groups/802/15/pub/TG4a.html.

9 Anexos

9.1 Programa do ADuC para os nós móveis

```
#include <ADuC842.h>
#include <stdio.h>

//Protótipos das funções
void configura_serial (void);
void configura_xbee (void);
void delay_ms (unsigned int atraso); //delay_ms = atraso * 1ms

//Programa principal
void main(void) {

    PLLCON = 0x80; //Configura para que o clock seja de 16,777 MHz
    T2CON = 0x00; //Configura Timer 2 no modo 16-bit autoreload

    TL2 = 0x76;   //Carrega no registrador RCAP2 o valor de 48758 (0xBE76) para que
    TH2 = 0xBE; //o atraso gerado pelo Time 2 seja cerca de 1ms

    RCAP2L = 0x76;   //O registrador RCAP2 contêm o valor de 48758 (0xBE76) para
    RCAP2H = 0xBE; //autocarregar os registradores TH2 e TL2 do Time 2

    configura_serial();
    configura_xbee();
    while(1) {
        printf("%c%c%c%c%c%c%c%c%c%c", 0x7E, 0x00, 0x06, 0x01, 0x5A, 0x00,
0x00, 0x01, 0x41, 0x62);
        delay_ms(100);
    }
}
```

```
//Configuração da serial
void configura_serial (void) {
    //Configuração do Timer 3 como gerador de BAUD RATE de 19200 kbps
    T3CON = 0x85;
    T3FD = 0x2D;
    SCON = 0x52;
}

//Configuração do módulo XBee
void configura_xbee (void) {
    //Entrando no modo de comando AT (+++)
    delay_ms(1100);
    printf("+++");
    delay_ms(1100);
    //Configurando o módulo como end device (ATCE0)
    printf("ATCE0,WR\n");
    delay_ms(50);
    //Configurando endereço fonte do módulo (ATMY0003)
    printf("ATMY0003,WR\n");
    delay_ms(50);
    //Configurando a string de identificação do nó (ATNIEND3)
    printf("ATNIEND3,WR\n");
    delay_ms(50);
    //Configurando o canal a ser escaneado (ATSC2)
    printf("ATSC2,WR\n");
    delay_ms(50);
    //Configurando o expoente de duração do scan (ATSD3)
    printf("ATSD6,WR\n");
    delay_ms(50);
    //Configurando o número de tentativas do módulo (ATRR3)
    printf("ATRR3,WR\n");
    delay_ms(50);
```

```
    //Configurando o expoente de back-off (ATRN1)
    printf("ATRN3,WR\n");
    delay_ms(50);
    //Configurando o power level (ATPL4)
    printf("ATPL4,WR\n");
    delay_ms(50);
    //Configura modo API tipo 1 (ATAP1)
    printf("ATAP1,WR\n");
    delay_ms(50);
    //Configurando a taxa de dados da serial (ATBD4)
    printf("ATBD4,WR\n");
    delay_ms(50);
    //Resetando o software para aplicar as modificações (ATFR)
    printf("ATFR\n");
    delay_ms(1000);
}

//Função para gerar atrasos em milisegundo(s)
void delay_ms (unsigned int atraso) {
    TR2 = 1; //Habilita a contagem do Timer 2
    while(atraso) {
      TF2 = 0;
      while(!TF2);
      atraso--;
    }
    TR2 = 0; //Desabilita a contagem do Timer 2
}
```

9.2 Programa do ADuC para os nós âncoras

```
#include <ADuC842.h>
#include <stdio.h>

//Variáveis
char mens[15];
unsigned char cont = 0;
unsigned char tam = 0;
unsigned char cks = 0;
unsigned char estado = 0;
unsigned char rssi = 0;

//Protótipos das funções
void configura_serial (void);
void configura_xbee (void);
void delay_ms (unsigned int atraso); //delay_ms = atraso * 1ms

//Programa principal
void main(void) {

    PLLCON = 0x80; //Configura para que o clock seja de 16,777 MHz
    T2CON  = 0x00; //Configura Timer 2 no modo 16-bit autoreload

    TL2 = 0x76; //Carrega no registrador RCAP2 o valor de 48758 (0xBE76) para que
    TH2 = 0xBE; //o atraso gerado pelo Time 2 seja cerca de 1ms

    RCAP2L = 0x76; //O registrador RCAP2 contêm o valor de 48758 (0xBE76) para
    RCAP2H = 0xBE; //autocarregar os registradores TH2 e TL2 do Time 2

    configura_serial();
    configura_xbee();
```

```
    while(1) {
        switch(estado) {
            case 0:
                while(!RI);
                if(SBUF==0x7E)
                    cont = 0;
                mens[cont] = SBUF;
                cont++;
                if(cont==7)
                    estado = 1;
                RI = 0;
                break;
            case 1:
                switch(mens[3]) {
                    case 0x81:
                        rssi = mens[6];
                        cks = 0xFF - (0x5C + rssi);
                    REP:  printf("%c%c%c%c%c%c%c%c%c%c",  0x7E,
