Monitorização de Provas de Educação Física

Filipe Edgar Sousa Santos

Novembro 2009

UNIVERSIDADE da MADEIRA

Monitorização de Provas de Educação Física

Dissertação submetida à Universidade da Madeira para obtenção do grau de Mestre em Engenharia de Telecomunicações e Redes

Filipe Edgar Sousa Santos

Dissertação realizada sob a supervisão de

Doutor Joaquim Amândio Rodrigues Azevedo

Professor Auxiliar do Centro de Competências de Ciências Exactas e Engenharias da Universidade da Madeira

Novembro 2009

RESUMO

O presente trabalho tem por objectivo desenvolver um sistema de monitorização de provas de educação física e do ambiente circundante.

Neste projecto desenvolveu-se um protótipo para uma Rede de Sensores Sem Fios (RSSF) que realiza a monitorização, em tempo real, do esforço e desempenho da actividade física dos atletas e das características físicas do ambiente.

Estudou-se o funcionamento das RSSF, baseadas no protocolo *ZigBee*, e foram desenvolvidos módulos de monitorização de atletas e ambiente que integram esta tecnologia como meio de comunicação. O módulo de monitorização de atletas é composto por acelerómetro, sensor de batimento cardíaco e GPS. Inclui um serviço de localização secundário a partir do *received signal strenght indicator* (RSSI) caso o serviço de GPS estiver indisponível. O módulo de monitorização ambiental é composto por vários sensores que monitorizam: humidade, temperatura, luminosidade, monóxido de carbono, dióxido de carbono e oxigénio. Cada módulo de monitorização ambiental foi munido com *Bluetooth*, por forma a que os atletas, sempre que no alcance da rede, possam com o próprio telemóvel consultar o valor actual dos parâmetros ambientais e a sua localização.

Estes dados são medidos e transmitidos periodicamente, em tempo real, pela rede *ZigBee* para uma estação base acoplada a um computador. Os dados são armazenados e processados e os resultados são disponibilizados através de uma aplicação no computador local e de uma página na Internet.

Neste trabalho verifica-se que a RSSF, que utiliza o protocolo *ZigBee*, é capaz de realizar comunicação entre atletas, sensores ambientais e computador com baixo consumo energético, optimizando a autonomia pretendida. Este sistema de RSSF integrado com a tecnologia sensorial actual, permite o desenvolvimento de módulos com um elevado nível de funcionalidades em dimensões relativamente reduzidas.

Palavras chave: *ZigBee*, RSSF, Monitorização, Ambiente, Atletas, Sensores, Baixo Consumo, *Bluetooth*.

ABSTRACT

This study aimed to develop a system to monitor athletes during their physical exercise and the surrounding environment.

In this project we developed a prototype for a Network of Wireless Sensors (WSN) that performs the monitoring, in real time, of athletes physical activity performance and the surrounding environment parameters.

We studied the operation of WSN based on the ZigBee protocol, and it was developed monitoring modules of athletes and environment that use this technology as a means of communication. The module for monitoring athletes consists of accelerometer sensor, heart rate sensor and GPS. It includes a secondary system for localization that results from the measure of the received signal strength indicator (RSSI) in the case that GPS signal is not available. The environmental monitoring module includes several sensors to monitor physical parameters: humidity, temperature, luminance, carbon monoxide, carbon dioxide and oxygen. Each environment sensor node was provided with Bluetooth, so that the athletes, where the range of the network can with their own mobile phone check information such as the present value of environmental parameters and their localization.

These data are measured and sent periodically, in real time, through the ZigBee network to a base station coupled to a computer. These data are stored and processed for presentation on the computer and Internet.

This study verifies that WSN based on ZigBee is capable of interconnect athletes, environmental sensors and the computer with low-power requirements. This nWSN network coupled with the current sensory technology, allows the creation of modules with a high level of integration of features, while maintaining small size.

Keywords: ZigBee, WSN, Monitoring, Environment, Athletes, Sensor, Low Power, Bluetooth.

AGRADECIMENTOS

Tenho que agradecer a várias pessoas pela ajuda e disponibilidade ao longo do desenvolvimento deste projecto. Começo por agradecer ao meu orientador, Professor Joaquim Amândio Azevedo.

Agradeço o apoio financeiro aportado pela bolsa de estudo fornecida pelo Centro de Ciências e Tecnologia da Madeira.

Um agradecimento especial é devido ao contributo dos seguintes colegas, sem nenhuma ordem em particular: Alexandre, Adrian, Aguiar, Hugo, Leonardo, Tiago, Tony e os Professores Dionísio Barros e Gabriel Lira.

Agradeço aos meus amigos, que foram perguntando pelo trabalho e suportaram as minhas ausências.

Agradeço aos meus pais e família, pelo apoio incondicional que me sempre deram.

Em especial tenho de agradecer à Catarina, companheira, namorada e amiga que me apoiou, suportou as minhas faltas e que me foi ajudando dentro do que lhe era possível.

Gostaria por fim agradecer a todos aqueles que dando as suas ideias e/ou críticas, ajudaram no desenvolvimento deste trabalho.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ACRÓNIMOS E ABREVIATURAS

ADC – Analog to Digital Converter
API – Application Protocol Interface
AT – significa attention – do padrão de comandos Hayes
CO – Monóxido de carbono
CO2 – Dióxido de carbono
CSMA/CD– Carrier Sense Multiple Access with Collision Detection
DSSS – Direct Spread Spectrum Sequence
FFD – Full Function Device
GPS – Global Positioning System
IDE – Integrated Development Environment
IEEE – Institute of Electrical and Electronics Engineers
ISM – Industrial, scientific and medical
LQI – Link Quality Indicator
MAC – Medium Access Control ou Message Authentication Code
MANET – Mobile Ad Hoc Network
NTW – Network
Q-PSK – Quadrature phase-shift keying
O2 – Oxigénio
PHY – Physical
PWM – Pulse Width Modulation
QoS – Quality of Service
RFD – Reduced Function Device
RSSF – Redes de Sensores Sem Fios
RSSI – Received Signal Strength Indication
RTC – Real Time Clock
SMAC – Sensor MAC
TDMA – Time Division Multiplexing
USART – Universal Synchronous/Asynchronous Receiver/Transmitter
WPAN – Wireless Personal Area Network
WSN – Wireless Sensor Network

1 INTRODUÇÃO

1.1 Motivação

A monitorização de atletas é de grande importância, para a saúde do atleta e seu desempenho durante a actividade desportiva.

Inicialmente a optimização do desempenho dos atletas implicava um treino personalizado, ou seja, um treinador por atleta. Quando se trata de uma equipa ou turma numa aula de desporto não é prático ter um treinador por atleta, mas continua a ser importante conhecer as características comuns e individuais de cada um por forma ao treinador poder tomar as melhores decisões.

Os registos da velocidade, altura de salto, entre outros eram realizados manualmente. O ritmo cardíaco era registado antes, a meio e no fim do treino. Isto era realizado com recurso a cronómetros analógicos, fita métrica e à informação transmitida pelo próprio atleta. Estes perfaziam os meios que os treinadores disponham para obter dados, para análise e tomada de decisões.

A necessidade de comparar os diferentes dados entre atletas implicava um grande esforço para analisar as tabelas de registos manualmente, diminuindo o tempo disponível para treino. A informatização dos vários dados em folhas de cálculo e visualização dos dados em gráficos permitiram simplificar e acelerar o processo de avaliação e tomada de decisões. Com o aparecimento dos Electrocardiogramas passou a ser possível registar o ritmo cardíaco com maior precisão e mais informação. A tecnologia inicial obrigava ao atleta estar confinado num espaço reduzido, para que pudesse estar ligado aos diversos equipamentos.

A evolução da tecnologia nas áreas sensoriais, principalmente no aumento da velocidade de resposta, precisão e redução das dimensões, permitiu ao atleta começar a transportar o equipamento de monitorização durante a actividade e consultar os dados em tempo real. De seguida passou a ser possível o registo destes dados para posterior análise. Com a chegada das redes sem fios como as *Wireless Personal Area Network* (WPAN) têm permitido o desenvolvimento de pequenos equipamentos contendo os diversos sensores de monitorização e um pequeno rádio. Este envia a informação em tempo real para um computador central, durante a prática de exercício físico. No computador os dados são processados e apresentados em tempo real ao treinador. Sendo esta uma das aplicações possíveis para as Redes de Sensores Sem Fios (RSSF).

Actualmente, a monitorização fisiológica em tempo real de atletas está chegando a um nível que leva aos treinadores de desportos de elite procurar, cada vez mais, a fórmula optimizada para o maior desempenho do atleta, evitando em paralelo lesões no atleta, e aumentando a disponibilidade dos mesmos.

É também interessante medir valores de poluição do ambiente de treino. Estes factores afectam a qualidade de vida do atleta e o seu desempenho. Particularmente no meio urbano onde geralmente a concentração de poluentes ambientais é superior.

Vários estudos recentes têm aplicado a tecnologia de RSSF à monitorização ambiental. Tipicamente os projectos focam o desenvolvimento do programa de gestão da rede, da agregação de dados e das técnicas de redes de larga escala. Contudo, foram encontrados poucos estudos que lidem com os problemas práticos da implementação do próprio módulo sensor, que consistem nas características dos sensores a utilizar, como o seu funcionamento, dimensões, consumo energético e integração com a rede.

E por último o desenvolvimento de pequenos dispositivos computacionais, pequenos, baratos e com meios de comunicação sem fios, têm permitido pôr em prática o paradigma das redes ubíquas. Redes em que os dispositivos no meio interagem consoante o utilizador de uma forma transparente e inteligente.

Ligar a monitorização de atletas a uma rede sem fios, permitindo ainda a avaliação das condições ambientais adequadas à actividade física, em tempo real, abre portas para novas metas na monitorização de atletas.

1.2 Objectivos

O principal objectivo deste trabalho é desenvolver uma rede para monitorizar a actividade da actividade física de atletas e do ambiente circundante. Identificando os problemas associados à utilização das redes de sensores sem fios neste tipo de aplicações.

Tem-se assim como metas:

- Implementar uma rede de sensores sem fios (de baixo custo), com módulos disponíveis no mercado, para a monitorização de parâmetros de atletas e do meio ambiente de uma prova de educação física;
- Utilizar sensores disponíveis no mercado com a melhor relação entre dimensão, precisão e custo;
- Encaminhar os valores monitorizados para uma base de dados, e disponibilizá-los em tempo real no computador e numa página de Internet;
- Permitir a visualização de valores monitorizados no telemóvel, incluindo informação da localização do módulo sensor.

1.3 Organização da Dissertação

Nesta dissertação apresentou-se um estudo das redes de sensores sem fios e a implementação de um sistema de monitorização de atletas. A dissertação encontra-se estruturada em seis capítulos. No primeiro capítulo fez-se uma abordagem ao tema e à motivação para o trabalho desenvolvido.

No segundo capítulo apresenta-se um apanhado do estado actual das redes de sensores sem fios e aborda-se os sistemas existentes, bem como, os parâmetros monitorizados.

No terceiro capítulo expõe-se as experiências realizadas com *kits* de desenvolvimento *ZigBee* de três marcas disponíveis comercialmente. Para estes *kits* desenvolveu-se funcionalidades como: armazenamento de dados, visualização de RSSI e interface *Web*. Numa primeira fase estudou-se as RSSF e numa segunda fase avaliou-se os pontos fortes e fracos de cada *kit*. Comparou-se ainda as diversas antenas disponíveis, e estudou-se o comportamento do sinal em ambientes reais.

O quarto capítulo apresenta o trabalho realizado sobre o desenvolvimento do sistema de rede de sensores sem fios para a monitorização de atletas. Começa-se por definir os requisitos para o sistema e a arquitectura proposta para o mesmo. De seguida, descreve-se o funcionamento dos principais componentes, identificando as características que levaram à sua escolha, e a forma de realizar a sua integração no sistema. A ordem de exposição segue a ordem de desenvolvimento prático dos elementos do sistema. Em primeiro lugar desenvolveu-se o módulo de monitorização ambiental do espaço de treino. Em segundo, implementou-se o módulo de monitorização do atleta. Finalmente, descreve-se as interfaces desenvolvidas para o sistema.

No capítulo quinto exibem-se os resultados obtidos, apontam-se possíveis optimizações ao sistema, e definem-se as características de funcionamento do sistema desenvolvido.

Finalmente, o capítulo sexto onde apresentam-se as conclusões ao trabalho desenvolvido, e o trabalho a realizar em futuros desenvolvimentos.

2 REDE DE SENSORES SEM FIOS E MONITORIZAÇÃO

Este capítulo encontra-se dividido em duas partes. Apresenta-se, inicialmente, um estudo sobre as redes de sensores sem fios englobando a arquitectura, topologia e protocolos. Deu-se especial ênfase ao estudo do protocolo *ZigBee*, por se destacar como o protocolo mais utilizado pelos vários fabricantes de RSSF. Apresenta-se um estudo de mercado de módulos que utilizam este protocolo. Descreve-se os sistemas de monitorização actuais com base nos produtos existentes no mercado. Finaliza-se com a descrição dos parâmetros que pretendemos monitorizar.

2.1 Redes de Sensores Sem Fios

Uma Rede de Sensores Sem Fios é uma rede, com topologia e tamanho dinâmico e variável, constituída por um conjunto de pequenos sistemas autónomos com ligação sem fios, denominados de nós sensores que cooperam na monitorização de condições físicas ou ambientais, como temperatura, som, vibração, pressão, movimento ou poluição em diferentes locais. A ideia de rede de sensores foi tornada popular pela Universidade de Berkeley que desenvolveu uma série de nós sensores chamados de *mica nodes* [1].

2.1.1 Componentes e Arquitectura das RSSF

Numa rede de sensores, os nós sensores estão dispersos por uma área física a monitorizar. Cada nó sensor disperso possui a capacidade para recolher informação e para reencaminhamento desta até ao nó de recolha de dados (*sink node*, também conhecido como *gateway*). Os dados são enviados para o nó *gateway* através de uma comunicação *multihop* como é apresentado na figura 2.1. O nó *gateway* pode comunicar com o gestor da tarefa via internet, satélite ou até mesmo ligado directamente [2]. O gestor da tarefa pode ser um *software* a correr num computador de utilizador. O desenho da rede de sensores é influenciado por vários factores, discutidos no ponto 2.1.6.

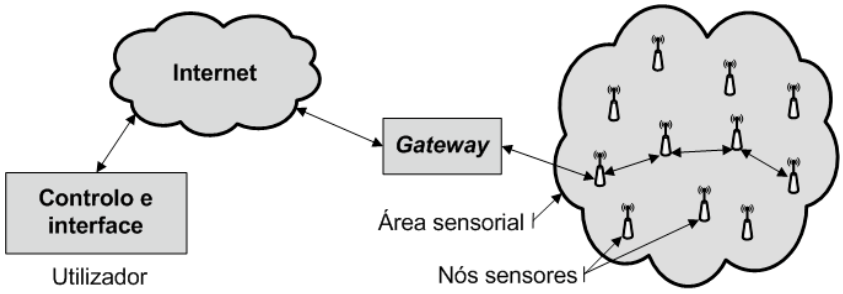


Figura 2.1 – Exemplo da arquitectura de uma rede de sensores sem fios.

Cada nó, além de ter um ou mais sensores, é tipicamente constituído por um rádio, um pequeno microcontrolador e uma fonte de energia, usualmente uma bateria. Na figura 2.2 apresenta-se a arquitectura geral de um nó de uma rede de sensores sem fios.

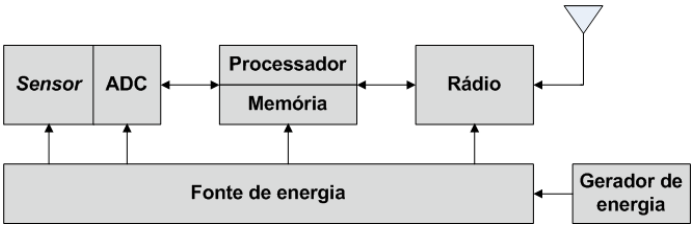


Figura 2.2 – Arquitectura de um nó sensor.

O tamanho de cada nó pode variar desde a dimensão de uma caixa de sapatos até ao de um grão de areia, embora nós de tamanhos microscópicos ainda estão para ser desenvolvidos. O custo de cada nó é similarmente variável desde centenas de euros até dezenas cêntimos, dependendo da complexidade requerida por cada sensor. Restrições no tamanho e custo de cada nó dependem directamente dos requisitos de memória, velocidade de processamento, largura de banda e da fonte de energia [3].

2.1.2 Topologias para RSSF

Os nós das RSSF estão tipicamente organizados em três tipos de topologias de rede, topologia em estrela, árvore e malha, como é apresentado na figura 2.3 [4].

A topologia em estrela consiste naquela em que cada nó se liga directamente à *gateway*. Na topologia em árvore cada nó liga-se a um nó num nível superior na árvore e o último à *gateway*, e os dados são reencaminhados desde o nó do nível mais baixo da árvore até à *gateway*. Finalmente, para aumentar a fiabilidade, a topologia de rede malha que contém nós que conseguem conectar-se com múltiplos nós no sistema e passar dados pelo caminho mais fiável disponível. A topologia malha, apresentando múltiplos caminhos de reencaminhamento entre os nós, é tolerante a falhas de nós individuais ou caminhos. Uma vantagem adicional das redes em malha é que, embora todos os nós têm de ter as mesmas características de *hardware*, certos nós podem possuir funções adicionais (considerados nós principais) para a gestão da rede. Caso algum falhe outro nó pode substituir as suas funções, apresentando-se como uma vantagem em relação à topologia em estrela que tem apenas um nó principal. Este nó é um ponto crítico de risco de falha de rede. As redes em malha são geralmente apropriadas para redes de sensores sem fios de grande escala distribuídos por uma área geográfica [4] [5].

Para permitir os fabricantes comercializarem dispositivos ao mais baixo custo. O padrão da IEEE define dois tipos de dispositivos: os *Full Funciton Device* (FFD) e *Reduced Function Device* (RFD). Os FFD têm as funções de reencaminhamento de mensagens na rede podendo ser nós coordenadores ou nós routers, podem funcionar em qualquer topologia e podem falar com qualquer dispositivo. Os RFD são de implementação simples e não têm qualquer função de reencaminhamento de mensagens, são nós terminais que apenas recolhem e transmitem a informação dos sensores para um coordenador de rede. O coordenador é o responsável pela criação da rede [6].

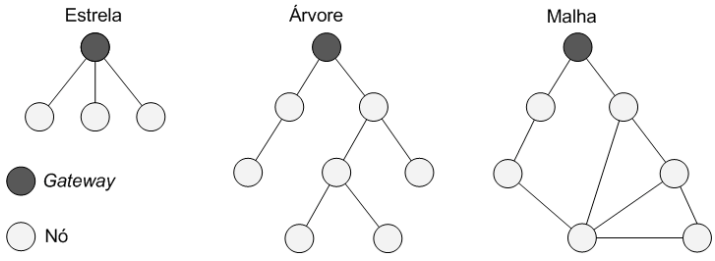


Figura 2.3 – Topologias básicas de redes de sensores sem fios.

É possível, numa só rede de sensores sem fios, usar combinações destas topologias tendo em conta as vantagens para a aplicação pretendida.

2.1.3 Protocolos para as RSSF (802.15.4)

Bluetooth e *Mobile Ad Hoc Network* (MANET, exemplo: *Wi-Fi*, redes de telemóveis) são provavelmente os protocolos mais próximos das redes de sensores sem fios. *Bluetooth* é um sistema sem fios de curto alcance com intenção de substituir os cabos entre equipamentos electrónicos e terminais de utilizador por ligações RF. A topologia *Bluetooth* é a estrela onde um nó "mestre" pode ter até sete nós "escravos" ligados via RF a ele, de maneira a formar uma *piconet*. A potência de transmissão é à volta dos 20 dBm e o alcance em torno das dezenas de metros [2]. Na figura 2.4 apresenta-se exemplos de aplicações do *Bluetooth*, adaptado de [7].

Figura 2.4 – Exemplos de aplicações *Bluetooth*.

O protocolo MAC numa MANET tem a tarefa de formar uma rede e mantê-la em face da mobilidade. O primeiro objectivo é a provisão de alto QoS sob condições de mobilidade. Os nós são alimentados a baterias. Estas podem ser trocadas ou recarregadas pelo utilizador, e logo o consumo de energia é de importância secundária. Em contraste com estes sistemas, as redes de sensores sem fios têm maior número de nós. Logo trocar a bateria a todos os nós da rede é uma tarefa de grandes dimensões. A potência de transmissão dos nós das redes de sensores sem fios ronda os 0 dBm e o alcance de um nó sensor é muito menor que numa MANET. A troca de topologia é mais frequente numa rede de sensores sem fios devido à mobilidade dos nós e possíveis falhas. Na maioria das aplicações é esperado que a mobilidade seja menor que nas redes MANET. Em essência o ponto de maior importância numa rede de sensores sem fios é a conservação de energia para prolongar o tempo de vida na rede, e tendo em conta a necessidade de nós *ad hoc* que se auto-organizam, demonstra que os protocolos já existentes, *Bluetooth* ou os protocolos MAC das MANET, não se adequam às redes de sensores sem fios [2].

A crescente importância de dispositivos baratos baseados em redes sem fios exige uma plataforma comum de forma que os dispositivos possam comunicar entre si e partilhar componentes a um baixo custo.

O grupo de trabalho da norma 802.15 do IEEE desenvolveu o protocolo 802.15.4, que é um padrão que especifica a camada Física e MAC para WPANs [5]. Este protocolo tem como objectivo ter aplicações ambientais, médicas, multimédia, monitorização de edifícios entre outros. Um dos focos do protocolo 802.15.4 é apresentar maior eficiência energética em relação ao 802.11 TDMA e SMAC [8].

De forma breve descreve-se de seguida os pontos principais da norma 802.15.4:

Opera em três bandas de rádio conhecidas como ISM (*Industrial, Scientifical and Medical*), as quais estão isentas de licenciamento. Globalmente, corresponde à banda de 2,4 GHz e ainda às bandas de 915 MHz (Estados Unidos) e 868 MHz (Europa). Consoante a banda, varia a taxa de transmissão possível: em 2,4 GHz podem ser obtidas taxas de transmissão até 250 Kbps, com 16 canais disponíveis; a 915 MHz, está disponível uma taxa de transmissão até 40 Kbps e 10 canais de comunicação; no caso de 868 MHz, possibilita 1 canal e uma taxa de transmissão de apenas 20 Kbps.

É utilizada a técnica de espalhamento espectral, DSSS (*Direct Sequence Spread Spectrum*), para lidar com uma das limitações das transmissões rádio, que é o desvanecimento. Esta técnica oferece adicionalmente um incremento da segurança no canal de comunicação, dada a utilização de uma chave de codificação dos dados, partilhada entre emissor e receptor.

Em termos de modulação, é utilizado o O-QPSK (*Offset quadrature phase shift keying*) para a banda dos 2,4 GHz e BPSK (*binary phase shift keying*) para os 915 ou 868 MHz [62].

Para controlar uma das principais funções da camada MAC, controlar o acesso ao meio sem fios, o protocolo usa o algoritmo CSMA/CA (*Carrier Sense Multiple Access/ Colission Avoidance*).

No padrão 802.15.4 são especificados e identificados os dois tipos de dispositivos das RSSFs: os FFD e RFD; e dois tipos de topologias: a estrela e estrela agregada (conhecida como *clustered star*).

O formato de uma mensagem genérica consiste num cabeçalho da camada física, um cabeçalho da camada MAC, dados que são entregues à camada MAC e por fim um *byte* de *checksum* da mensagem. São definidos quatro tipos de mensagens pela especificação 802.15.4: *Beacom, Data, Command, Acknowledge*. Estes tipos de mensagem são identificados no *Frame Control Field*, localizado nos primeiros dois bytes do cabeçalho MAC. Na figura 2.5 apresenta-se a estrutura da mensagem 802.15.4 [9].

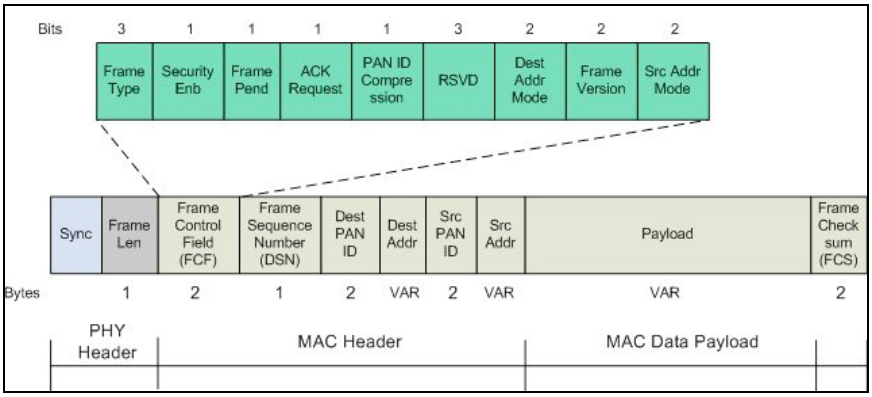


Figura 2.5 – Formato de mensagem do 802.15.4.