0x00, 0x06, 0x01, 0x5A, 0x00, 0x00, 0x01, rssi, cks);
                        estado = 0;
                        break;
                    case 0x89:
                        if(mens[5]==0x00)
                            estado = 0;
                        else
                            goto REP;
                        break;
                }
                break;
        }
    }
}
```

```
//Configuração da serial
void configura_serial (void) {
    //Configuração do Timer 3 como gerador de BAUD RATE  de 19200 kbps
    T3CON = 0x85;
    T3FD = 0x2D;
    SCON = 0x52;
}

//Configuração do módulo XBee
void configura_xbee (void) {
    //Entrando no modo de comando AT (+++)
    delay_ms(1100);
    printf("+++");
    delay_ms(1100);
    //Configurando o módulo como end device (ATCE0)
    printf("ATCE0,WR\n");
    delay_ms(50);
    //Configurando endereço fonte do módulo (ATMY000?)-
    printf("ATMY0001,WR\n");
    delay_ms(50);
    //Configurando a string de identificação do nó (ATNIEND?)-
    printf("ATNIEND1,WR\n");
    delay_ms(50);
    //Configurando o canal a ser escaneado (ATSC2)-
    printf("ATSC2,WR\n");
    delay_ms(50);
    //Configurando o expoente de duração do scan (ATSD3)-
    printf("ATSD3,WR\n");
    delay_ms(50);
    //Configurando o número de tentativas do módulo (ATRR3)-
    printf("ATRR3,WR\n");
    delay_ms(50);
```

```
    //Configurando o expoente de back-off (ATRN1)-
    printf("ATRN1,WR\n");
    delay_ms(50);
    //Configurando o power level (ATPL4)-
    printf("ATPL4,WR\n");
    delay_ms(50);
    //Configura modo API tipo 1 (ATAP1)-
    printf("ATAP1,WR\n");
    delay_ms(50);
    //Resetando o software para aplicar as modificações (ATFR)
    printf("ATFR\n");
    delay_ms(1000);
}

//Função para gerar atrasos em milisegundo(s)
void delay_ms (unsigned int atraso) {
    TR2 = 1; //Habilita a contagem do Timer 2
    while(atraso) {
      TF2 = 0;
      while(!TF2);
      atraso--;
    }
    TR2 = 0; //Desabilita a contagem do Timer 2
}
```

9.3 Programa desenvolvido no BCB

```
#include <vcl.h>
#include <Registry.hpp>
#include <iostream>
#pragma hdrstop

#include "Programa.h"
```

```
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "VaClasses"
#pragma link "VaComm"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  //Identificando quais portas seriais estão ativas
  TRegistry *Reg = new TRegistry;
  TStringList *ComList = new TStringList;
  try {
    Reg->RootKey = HKEY_LOCAL_MACHINE;
    if(!Reg->OpenKey("HARDWARE\\DEVICEMAP\\SERIALCOMM", false))
      ShowMessage("Erro ao abrir chave");
    else {
      Reg->GetValueNames(ComList);
      for(int i = 0; i < (ComList->Count); i++)
        ComboBox1->Items->Add(Reg->ReadString(ComList->Strings[i]));
    }
  }
  __finally {
    delete Reg;
    delete ComList;
  }
```

```
  //Inicializando algumas variáveis
  Aux = 0;
  Aux2 = 0;
  Aux3 = 0;
  Aux4 = 0;
  IniAq = 0;
  ContDado = 0;
  TamDado = 0;

  //Configuração inicial da tela do programa
  Memo4->Visible = false;
  Edit4->Visible = false;
  GroupBox4->Visible = false;
  GroupBox11->Visible = false;
  GroupBox6->Width = (353-88);
  GroupBox7->Width = (353-88);
  this->Width = (657-88);
  this->Position = poDesktopCenter;

  //Setando os valores default da porta e da taxa da comunicação serial
  ComboBox1->ItemIndex = 0;
  ComboBox2->ItemIndex = 4;
  ComboBox1Change(0);
  ComboBox2Change(0);
}
//---------------------------------------------------------------------------

//Botão para abrir/fechar a porta de comunicação serial
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(!Aux) {
    VaComm1->Open();
    BitBtn1->Glyph->LoadFromFile("door01.BMP");
```

```
    BitBtn1->Caption = "Fechar";
    Aux = !Aux;
  }
  else {
    VaComm1->Close();
    BitBtn1->Glyph->LoadFromFile("door02.BMP");
    BitBtn1->Caption = "Abrir";
    Aux = !Aux;
  }
}
//---------------------------------------------------------------------------

//Botão para atualizar a lista de portas seriais que estão ativas
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  if(Aux == 1) {