Existem dois tipos de endereço no 802.15.4: o endereço estendido com 64 *bits*, sendo usado como um identificador único universal para o dispositivo (parecido com o endereço MAC das placas de Ethernet), e o endereço curto constituído por 16 *bits*, sendo único na rede em que o dispositivo juntou-se.

O padrão 802.15.4 tem como principal alvo as aplicações de monitorização e controlo. O baixo consumo de potência é a característica mais importante nesta norma, que permite dispositivos que operam a baterias funcionar por longos períodos de tempo. A quantidade de *throughput* é relativamente baixa quando comparado, por exemplo, com a norma 802.11, mas

com 250 kbps é mais do que suficiente para muitas aplicações. A distância entre dois nós pode ir até os 100 metros, o que permite ter uma rede a cobrir perímetros e distâncias significativas, especialmente porque permitem o reencaminhamento de mensagens [9].

Esta norma, actualmente na versão 2006, é a base das especificações *ZigBee*, *WirelessHART* e *MiWi* que tentam oferecer uma solução completa de rede através do desenvolvimento das camadas superiores que não estão abrangidas pela norma.

O protocolo *ZigBee*, mantido pela *ZigBee Aliance*, é das primeiras especificações e a mais utilizada, que já é comercializada por muitos fabricantes [6].

2.1.4 Protocolo *ZigBee*

Em poucas palavras: Baixo custo, baixa potência, diversas possibilidades [6].

ZigBee é um protocolo de redes sem fios que está orientado para o controlo remoto e para aplicações sensoriais. É adequado para operações em ambientes rádio hostis, e para localizações isoladas. A responsável pelo aparecimento desta tecnologia foi a empresa Philips, sendo nessa época designado por protocolo *Home RF Lite*. Foi definido e mantido actualmente por uma aliança de várias empresas denominada *ZigBee Alliance*. O *ZigBee* é construído sobre o padrão 802.15.4 que define as camadas física e MAC. Por cima destas, o *ZigBee* define a camada de rede e segurança, e a de aplicação, permitindo a interoperabilidade entre produtos de diferentes fabricantes. Desta forma o *ZigBee* é uma tecnologia que maximiza as especificações da norma 802.15.4. A figura 2.6 apresenta a composição da arquitectura do protocolo *ZigBee*, baseada em [10].

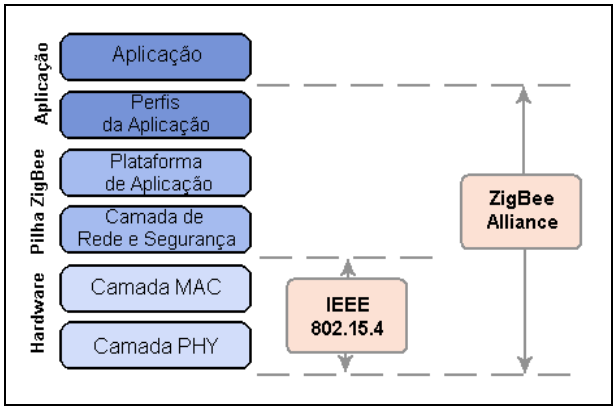


Figura 2.6 – Arquitectura geral do *ZigBee*.

A camada de rede começa por definir os diferentes tipos de dispositivos de rede, que no *ZigBee* são denominados por *ZigBee Coordinator* (ZC), *ZigBee Router* (ZR) e de *ZigBee End Device* (ZED). Faz a gestão de endereçamento e encaminhamento de mensagens. É ainda responsável pela implementação de segurança, onde são fornecidos quatro serviços: controlo de acessos, encriptação de dados, integração de tramas e controlo de sequência de mensagens. A camada de aplicação controla o módulo conforme o tipo de dispositivo *ZigBee* e providência o interface às funcionalidades do dispositivo à aplicação final [10].

2.1.5 Aspectos da Criação de uma RSSF

O estado da arte da tecnologia de redes sensores sem fios, oferecem soluções para desenhar e desenvolver muitos tipos de aplicações [11]. Na figura 2.7, adaptada de [12], apresenta-se as principais áreas de aplicação das RSSF. Ao criar aplicações baseadas nas RSSF, identificam-se uma série de requisitos, aos quais as redes actualmente, ainda não

satisfazem por completo. Situação pela qual as redes de sensores são tema de vários trabalhos de investigação nos últimos anos [11].

Figura 2.7 – Conjunto de aspectos no funcionamento de uma RSSF.

Assim, ao criar-se uma RSSF é necessário ter atenção a vários aspectos da rede para garantir a sua robustez. As aplicações usam frequentemente um controlo embutido por microcontroladores de 8 ou 16 bit, com o intuito de operar sem a intervenção humana por períodos de tempo que atingem anos. Este longo período de operação sem intervenção humana em situações de baixo consumo energético trazem duas grandes implicações para a projecção deste sistemas: O *software* que opera nestes dispositivos deve ser simples e exacto e os dispositivos devem fazer a gestão da energia que possuem da melhor forma possível. O circuito integrado de comunicação sem fios é a parte que mais deve ter em atenção a estes pormenores, uma vez que o próprio e o microcontrolador representam as duas maiores fontes de consumo energético. Além do consumo, é preciso ter em conta vários aspectos no desenho de uma rede de sensores sem fios para uma determinada aplicação, nomeadamente: distribuição, mobilidade, custo, tamanho, recursos energéticos, heterogeneidade, meios de comunicação, segurança, infra-estrutura, topologia de rede, cobertura, conectividade, tamanho da rede, tempo de vida, serviços e outros parâmetros de QoS [3].

2.1.6 Módulos no Mercado

Actualmente no mercado são diversas as opções de rádios e módulos *ZigBee* para iniciar o desenvolvimento com redes de sensores sem fios. Estes podem ser divididos em três categorias dependendo do nível de integração:

- **Rádios transceptores *ZigBee*** (*transceivers*) – Módulo rádio que apenas contém implementadas as camadas físicas;
- **Rádios transceptores + microcontrolador integrado** – Neste caso é fornecido o rádio em conjunto com um microcontrolador, do tipo *SoC*, onde são implementadas todas as camadas do *ZigBee*; dependendo do fabricante o desenvolvedor da aplicação pode adicionar a sua aplicação na memória do microcontrolador integrado, mas necessita de saber utilizar as primitivas *ZigBee*;
- **Módulos e *kit's*** – Módulos completamente funcional, disponibilizando no caso mais simples apenas uma ligação de comunicação (ex: porta UART) para a aplicação enviar os dados pela rede *ZigBee*, até um módulo que já inclui os sensores e um *firmware* completo, em que basta configurar os parâmetros para adaptar o módulo sensorial à aplicação.

Dado pretender-se a utilização de novos sensores nas redes *ZigBee*, escolheu-se usar módulos dentro da última categoria supracitada em que apenas oferecem um interface simples para a aplicação enviar os seus dados via rede *ZigBee*. Na tabela 2.1 apresenta-se a comparação das opções disponíveis no mercado que se adequam ao projecto (adaptado de [13] e corrigido por [14 a 23]). A maior parte destes módulos já são comercializados com um *firmware* que trabalha sobre o protocolo *ZigBee*, embora também se possa adquirir rádios livres que permitem o desenvolvimento de novos protocolos.

TABELA 2.1 - COMPARAÇÃO DE MÓDULOS 802.15.4 COMERCIALIZADOS.

Fabricante	Módulo	SoC/SiP	MCU	Antena Conector	Sleep	TX mA	RX mA	TX dBm	RX dBm	mm	Data	Interface	Firmware
Radiocrafts	RC220x	Chipcon CC2420	8bit ATmega128	Integrado MMCX	23µA, 1.3µA	23 27	26 30	0	-94	16.5 x 29.2 x 3.5	2008	UART, SPI, JTAG, ISP	Chipcon Z-stack ou *ZigBee* stack de terceiros
	RC2201 HP		8bit ATmega1281			140	28	17	-92	16.5 x 35.6 x 3.5	2008		
	RC230x	Chipcon CC2430 ou CC2431 (motor de localização)	8bit 8051			27	27	0		12.7 x 25.4 x 2.5	Jul 2008	UART, SPI	
Jennic	JN5139 M00 M01 M03	JN5139	16MHz 32bit RISC	Na placa SMA U.FL	2.6uA	37	37	2.3	-96	18x 30	Jun 2008	UART, SPI	*JenNet stack*, *ZigBee stack*
	JN5139 M02 M04			SMA U.FL		125	45	19	-100	18x 41			
Crossbow Technology	IRIS	ATmega128L AT86RF230	8bit ATmega1281	Interna	8uA	21 25	24	-3 a 3	-101	58x 32x 7	Abr 2007	UART, SPI I2C, Conector de expansão	*ZigBee stack*, Plataforma Moteworks
	MicaZ	Chipcon CC2420	8bit ATmega128L	Conector MMCX	15uA	22 25.4	27.7	-5 a 0	-94	58x 32x 7	Set 2005		
MeshNetics	ZDM-A1281-B0 (MNZB-24)	AT86RF230	8bit ATmega 1281v	Chip, Saída RF	6uA	18	19	-17 a 3	-101	18.8x 13.5x 2	2008	UART, SPI, I2C, JTAG	*BitCloud stack*, *SerialNet*, *OpenMAC*
	ZDM-A1281-PN (MNZB-A24)			Saída RF, U.FL		50	23	20	-104	38x 13.5x 2			
	ZigBit 900 (MNZB-900)	AT86RF212		Saída RF		26	11	11	-110	24x 13.5x 2.8			
Digi International Inc.	Series 1 XBEE	Freescale MC13213	8bit 689S08A HCS08	Mini monopolo, Chip, U.FL, RPSMA	10uA, 50uA	45	50	0	-92	24.3x 27.6	2006	UART	*ZigBee stack*, DigiMesh
	Series 1 Xbee PRO					250 180	55	18 10	-100	24.3x 33			
	Series 2 Xbee ZB	Ember EM250	16bit 12MHz RISC		1uA	40 35	40 38	3 1	-96 -95	24.3x 27.6	Abr 2008		Ember ZNet, DigiMesh
	Series 2 Xbee PRO ZB				10uA	295 170	45	17 10	-102	24.3x 33			

Da tabela infere-se que o módulo ZDM-A1281-B0 da MeshNetics destaca-se na maior parte das características. Principalmente, apresenta um menor consumo e maior sensibilidade rádio. A Universidade da Madeira já havia adquirido três opções no âmbito do estudo das redes de sensores sem fios. Por isso, limitou-se a nossa escolha a um dos três. Se num trabalho futuro for necessário um menor consumo energético ou maior sensibilidade de recepção, será suficiente substituir o rádio utilizado na aplicação. A comparação dos rádios, só fica completa analisando o *software* que os acompanha, desde o *firmware* à aplicação de visualização. Analisou-se este software para os rádios disponíveis nos laboratórios no capítulo três desta dissertação.

2.2 Sistemas de Monitorização

Nos dias de hoje, a monitorização em tempo real de atletas está chegando a um nível que leva aos treinadores, de desportos de elite, procurar cada vez mais a fórmula optimizada para o maior desempenho do atleta. Os parâmetros que interessa conhecer sobre um atleta são vários dependendo do tipo de exercício ou desporto. E são de diversos tipos, desde a mediação de resistência, passando pelo ângulo máximo de uma articulação até ao controlo mental sobre as capacidades motoras. Quanto maior for o número de parâmetros e quanto maior a precisão obtida, melhor será a fórmula de optimização de determinado atleta. Este tema é aprofundado pela Engenharia do Desempenho Humano [24]. Também são vários os parâmetros de interesse a conhecer do ambiente de treino. O ambiente presente na área de treino dependendo das características apresentadas pode influenciar de forma decisiva no rendimento do atleta [25] [26]. É corrente treinar atletas de alto nível, dependendo do desporto, em locais onde as características ambientais são de menor adequação do que na área de prova, em certos parâmetros como: temperatura, concentração de oxigénio, pressão atmosférica e outros, para que o atleta obtenha maior desempenho aquando na área de prova [27].

2.2.1 Sistemas Existentes

Realizou-se um estudo de mercado, sobre sistemas de monitorização de atletas que funcionam em tempo real e não interferem com a actividade realizada pelo atleta. Registou-se na tabela 2.2 o modo de funcionamento e os parâmetros monitorizados por estes sistemas.

TABELA 2.2 – EXEMPLOS DE SISTEMAS DE MONITORIZAÇÃO DE ATLETAS.

Tipo	Sistema	Funcionamento	Sensores	Parâmetros monitorizadas	Observações
Rede RF	GPSports	Módulo, fixado às costas do atleta via uma banda elástica. Envio da informação via RF para um computador em tempo real.	GPS, Sensor de batimento cardíaco, Acelerómetro.	Ritmo cardíaco, Velocidade, Posição, Acelerações, Distância.	Dados em tempo real. Só funciona no exterior com cobertura do sinal GPS. Máximo de 32 atletas. Elevado custo, 5000 Dólares / 5 atletas.
	Hosand	Módulo preso à camisola do atleta. E uso da banda torácica.	Sensor batimento cardíaco.	Ritmo cardíaco	Dados em tempo real. Limite a 1 só variável.
Individual	Suunto	Banda torácica + relógio com indicação em tempo real + pedómetro no pé do atleta.	Sensor de bat. cardíaco. Acelerómetro ou GPS.	Ritmo cardíaco, Velocidade, Distância.	Dados podem ser guardados para posterior análise. Individual.
	Garmin				
	Polar				
	Nike				
Software	PROZONE 3	Entre 8 a 12 câmaras de vídeo gravam todo o evento desportivo (Futebol)		Velocidade, Posição, Distância.	È feita uma análise por software do movimento do atleta. 5000 Euros por jogo.
	Athletemonitoring.com	Base de dados preparada para receber dados de cada atleta de equipas inteiras, e auxiliar o treinador.			É necessário registar os dados para então introduzi-los neste sistema.

Observou-se que os sistemas de monitorização existentes de atletas em tempo real actuais, que já utilizam redes sem fios, são adequados apenas a um número limitado de actividades desportivas. São geralmente dependentes do GPS, que se traduz na falta de cobertura no interior de edifícios. Os sistemas mais completos apresentam elevado custo. E mesmo estes, não oferecem as medições de parâmetros dos atletas e do ambiente de treino no mesmo sistema [28] [29]. No melhor dos casos de monitorização integrada, facultam a

monitorização da temperatura, e pressão do ar, mas num sistema que não funciona em tempo real [30] [31].

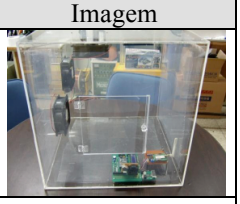
Pesquisou-se por sistemas de monitorização ambiental independentes, desenvolvidos sobre redes de sensores sem fios que se adeqúem à integração com o sistema de monitorização de atletas. Dos sistemas já comercializados, destacam-se na tabela 2.3 uma amostra dos produtos disponíveis no mercado, que mais se aproximam do sistema pretendido.

TABELA 2.3 – EXEMPLOS DE SISTEMAS DE MONITORIZAÇÃO AMBIENTAL COMERCIALIZADOS.

Sistema	Funcionamento	Sensores	Observações	
EKO Pró *Series Kit* Crossbow [32]	Série de nós preparados para a intempérie. A cada um destes nós pode ser ligados até 4 sensores. Utiliza uma rede *ZigBee* para encaminhar, em tempo real, a informação a um computador.	Temperatura Humidade Radiação Solar Humidade do solo Humidade nas folhas	Usado para a monitorização de solos para a ajuda no cultivo de alimentos. E para monitorizar alterações ambientais.	
Hobonode Wireless Sensors [33]	Série de nós preparados para a intempérie. Utiliza uma rede wireless para encaminhar os dados, em tempo real, para um computador.	Humidade do solo Temperatura	Usado para a monitorização de solos para a ajuda no cultivo de alimentos. E para monitorizar alterações ambientais. Extensão até 6km	

À parte dos sistemas comercializados, pesquisou-se por sistemas em fase de desenvolvimento. Na tabela 2.4 lista-se três exemplos de projectos publicados.

TABELA 2.4 – EXEMPLOS DE PROJECTOS DE SISTEMAS DE MONITORIZAÇÃO AMBIENTAL.

Projecto	Funcionamento	Sensores	Imagem
Design of Air Pollution Monitoring System using ZigBee Networks for Ubiquitous-City [34]	Sistema de monitorização ambiental para cidades com base na tecnologia de redes de sensores sem fios.	Poeiras CO_2 Humidade Temperatura	
SensorScope [35]	Desenvolvido para providenciar uma observação no espaço e no tempo de uma grande área. Alimentado por pequenos painéis solares.	Radiação Solar Precipitação Velocidade do vento E direcção Humidade do solo Temperatura Humidade	
CitySense [36]	Desenvolvimento e avaliação de um sistema sem fios que monitoriza uma cidade inteira via pontos Wi-Fi nos edifícios.	Velocidade do vento e direcção Precipitação Pressão barométrica Temperatura Humidade	

Com o resultado da pesquisa, verificou-se que os sistemas de monitorização ambiental embora meçam parte dos parâmetros pretendidos, têm sido desenvolvidos em sistemas dedicados e independentes. A integração destes módulos ao sistema de monitorização de

atletas requer adaptação. Esta adaptação apresenta várias dificuldades de integração em *hardware* e *software*. Inclusive o elevado custo monetário destes sistemas, fazem com que se pretenda um desenvolvimento integrado de raiz, com soluções de baixo custo em que partilham componentes comuns, como a rede de comunicações.

Estes sistemas são actualmente uma área de desenvolvimento activo pelas empresas de equipamentos para desporto, verificado pelo lançamento em curto espaço de tempo de novos dispositivos pelas marcas líderes [28] [30].

As limitações dos sistemas de monitorização de atletas no interior de edifícios, o número de componentes pelo corpo do atleta, o uso de sensores adequados à monitorização do ambiente de treino, a integração de ambos os sistemas na mesma rede de sensores sem fios e o custo final são objectivos do presente trabalho.

2.2.2 Parâmetros a Monitorizar

De entre os vários parâmetros importantes a conhecer, tanto de atletas como do ambiente de treino, definiu-se para estudo parâmetros importantes para a maioria das actividades e mais usados pelos sistemas de monitorização actuais. Estes são, no atleta: velocidade, distância, posição e ritmo cardíaco. No ambiente de treino são: luz, temperatura, humidade, monóxido de carbono, dióxido de carbono e oxigénio. De seguida descrevem-se os parâmetros a monitorizar e seus valores de referência, quando aplicável, para o melhor desempenho do atleta.

Velocidade

A velocidade de movimento pode ser medida como uma quantidade linear ou angular. Tipicamente quando todo o corpo está movendo linearmente no espaço, como a andar ou a correr, é medido o movimento de translação em relação ao espaço de prova, vindo expresso em metros por segundo. Se for a velocidade de rotação de um segmento do corpo, então mede-se uma quantidade angular, expresso em graus por segundo. Com estas medidas é possível comparar a velocidade em diferentes tarefas, ou entre atletas. Já no caso de ser um conjunto de movimentos complexos, torna-se difícil distinguir como medir a velocidade.

A velocidade pode ser expressa como máximo, média ou em valores instantâneos dependendo de qual medida é um indicador mais útil para o desempenho que está a ser medido. O simples registo de históricos de velocidade pode ser muito útil no estudo e desempenho de um atleta durante o 3535treino ou prova, bem como permite a comparação entre atletas. A velocidade máxima registada em atletas de elite é de 12,27 m/s pelo Usain Bolt [37].

Sensores de velocidade

Quando o tempo de movimento é superior a um segundo e a distância é conhecida, a velocidade pode ser medida com um cronómetro. Com movimentos angulares das articulações a velocidade pode ser medida por interruptores que actuam a cada movimento repetido da articulação, ou com sensores electromagnéticos. Neste caso é requerido electrónica com relógio. A velocidade pode ser processada através de distância ou ângulos e informação de tempo a partir de cinematografia. Por fim podem ser utilizados acelerómetros. Acelerómetros são sensores capazes de converter a aceleração da gravidade ou movimento em sinais eléctricos.

Com um único acelerómetro consegue-se medir movimento linear. Para medir rotações são necessários dois acelerómetros. Existe comercialmente disponíveis acelerómetros triaxiais. Com estes é possível calcular, com o processamento adequado, o vector de

velocidade [24]. Acelerómetros têm a vantagem de apresentarem uma medida directa e contínua. São utilizados apropriadamente quando montados sobre materiais rígidos. Quando presos ao tecido mole do atleta podem induzir a erro devido aos movimentos relativos do tecido.

Distância percorrida

A distância pode ser obtida através de optoelectrónica no campo de treino que regista a passagem do atleta em distâncias conhecidas, através de cinematografia ou indirectamente pela informação de velocidade [24].

Posição

A posição do atleta será sempre relativa a um ponto conhecido, pode ser definida por coordenadas geográficas onde são obtidas a latitude e a longitude da posição do atleta, ou por duas coordenadas ortogonais, expressas em metros, em relação a um ponto conhecido da área de treino.

Sensores de posição

Dos vários sistemas de posicionamento, destaca-se o sistema de posicionamento global, o GPS. Actualmente já apresenta precisões na ordem dos dois metros e tempo de resposta de 1/5 de segundo em equipamentos comerciais [38], sendo adequados para exercícios em áreas de longa extensão. Sua desvantagem é a impossibilidade de ser utilizada dentro de edifícios. Outra opção no exterior é a utilização de sistemas optoelectrónicos e de cinematografia em certos pontos de provas para detectar a passagem dos atletas por pontos pré-conhecidos. No caso de interior de edifícios podem, também, ser utilizados sistemas de cinematografia, em conjunto com processamento adequado, sistemas estes que apresentam elevada precisão [39] . Outros sistemas poderão ser utilizados, como ultra sons, embora ainda não tenha sido desenvolvidos sistemas simples [40] e adequados ao cenário de ter muitos atletas na mesma área de treino.

Na tabela 2.5, registou-se uma amostra dos vários tipos de equipamentos no mercado que realizam a monitorização da velocidade, distância percorrida e num caso a posição (via GPS).

TABELA 2.5 – EQUIPAMENTOS COMERCIALIZADOS DE MONITORIZAÇÃO DE VELOCIDADE, DISTÂNCIA PERCORRIDA E POSIÇÃO.

Sistema	Funcionamento	Preço	Imagem
Pedómetro no pé. Polar S3 Stride sensor [41]	Utiliza um acelerómetro para monitorização do movimento. Rádio próprio.	79 €	
Relógio. Forerunner 305 [42]	Utilização do GPS embebido no relógio.	120 €	
Pedómetro de bolso. Omron HJ-112 Digital Pocket Pedometer [43]	Utiliza um acelerómetro para medir os passos.	25 €	

Ritmo Cardíaco

Em geral o ritmo cardíaco pode variar de 50 batimentos por minuto até aos 190, dependendo da pessoa e da actividade realizada. A diferença entre o ritmo de repouso e o

ritmo máximo não é um bom indicador de capacidade física, mas reflecte o esforço desenvolvido durante o exercício pelo atleta.

Para evitar complicações com a saúde do atleta é importante conhecer o seu ritmo cardíaco máximo, para que a capacidade física do atleta não seja ultrapassada. À data actual existem fórmulas de cálculo deste valor máximo, que continuam a ser discutidas. Um debate mais aceso entre investigadores é o de inferir um diagnóstico cardíaco a partir da recuperação após uma actividade física. O consenso aponta que a taxa de recuperação no primeiro minuto após a actividade é um excelente guia para a capacidade física do atleta. Um estudo feito pela Cleveland Clinic [44], nos Estados Unidos, mostrou que, numa pessoa comum, os batimentos devem cair 20 pontos após o primeiro minuto de repouso. Nos atletas deve cair 50 pontos, e que no caso de cair 12 ou menos pontos, nesse primeiro minuto, indica uma probabilidade elevada de problemas cardíacos.

Existe uma dezena de maneiras de registar o ritmo cardíaco manualmente, mas durante a prática da actividade física estes métodos são impraticáveis. Um método utilizado e não evasivo é o registo dos sinais eléctricos gerados na contracção e relaxamento do coração, denominado de Electrocardiograma (ECG). Isto pode ser feito através de eléctrodos presos ao corpo, já que o corpo é um bom condutor eléctrico.

O coração pode ser pensado como um gerador de sinais eléctricos e que está envolvido num volume condutor, o corpo. O corpo e o coração são tridimensionais e logo, os sinais medidos através da pele vão variar dependendo da posição e inclusive da "qualidade" de contacto com a pele. Na figura 2.8 apresenta-se o sinal registado através de 12 eléctrodos colocados em diferentes partes do corpo, é possível ver os pontos I, II e III na figura 2.9.