    ShowMessage("Para atualizar a lista de dispositivos, é\n"
                "     necessário fechar a porta antes.");
  }
  else {
    //Identificando quais portas seriais estão ativas
    TRegistry *Reg = new TRegistry;
    TStringList *ComList = new TStringList;
    try {
      Reg->RootKey = HKEY_LOCAL_MACHINE;
      if(!Reg->OpenKey("HARDWARE\\DEVICEMAP\\SERIALCOMM", false))
        ShowMessage("Erro ao acessar chave de registro");
      else {
        ComboBox1->Items->Clear();
        Reg->GetValueNames(ComList);
        for(int i = 0; i < (ComList->Count); i++)
          ComboBox1->Items->Add(Reg->ReadString(ComList->Strings[i]));
        ComboBox1->ItemIndex = 0;
```

```
        }
      }
      __finally {
        delete Reg;
        delete ComList;
      }
    }
}

//---------------------------------------------------------------------------

//Escolha da porta de comunicação serial
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  if(Aux == 1) {
    ShowMessage("O parâmetro não pode ser modificado enquanto\n"
                "a porta de comunicação estiver aberta.");
    ComboBox1->ItemIndex = PortaIndex;
  }
  else {
    PortaIndex = ComboBox1->ItemIndex;
    VaComm1->DeviceName = ComboBox1->Text;
  }
}
//---------------------------------------------------------------------------

//Escolha da taxa de comunicação serial
void __fastcall TForm1::ComboBox2Change(TObject *Sender)
{
  if(Aux == 1) {
    ShowMessage("O parâmetro não pode ser modificado enquanto\n"
                "a porta de comunicação estiver aberta.");
    ComboBox2->ItemIndex = TaxaIndex;
```

```
  }
  else {
    TaxaIndex = ComboBox2->ItemIndex;
    switch(TaxaIndex) {
      case 0:
        VaComm1->Baudrate = br1200;
        break;
      case 1:
        VaComm1->Baudrate = br2400;
        break;
      case 2:
        VaComm1->Baudrate = br4800;
        break;
      case 3:
        VaComm1->Baudrate = br9600;
        break;
      case 4:
        VaComm1->Baudrate = br19200;
        break;
      case 5:
        VaComm1->Baudrate = br38400;
        break;
      case 6:
        VaComm1->Baudrate = br57600;
        break;
      case 7:
        VaComm1->Baudrate = br115200;
        break;
    }
  }
}
//---------------------------------------------------------------------------
```

```
//Chama a função que atualiza a lista de portas seriais que estão ativas
void __fastcall TForm1::Atualizar1Click(TObject *Sender)
{
  BitBtn2Click(0);
}
//---------------------------------------------------------------------------

//Finaliza o programa
void __fastcall TForm1::Fechar1Click(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------

//Configuração da tela para a aquisição de 3 dados
void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
  Aux2 = 0;
  Label5->Caption = "m";
  Memo4->Visible = false;
  Edit4->Visible = false;
  GroupBox4->Visible = false;
  GroupBox11->Visible = false;
  GroupBox6->Width = (353-88);
  GroupBox7->Width = (353-88);
  this->Width = (657-88);
  this->Position = poDesktopCenter;
}
//---------------------------------------------------------------------------

//Configuração da tela para a aquisição de 4 dados
void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
```

```
  Aux2 = 1;
  Label5->Caption = "cm";
  GroupBox6->Width = (353);
  GroupBox7->Width = (353);
  this->Width = (657);
  Memo4->Visible = true;
  Edit4->Visible = true;
  GroupBox4->Visible = true;
  GroupBox11->Visible = true;
  this->Position = poDesktopCenter;
}
//---------------------------------------------------------------------------

//Botão da inicialização/finalização de aquisição das amostras
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
  //Inicializa a aquisição
  if(!IniAq) {
    if(!VaComm1->Active())
      ShowMessage("Verifique se a porta serial foi aberta.");
    else {
      RadioButton1->Enabled = false;
      RadioButton2->Enabled = false;
      Edit5->Enabled = false;
      Edit6->Enabled = false;
      Memo1->Clear();
      Memo2->Clear();
      Memo3->Clear();
      Memo4->Clear();