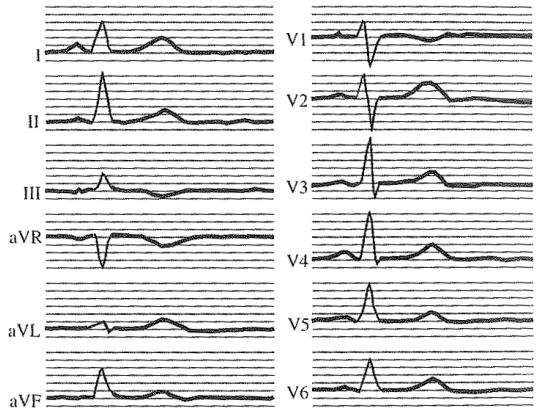


Figura 2.8 – Sinais em 12 pontos diferentes do corpo para uma mesma contracção do coração.

O diagnóstico usa a comparação do ECG entre diferentes indivíduos, sendo assim, necessárias normas padrão para a colocação dos eléctrodos. Para isso, é considerado como padrão o triângulo de Einthoven, apresentado na figura 2.9 [24].

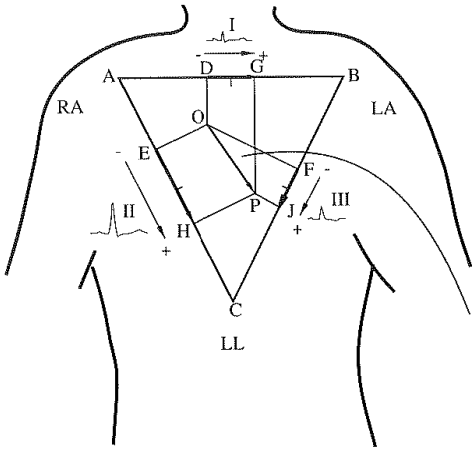


Figura 2.9 – Triângulo de Einthoven.

O sistema padrão para a análise completa do funcionamento do coração, usa 12 eléctrodos. Para a simples detecção de uma contracção cardíaca é possível usar apenas um subsistema com dois eléctrodos apenas [24] [45].

Outra técnica baseia-se em sistemas optoelectrónicos, que utilizam o princípio de funcionamento da fotopletismografia. Esta técnica usa a emissão de ondas luminosas sobre a pele dos pacientes, analisando o comprimento de onda da luz reflectida e suas variações nos diferentes tecidos do corpo, estimando assim o fluxo sanguíneo [46].

Na tabela 2.4 registou-se uma amostra dos vários tipos de equipamentos no mercado que realizam a monitorização do ritmo cardíaco.

TABELA 2.6 – SISTEMAS COMERCIALIZADOS DE LEITURA DO RITMO CARDÍACO.

Sistema	Funcionamento	Preço (aproximado)	Ilustração
Banda toráxica [32] [33].	Mede potencial eléctrico entre dois terminais encostados ao peito.	35 €	
Sensor óptico no braço [47].	Conjunto LED – sensor óptico, que verifica a proporcionalidade entre a passagem do sangue e a luz reflectida no braço.	110 €	
Sensor óptico no dedo [48].	Conjunto LED – sensor óptico, que verifica a proporcionalidade entre a passagem do sangue e a luz reflectida no dedo.	95 €	
Por "pedido" [49].	Dois contactos eléctricos, num pequeno relógio onde o utilizador encosta dois dedos de uma mão.	15 €	

Os vários sistemas acusam elevada precisão, comparável com os resultados de um electrocardiograma. Diferenciam-se no grau de conforto para o atleta.

Luz

A luz geralmente é referenciada como o espectro electromagnético visível da energia solar.

A luz é constituída por raios gama, raios X, raios ultravioleta, luz visível, infravermelhos, microondas e ondas rádio. Parte destas radiações conseguem atravessar a atmosfera e chegar à superfície terrestre. Chegam até nós: luz visível (cerca de 40%), raios ultravioletas, infravermelhos e ondas rádio. Como se pode verificar pela figura 2.10, o espectro visível para o olho humano vai desde os 400nm até os 700nm [50].

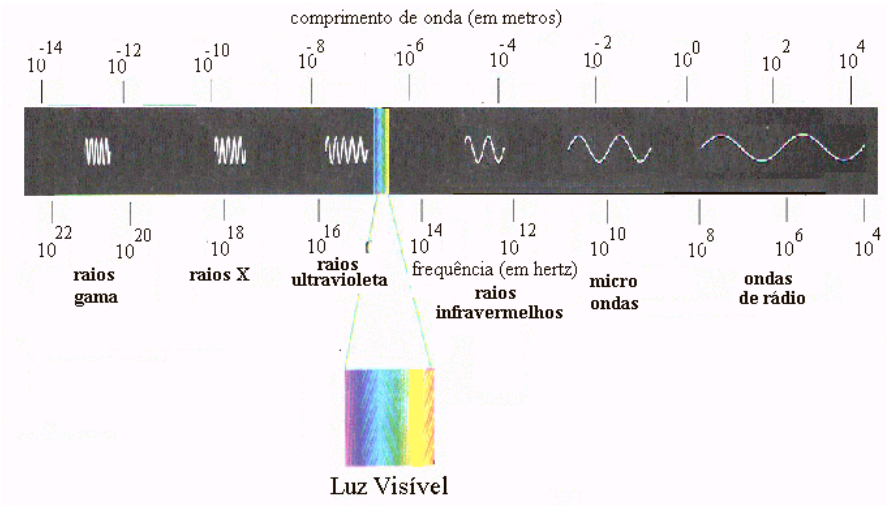


Figura 2.10 – Espectro electromagnético.

Lux é a unidade do Sistema Internacional (SI) para a iluminação. 1 lux é igual a 1 lumen por metro quadrado. Na tabela 2.7 apresenta-se valores típicos de luminosidade.

TABELA 2.7 – VALORES TÍPICOS DE LUMINOSIDADE.

Situação	Valores típicos
Luz do luar	1 lux
Um escritório claro	400 lux
Luz do sol num dia claro	32 000 a 100 000 lux

Um dos interesses de monitorizar a luz, é o controlo de exposição do corpo do atleta à luz solar durante o treino, pois esta faz parte do processo de produção de vitamina D, a qual é conhecida por aumentar o desempenho do atleta [51].

Temperatura

A temperatura e humidade determinam o quanto o ambiente é confortável para os humanos, animais e plantas.

Nos atletas cerca de 75% da energia consumida durante o exercício é gasta sob a forma de calor, causando inevitavelmente o aumento da temperatura corporal. Em ambientes frios, a maioria deste calor corporal é transferida para o ar, mas quando a temperatura ambiental excede a temperatura da pele, o calor é acumulado e pode levar a temperatura corporal subir a níveis perigosos [26].

Humidade

A atmosfera nunca está perfeitamente seca, existe sempre uma certa quantidade de vapor de água presente no ar. O ar consegue carregar água sobre a forma de vapor. Quanto mais quente mais vapor de água pode carregar. A 25ºC ao nível do mar 1kg de ar pode carregar cerca de 20g de água. A humidade absoluta neste caso é de 20g/kg, enquanto a relativa é de 100%. Nas mesmas condições se fossem apenas 10g/kg a humidade relativa seria de 50%. Se uma camada de ar apresentar uma humidade relativa de 70% a 30ºC e for arrefecida, chegará a um ponto a que a humidade relativa chegará os 100%, e dar-se-á condensação.

Nos atletas, grande parte do calor é dissipado pela evaporação do suor produzido. Se a humidade for excessiva esta transferência será reduzida, contribuindo no aumento de temperatura do atleta [26].

Monóxido de carbono

O monóxido de carbono (CO) é um gás, incolor e inodoro mas extremamente tóxico. Este gás sempre esteve presente como um constituinte menor da atmosfera, como produto de actividade vulcânica, fogos naturais ou iniciados pelo homem. Nas cidades o CO é um dos maiores poluentes atmosféricos. Os automóveis são os maiores responsáveis pela produção de CO, sendo as pessoas que vivem nos primeiros andares de edifícios nos centros urbanos as mais sujeitas a grandes concentrações de CO. Este também é produzido pela queima imprópria de outros combustíveis, como gás natural, madeira etc. Na tabela 2.8 apresenta-se valores de concentrações típicas [52][53][54].

TABELA 2.8 – VALORES DE CONCENTRAÇÃO TÍPICOS DE MONÓXIDO DE CARBONO.

Situação	Concentrações típicas
Nível natural presente na atmosfera	0,1 ppm
Média típica dentro de casas	0,5 a 5 ppm
Níveis junto de fogões ajustados apropriadamente	5 a 15 ppm
Fogões mal ajustados	30 ppm
Cidades com elevados níveis de tráfego	100 a 200 ppm
Chaminé de uma churrascaria caseira	5 000 ppm
Na saída do escape de um carro antes de se diluir no ar (sem conversor catalítico)	7 000 ppm

O nível máximo tolerado para a qualidade do ar nas cidades europeias é de 8,5ppm [55]. As consequências por inalação de CO, são dores de cabeça e náuseas para concentrações inferiores a 400ppm, e acima deste valor são potencialmente mortais. Através do programa europeu de luta anti-tabaco foram feitos estudos sobre o monóxido de carbono libertado pelo acto de fumar. Estes mostram que no meio dos fumadores a média é de 15ppm, valor muito superior aos 8,5ppm, e de 3,8ppm nos não fumadores [56]. O estudo [25] demonstra que o CO diminui o desempenho atlético, e recomenda que os atletas não devem exercitar em ambientes na presença de CO, em particular, nas bermas das estradas. Pelos dados aqui apresentados afere-se que os atletas não devem fumar.

Dióxido de Carbono

O dióxido de carbono (CO_2) é um gás à temperatura e pressão normal, que está presente na atmosfera. É muito conhecido como dos principais gases produzidos pela acção do homem que estão a alterar o clima terrestre. O CO_2 também é um produto resultante de processos naturais, como a respiração dos seres humanos, animais, plantas e muitos outros organismos.

O CO_2 é absorvido no processo da fotossíntese pelas plantas, mas também emitido durante a respiração das mesmas.

A concentração de dióxido de carbono (CO_2) na atmosfera começou a aumentar no final do século XVIII, quando ocorreu a revolução industrial, a qual demandou a utilização de grandes quantidades de carvão mineral e petróleo como fontes de energia. Desde então, a concentração de CO_2 passou de 280 ppm no ano de 1750, para os 368 ppm em 2000, representando um incremento de aproximadamente 30% [57].

Mais uma vez, nas cidades, os automóveis são grandes responsáveis pela produção de CO_2. O uso de combustíveis fósseis liberta gás carbónico e outros gases que ao se acumularem na atmosfera contribuem para o efeito estufa. Na tabela 2.9 apresenta-se exemplos de valores típicos de concentração de dióxido de carbono [58].

TABELA 2.9 – VALORES DE CONCENTRAÇÃO TÍPICOS DE DIÓXIDO DE CARBONO.

Situação	Valores típicos
Nível presente na atmosfera	360 a 390 ppm
Dentro de um escritório ou carro com janelas fechadas	500 a 2500 ppm
Auditório lotado com pouca ventilação	10 000 ppm

A inalação em excesso causa asfixia, levando à hiperventilação. O CO2 passa a ser tóxico para o ser humano para concentrações iguais ou superiores a 1% (10.000 ppm).

Vários estudos sobre os efeitos do aumento da concentração de CO2 na atmosfera indicam que este aumento reduz o desempenho dos atletas, inclusive apontado como factor decisivo para a não obtenção de novos recordes mundiais, nos últimos anos [59].

Oxigénio

É um dos elementos mais importantes da química orgânica, participando de maneira relevante no ciclo energético dos seres vivos, sendo essencial na respiração celular dos organismos aeróbicos. É um gás à temperatura ambiente, incolor, insípido, e inodoro. Representa aproximadamente 20% da composição da atmosfera terrestre.

O oxigénio é formado pela fotossíntese das plantas e posteriormente utilizado pelos seres vivos na respiração. O valor médio actual é de 20,9%. Valores inferiores a 19.5% são considerados de insuficientes para o ser humano [60]. Embora a capacidade de absorver oxigénio por um ser vivo dependa de inúmeros factores, como número de glóbulos vermelhos e outros, vários estudos indicam que a presença de maior concentração de oxigénio facilita a recuperação do atleta da actividade física realizada [61].

2.2.3 Sensores

De seguida faz-se uma abordagem às diferentes tecnologias sensoriais.

Ópticos – Quantificam a concentração de substâncias medindo o nível de radiação electromagnética que é absorvida, emitida ou dispersada (Ex: compostos orgânicos voláteis). Na figura 2.11, adaptada de [62], apresenta-se a absorção de radiação electromagnética em distintos comprimentos de onda para diferentes gases, servindo como meio de medida da sua concentração no ar.

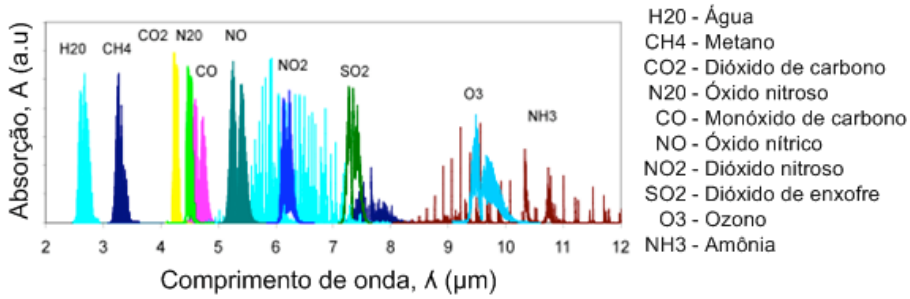


Figura 2.11 – Absorção de radiação electromagnética de vários gases.

Fibra-óptica – Os sensores em fibra óptica podem ser classificados em dois grandes grupos: internos e externos. Nos internos um feixe óptico dentro da fibra é afectado pela exposição da mesma à amostra. Nos externos o feixe óptico atravessa a amostra no ar. Ambas as medidas são feitas pelas alterações na luz emitida ou reflectida. Ex: compostos químicos, temperatura, força, pressão [63].

Electroquímicos – Informação em tempo real sobre a composição química de um determinado meio. Quando o gás a medir entra em contacto com o electrólito, é produzido uma diferença de potencial entre os eléctrodos. Ex: oxigénio, CO, Cloro PH [64].

Biológicos – Incluem um elemento de reconhecimento biológico (enzima) associado a um transformador. Ex: pesticidas. [65].

Colorimétricos – Um material reactivo produz cor, cuja intensidade é proporcional à quantidade da substância a medir. Pode ser medido visualmente ou através de espectrofotometria. Ex: hidrocarbonetos, pH [66].

Piezoeléctricos – Sensores que absorvem o elemento químico numa superfície provocando uma alteração de massa que pode ser medida. Ex: força, aceleração temperatura [67].

A escolha de um sensor pode seguir diversos critérios, de seguida destacam-se os principais:

- Resolução espacial (geo-referenciação, alcance)
- Resolução temporal (duração, periodicidade, tempo de resposta)
- Características das medições (precisão, gama de concentrações, destrutivo, manual/automático, natureza da amostra, limite de detecção)
- Facilidade de utilização (complexidade da utilização, informação, tamanho, portabilidade)
- Capacidade de se ligarem em rede (processamento local, eficiência energética)
- Acessibilidade (disponibilidade, custo).

3 ESTUDO DE SISTEMAS *ZIGBEE*

Um dos objectivos deste trabalho era o estudo do *hardware* de suporte ás comunicações sem fios baseadas no protocolo *ZigBee*. Neste capítulo, apresentam-se as características e os testes realizados com base em três kits *ZigBee* disponíveis no laboratório de telecomunicações e redes: Tmote Sky, MicaZ disponibilizados pelo projecto Foresmac [68] e XBEE que foram adquiridos no início do projecto como terceira opção. Implementou-se em cada caso uma RSSF, na qual desenvolveu-se funcionalidades que nos permitiram inteirarmo-nos do funcionamento destas redes e identificar os seus pontos fracos e fortes, permitindo a comparação entre eles, nomeadamente quanto à facilidade de desenvolvimento, custo, consumo, funcionalidades e limitações.

Ainda neste capítulo, apresenta-se um estudo de propagação do sinal RF em que realizou-se várias medidas para verificar o desempenho dos nós sensores em diferentes ambientes. Verificou-se que a existência de obstáculos, reflexões de interiores e proximidade ao solo afectam drasticamente a qualidade do sinal. Isto pode influenciar na escolha de topologia para a RSSF e na densidade de nós pela área de monitorização dos atletas.

3.1 Tmote Sky

Implementou-se uma rede de seis nós sensores nos laboratórios da Universidade com nós Tmote Sky da Moteiv [69], com topologia em malha. Tmote Sky é um módulo de baixa potência sem fios para utilização em redes de sensores, em aplicações de monitorização, e de prototipagem rápida. Cada nó está equipado com sensores, como humidade relativa do ar, temperatura e luminosidade, e os dados registados pela rede incluem também a tensão interna do módulo e a qualidade do sinal recebido (LQI). O protocolo de comunicação usado é o IEEE 802.15.4 *ZigBee*. Os dados são reunidos num computador e exibidos numa interface apropriada.

A programação dos nós sensores é suportada pelo TinyOS [70], que é um micro sistema operativo de fonte aberta, desenhado para redes de sensores sem fios. Este micro sistema operativo é baseado no modelo *event-driver programming* em vez do *multithreading*. O sistema operativo é escrito na linguagem nesC, que é uma extensão do código C preparada para a programação de sistemas embebidos. Um dos nós sensores age como *gateway* e está ligado ao servidor para a recepção e registo de dados. O protocolo de encaminhamento é do tipo *multi-hop*, permitindo que os dados sejam transferidos pela rede até à *gateway*.

3.1.1 Características

As características principais do Tmote Sky são:

- 250 kbps, rádio 2.4 GHz IEEE 802.15.4 Chipcon;
- 8 MHz TI MSP430 microcontrolador com 10 Kb RAM, consumo de 0,5 mA;
- Antena integrada no circuito com 125 m de alcance no exterior e 50 m interior;
- Sensor integrado de humidade e temperatura;
- Baixo consumo;
- Início rápido do estado de adormecido (< 6 us);
- Encriptação e autenticação por *hardware*;
- Programação e recepção de dados por USB;
- 16 pinos de expansão e conector de antena SMA opcional;
- Suporte do TinyOS;

A figura 3.1 apresenta o módulo Tmote Sky onde pode-se observar os componentes que o compõem [69]. O módulo é alimentado por duas baterias AA. No caso de estar ligado ao computador por USB, não há necessidade do conjunto de baterias. A antena integrada do Tmote Sky é uma antena impressa do tipo F-invertida (semelhante a um monopolo onde a secção superior é dobrada para manter-se paralela ao plano de massa) colocada no fim da placa afastada das baterias. Embora não apresente um diagrama de radiação omnidireccional, a antena permite o sinal alcançar os 50 metros no interior de edifícios e até os 125 metros no exterior.

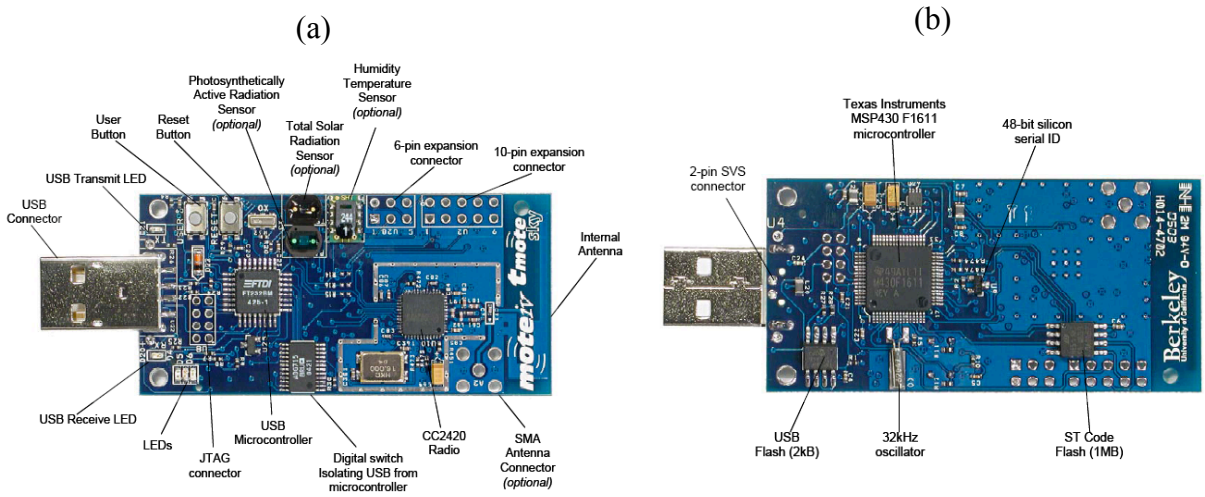


Figura 3.1 – Módulo Tmote Sky: a) vista de topo; b) vista inferior.

3.1.2 Arquitectura

A figura 3.2 apresenta a arquitectura do sistema. A rede de sensores recolhe dados do ambiente e envia-os para a *gateway*. Este nó, por sua vez, está ligado a um computador que apresenta o estado da rede e os dados recolhidos. Os dados podem ser registados num ficheiro.

Para garantir a recepção dos dados pela *gateway* a partir de qualquer nó da rede, realizou-se um estudo de cobertura, através da análise da propagação do sinal em ambientes no interior e exterior de edifícios. O estudo é apresentado em detalhe numa secção à frente.

O interface gráfico de utilizador é fornecido pela ferramenta denominada Monitor ForesMac. Basicamente, é uma ferramenta Java modificada a partir do *Trawler*, uma ferramenta fornecida por defeito pela Moteiv. O Monitor ForesMac inclui um conjunto de funcionalidades desenvolvidas pela empresa Edosoft [71], como o reconhecimento dos vários canais dos diferentes sensores, apresentação em tempo real dos dados recolhidos e legenda detalhada na janela de leitura dos dados dos sensores. Após a aplicação iniciar, uma janela com a topologia de rede aparece, como a apresentada na figura 3.3 adaptada de [71]. A janela apresenta as conexões entre os nós e a qualidade do sinal actual. Os nós são amovíveis, de forma a serem posicionados de uma maneira adequada para visualização.

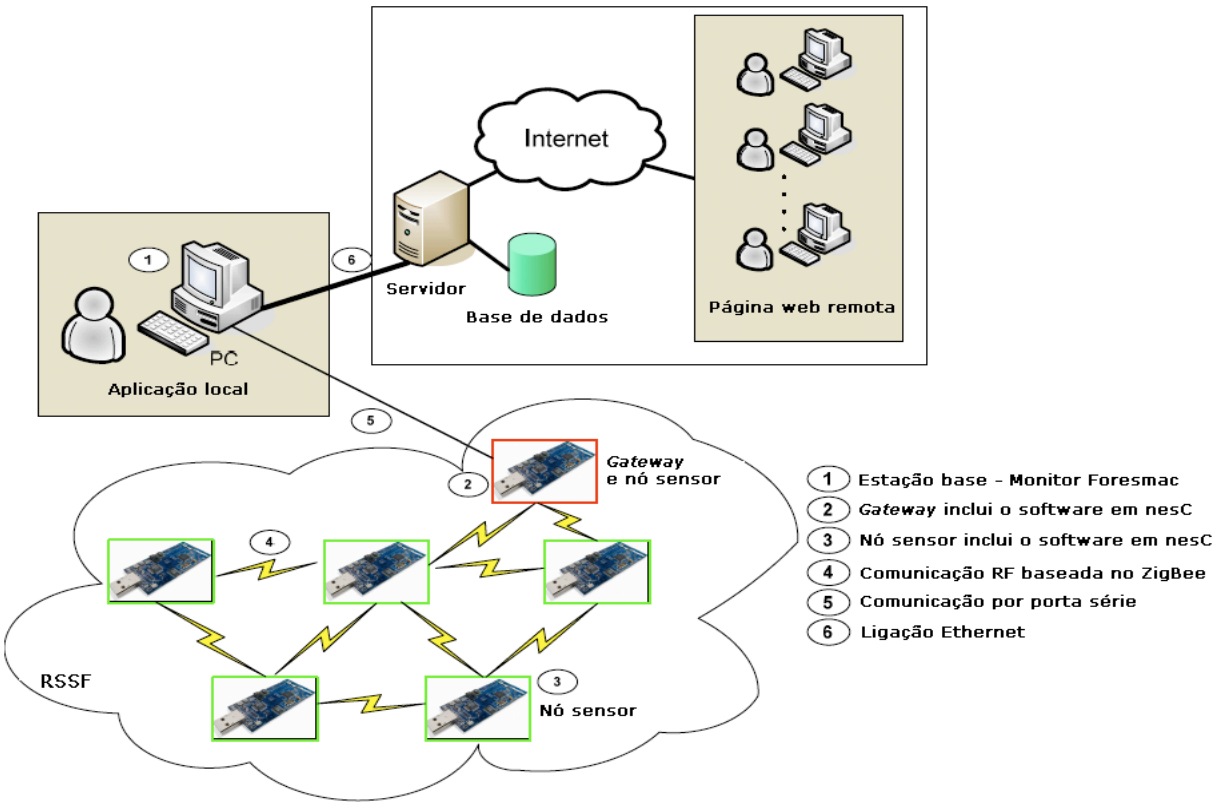


Figura 3.2 – Arquitectura da rede Tmote Sky.