      Edit1->Text = (Cont1 = 0);
      Edit2->Text = (Cont2 = 0);
      Edit3->Text = (Cont3 = 0);
      Edit4->Text = (Cont4 = 0);
```

```
        QuantAmost = StrToInt(Edit5->Text);
        VaComm1->PurgeRead();
        BitBtn4->Glyph->LoadFromFile("stop.BMP");
        BitBtn4->Caption = "Parar Aquisição";
        IniAq = !IniAq;
      }
    }
    //Finaliza a aquisição
    else {
      RadioButton1->Enabled = true;
      RadioButton2->Enabled = true;
      Edit5->Enabled = true;
      Edit6->Enabled = true;
      BitBtn4->Glyph->LoadFromFile("play.BMP");
      BitBtn4->Caption = "Iniciar Aquisição";
      IniAq = !IniAq;
    }
}
//---------------------------------------------------------------------------

//Função da leitura e tratamento dos dados recebidos pela serial
void __fastcall TForm1::VaComm1RxChar(TObject *Sender, int Count)
{
  if(IniAq == 1) {

    //Leitura do pacote API
    VaComm1->ReadChar(Dado[ContDado]);
    if(Dado[ContDado] == 126) { //126 == 0x7E
      Aux4 = 1;
      while(Aux4 == 1) {
        VaComm1->ReadChar(Dado[++ContDado]);
        if(ContDado == 2)
          TamDado = Dado[2] + 1;
```

```
  if(ContDado > 2) {
    TamDado--;
    if(TamDado == 0)
      Aux4 = 0;
  }
}
ContDado = 0;

//Se o pacote for do tipo RX de 16 bits de endereço, o valor do RSSI
//será lido
if(Dado[3] == -127) { //-127 == 0x81
  switch(Dado[5]) {
    case 1:
      if(Cont1 < QuantAmost) {
        if(Aux2 == 0)
          RSSI = Dado[8];
        else
          RSSI = Dado[6];
        Memo1->Lines->Add(-RSSI);
        Edit1->Text = ++Cont1;
      }
      break;
    case 2:
      if(Cont2 < QuantAmost) {
        if(Aux2 == 0)
          RSSI = Dado[8];
        else
          RSSI = Dado[6];
        Memo2->Lines->Add(-RSSI);
        Edit2->Text = ++Cont2;
      }
      break;
    case 3:
```

```
            if(Cont3 < QuantAmost) {
              if(Aux2 == 0)
                RSSI = Dado[8];
              else
                RSSI = Dado[6];
              Memo3->Lines->Add(-RSSI);
              Edit3->Text = ++Cont3;
            }
            break;
          case 4:
            if((Aux2 == 1) && (Cont4 < QuantAmost)) {
              RSSI = Dado[6];
              Memo4->Lines->Add(-RSSI);
              Edit4->Text = ++Cont4;
            }
            break;
        }

        //Finalização automática da aquisição dos dados quando a quantidade
        //total de amostras for atingida
        if((Cont1==QuantAmost) && (Cont2==QuantAmost) &&
(Cont3==QuantAmost) && ((Cont4 == 0)||(Cont4 == QuantAmost)))
          BitBtn4Click(0);
      }
    }
  }
}
//---------------------------------------------------------------------------

//Função que, ao finalizar o programa, garante que a porta serial seja fechada
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
  if(!VaComm1->Active())
```

```
    VaComm1->Close();
}
//---------------------------------------------------------------------------

//Salva as amostras adquiridas em arquivos do tipo txt
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  if(!Aux2) {
    Memo1->Lines->SaveToFile("03Aq_No1_Ponto_" + Edit6->Text + "m.txt");
    Memo2->Lines->SaveToFile("03Aq_No2_Ponto_" + Edit6->Text + "m.txt");
    Memo3->Lines->SaveToFile("03Aq_No3_Ponto_" + Edit6->Text + "m.txt");
    Edit6->Text = StrToInt(Edit6->Text) + 10;
  }
  else {
    Memo1->Lines->SaveToFile("04Aq_No1_Ponto" + Edit6->Text + "cm.txt");
    Memo2->Lines->SaveToFile("04Aq_No2_Ponto" + IntToStr(StrToInt(Edit6->Text)
+ 25) + "cm.txt");
    Memo3->Lines->SaveToFile("04Aq_No3_Ponto" + IntToStr(StrToInt(Edit6->Text)
+ 50) + "cm.txt");
    Memo4->Lines->SaveToFile("04Aq_No4_Ponto" + IntToStr(StrToInt(Edit6->Text)
+ 75) + "cm.txt");
    Edit6->Text = StrToInt(Edit6->Text) + 100;
  }
  ShowMessage("Os arquivos foram salvos");
}
//---------------------------------------------------------------------------

//Chama a função para salvar as amostras adquiridas
void __fastcall TForm1::Salvar1Click(TObject *Sender)
{
  BitBtn3Click(0);
}
//---------------------------------------------------------------------------
```