A informação apresentada inclui:

- Número de identificação do nó
- Número de mensagem recebida
- Número de pacotes perdidos
- Percentagem de humidade
- Temperatura do ar
- Sensor de luz da gama fotossintética – espectro visível (320 – 730 nm)
- Sensor de luz da gama solar total – espectro visível mais infravermelho (320 – 1100 nm)
- Temperatura interna do microcontrolador
- Tensão interna do microcontrolador

Outra janela de visualização é apresentada na figura 3.4, que é um interface para monitorizar as várias leituras dos sensores. Os valores recolhidos dos nós sensores são apresentados em diferentes cores ao longo do tempo. Estão disponíveis botões para facilitar a análise dos resultados. É possível navegar através do gráfico, ampliar e voltar a inciar o registo a partir dos 0 segundos. O eixo vertical corresponde aos valores dos ADCs, sem calibração. Para obter os valores correctos é necessário aplicar a curva de calibração às leituras.

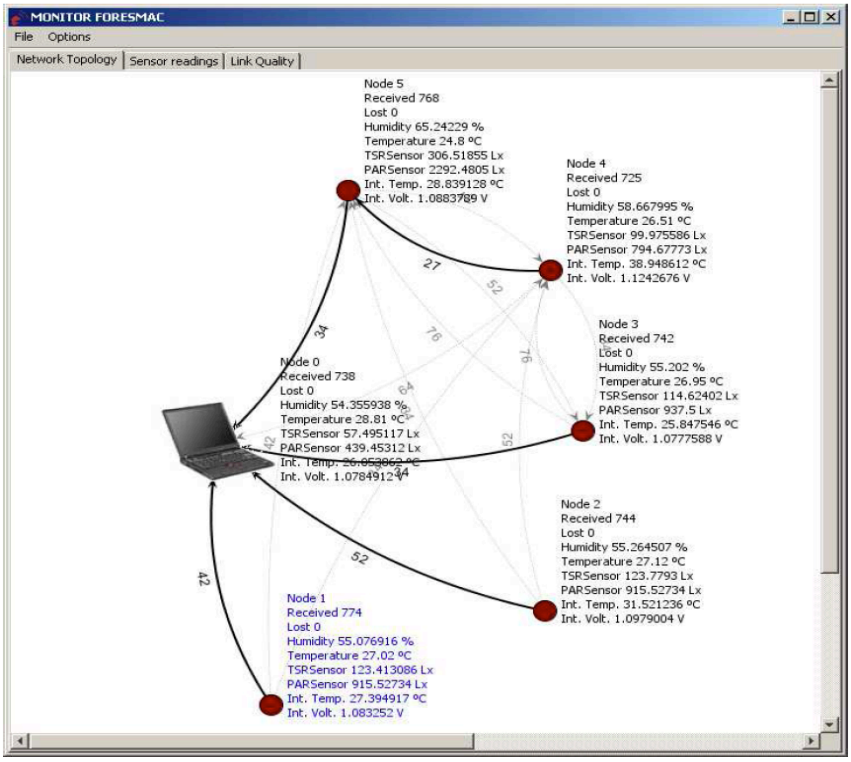


Figura 3.3 – Monitor ForesMac, topologia de rede.

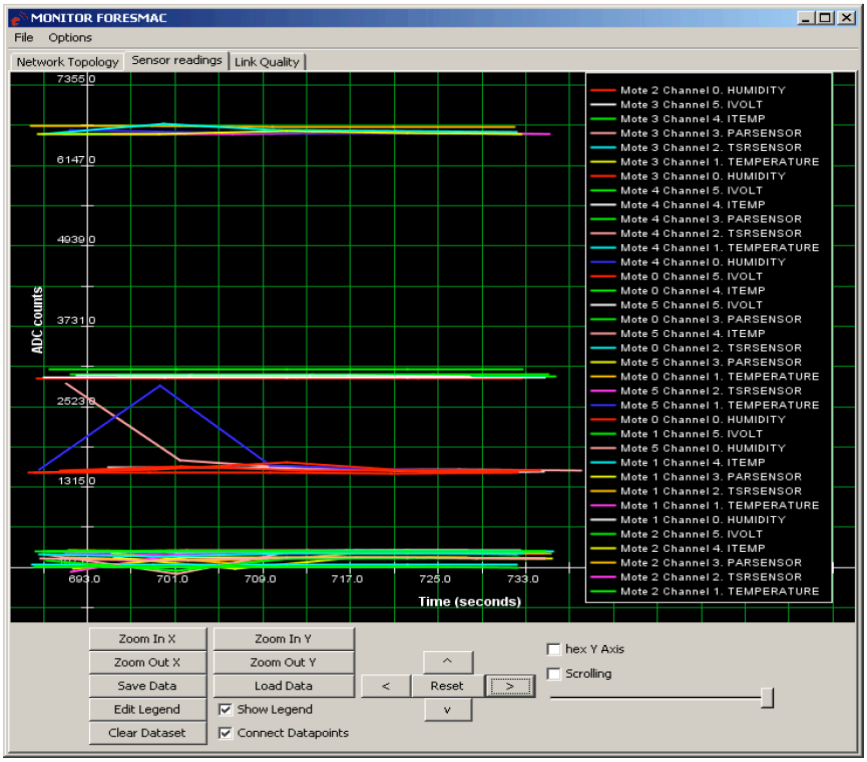


Figura 3.4 – Monitor ForesMac, leituras dos sensores.

A última janela apresenta a qualidade do sinal das várias ligações entre os diversos nós ao longo do tempo, como apresentado na figura 3.5.

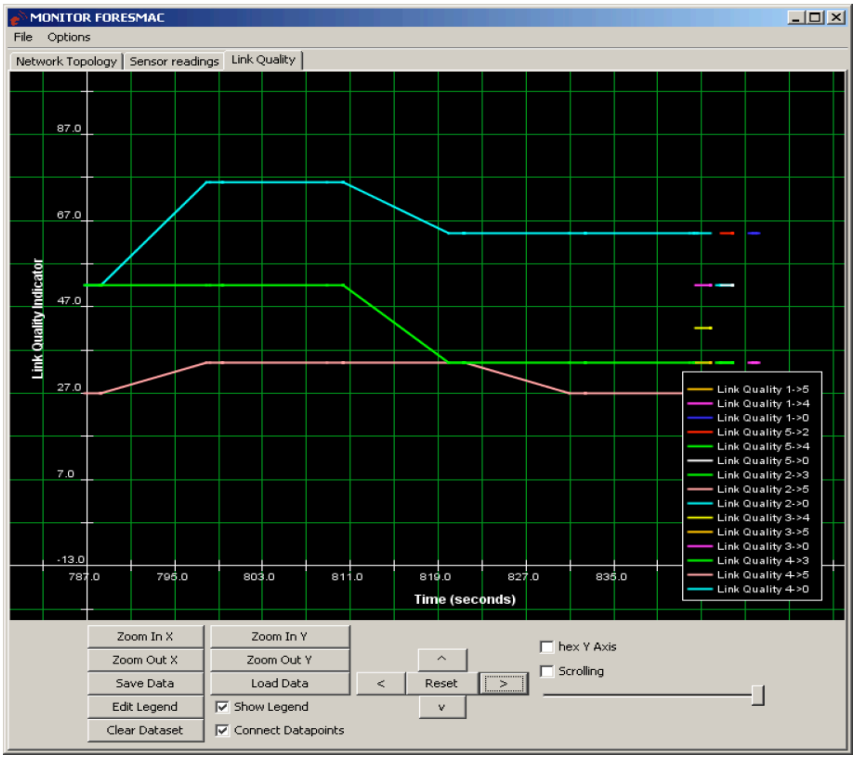


Figura 3.5 – Monitor ForesMac, qualidade do link.

A qualidade do *link* permite observar o efeito de perturbações pelo caminho de comunicação entre nós.

3.1.3 Configurações

A rede de sensores foi testada para avaliação do seu desempenho. No primeiro exemplo foi considerado cinco nós sensores em torno da *gateway*, criando-se assim uma topologia em estrela, como é apresentado pela figura 3.6. Neste teste ainda avaliou-se a operação dos vários sensores físicos. Como todos os nós estavam no mesmo laboratório, a temperatura é quase a mesma. O sensor dois estava num lugar escuro, como se pode verificar pelos baixos valores de luminosidade. O nó três estava numa região com sombra. O número quatro estava debaixo de uma luz fluorescente, indicando um valor superior. Finalmente o sensor cinco junto da janela, embora sem luz directa do sol, apresentou o valor mais elevado de luminosidade.

Para uma disposição diferente da rede de sensores, o encaminhamento da informação é apresentado na figura 3.7. Neste caso, a informação do sensor número cinco chega à *gateway* por meio dos nós quatro e três. O mesmo acontece para o sensor dois que usa o nó três para enviar os seus dados. Num outro teste os nós foram colocados em linha, e na figura 3.8 apresenta-se o encaminhamento da informação estabelecido.

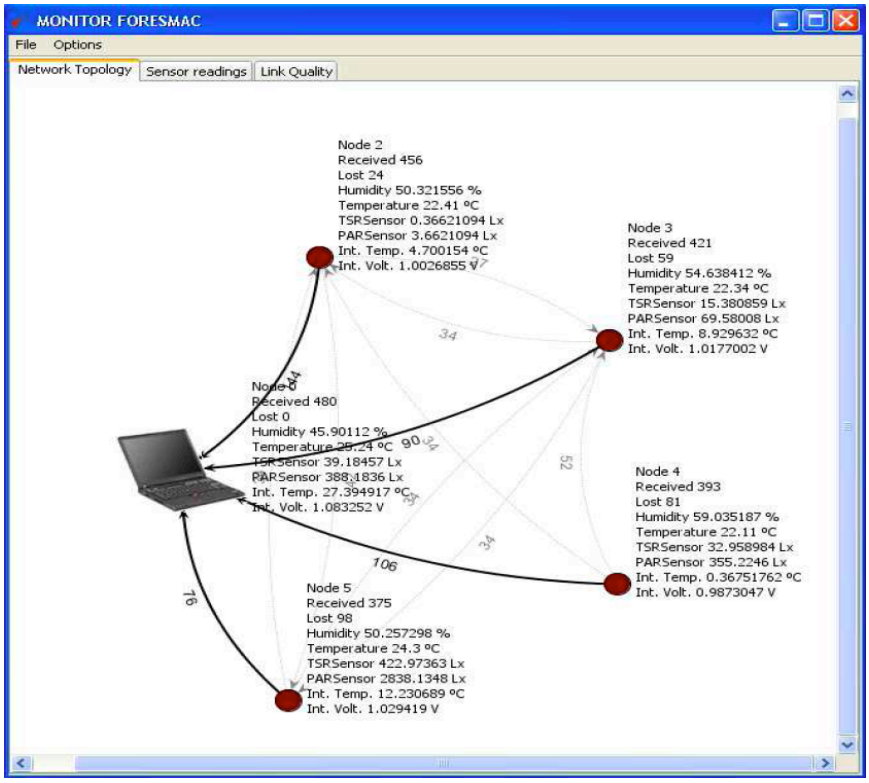


Figura 3.6 – Resultados para uma topologia em estrela.

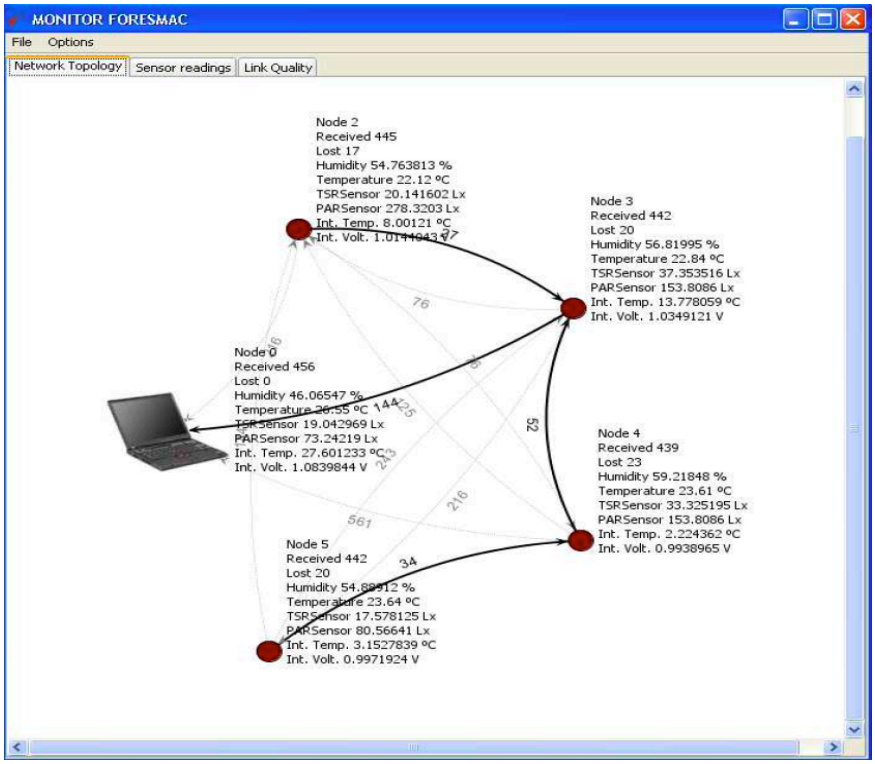


Figura 3.7 – Topologia *multi-hop*.

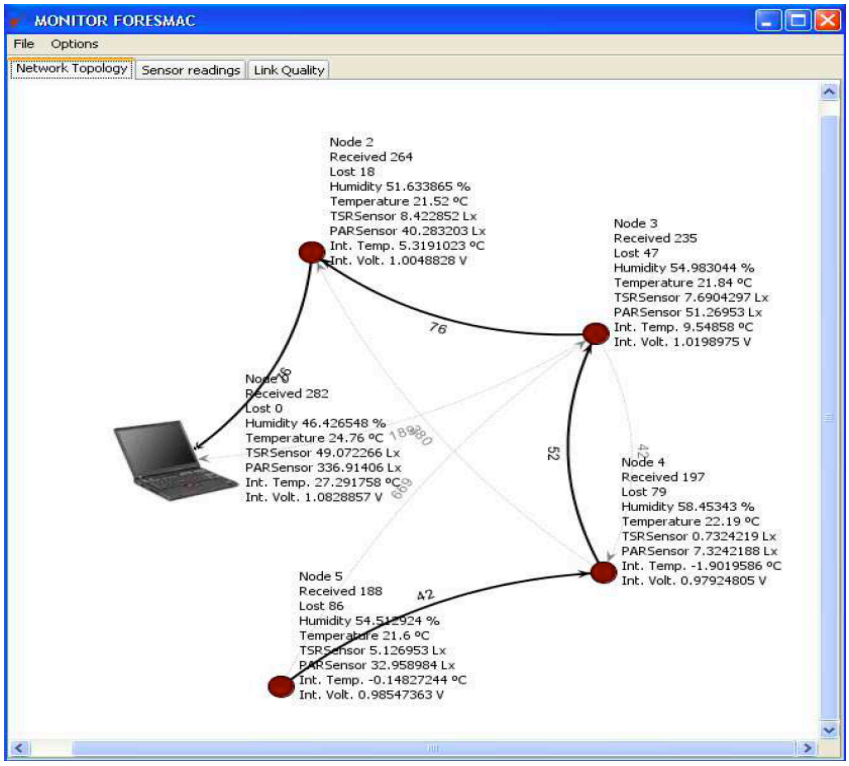


Figura 3.8 – Teste de rede para sensores em linha.

O software oferece uma percepção das ligações entre nós da rede de uma forma gráfica adequada. Mas a disposição apresentada não representa a posição real dos nós sensores.

3.1.4 Resultados

A ferramenta permite registar os dados recolhidos, para permitir a análise posterior, num ficheiro. No caso de uma amostragem de período curto o ficheiro torna-se muito longo, devido à quantidade de dados, dificultando a leitura dos dados. Assim, optou-se por registá-los numa Base de Dados. Para esse fim, alterou-se a aplicação de visualização para além de apresentar os dados, registá-los ao mesmo tempo numa base de dados em MySQL para posterior análise. Utilizou-se o Matlab [72] para representar os dados armazenados. Para calibração das leituras, foram usadas as seguinte expressões:

$$\begin{aligned}
&\text{Humidity} = -4 + 0.0405 * Hum_reading - 2.8\times10^{-6} Hum_reading^2 \\
&\text{Temperature} = -39.6 + 0.01 * Temp_reading \\
&\text{Luminosity} = \frac{15000 * PSR_reading}{4096} \\
&\text{Total Luminosity} = \frac{15000 * TAR_reading}{4096} \\
&\text{Ivolt} = \frac{1.5 * ivolt_reading}{4096}
\end{aligned} \tag{3.1}$$

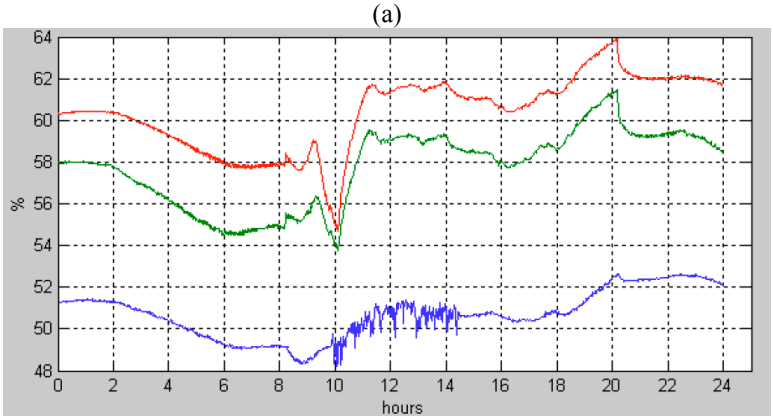
A figura 3.9 apresenta as instalações de teste da Universidade da Madeira onde a rede de sensores foi instalada, consistindo num laboratório com diversos equipamentos e um longo corredor.

(a) (b)

Figura 3.9 – Instalações de teste UMa: a) laboratório de telecomunicações; b) corredor.

Nas figuras 3.10 e 3.11 apresentam os gráficos com as leituras dos sensores de três nós. Os gráficos correspondem a um determinado dia, sendo a meia-noite o início dos dados. Figura 3.10a) apresenta a humidade relativa dos três nós sensores e na figura 3.10b) a temperatura correspondente. É interessante reparar, que como é esperado, o aumento da temperatura corresponde à descida da humidade. A luminosidade pode ser verificada na figura 3.11.

(a)



(b)

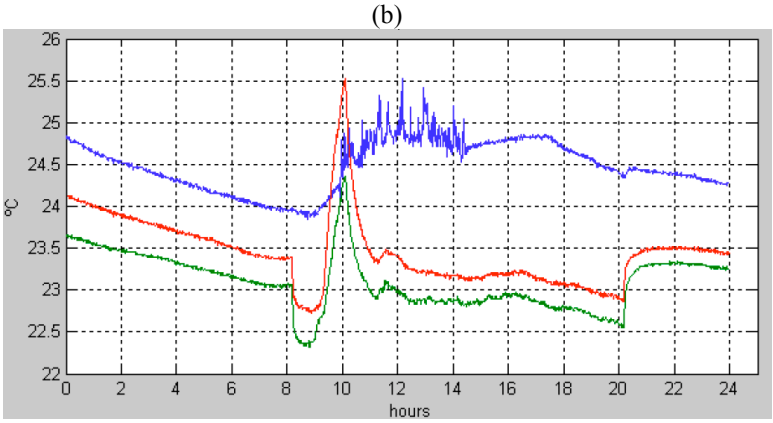


Figura 3.10 – Leitura de três nós sensores: a) humidade; b) temperatura.

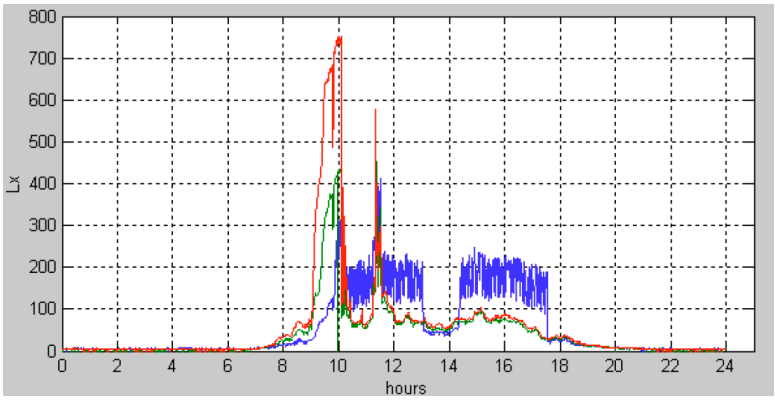


Figura 3.11 – Leitura da luminosidade de três nós sensores.

Para avaliar a duração da bateria avaliou-se a descarga destas, pela tensão interna do módulo, ao longo de vários dias. Este indicador corresponde à energia consumida das baterias, para um período de amostragem de 10 segundos. Para este período notou-se que as baterias apresentam uma duração inferior aos três dias, como se pode verificar pela figura 3.12. A figura apresenta a mudança de bateria no terceiro dia.

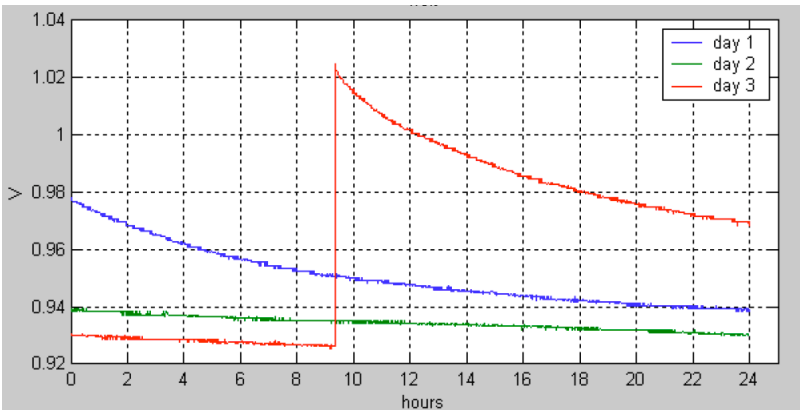


Figura 3.12 – Tensão interna de um nó sensor para três dias consecutivos.

3.2 MicaZ

A segunda rede de sensores sem fios foi baseada no nó sensor MicaZ. Os nós sensores MicaZ são distribuídos comercialmente pela Crossbow technology Inc. A figura 3.13 apresenta o módulo rádio MicaZ e o diagrama de blocos respectivo [18].

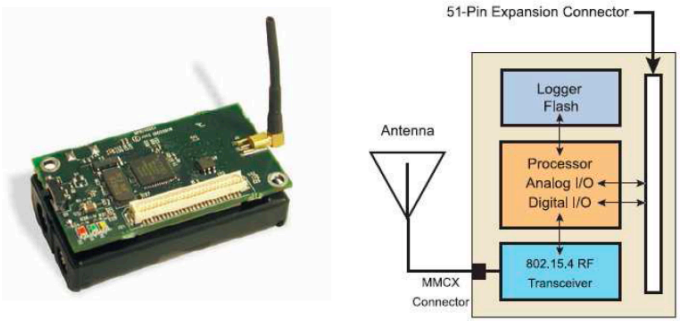


Figura 3.13 – Diagrama de blocos do MicaZ.

Este nó oferece as seguintes capacidades:

- Rádio RF conforme o IEEE 802.15.4/*ZigBee*;
- 2.4 a 2.4835 GHz;
- *Direct Sequence Spread Spectrum* que é resistente à interferência RF e providencia segurança de dados;
- Taxa de transferência de dados de 250 kbps;
- Corre TinyOS 1.1.7 ou superior;
- *Plug and play* com toda a gama de placas de sensores da Crossbow.

O MicaZ é suportado pelo MoteWorks, uma plataforma completa para a criação de redes de sensores sem fios. A plataforma de software fornecida com o MoteWorks é optimizada para redes de baixa potência operadas a baterias, sendo uma solução que se pode usar nas diversas áreas de aplicação das RSSF. De entre várias ferramentas do MoteWorks, XSniffer permite que os utilizadores monitorizem a comunicação *multi-hop* da rede. Este programa corre num PC e usa um MicaZ com um *firmware* apropriado para monitorizar o tráfico de pacotes RF.

Implementou-se uma RSSF formada por cinco nós sensores MicaZ. Após programar todos os nós com o *firmware* da aplicação, o desempenho da rede foi testado usando o XSniffer. A figura 3.14 apresenta o aspecto do XSniffer, com um nó programado como estação base. O ecrã de registo apresenta todo o tráfico permitido pelo filtro definido em *Options*. O endereço dos nós remetentes é apresentado na coluna *Orgn*, a coluna *Addr* mostra o endereço do destinatário da menssagem. O nível de sinal recebido medido pelo nó do XSniffer é apresentado em *RF*. *Len* é o comprimento da mensagem, *Src* é o endereço do nó que reencaminhou a mensagem, sendo igual ao *Orgn* no caso de não ter chegado directamente à *gateway*. As colunas numeradas referem aos dados contidos nas mensagens.

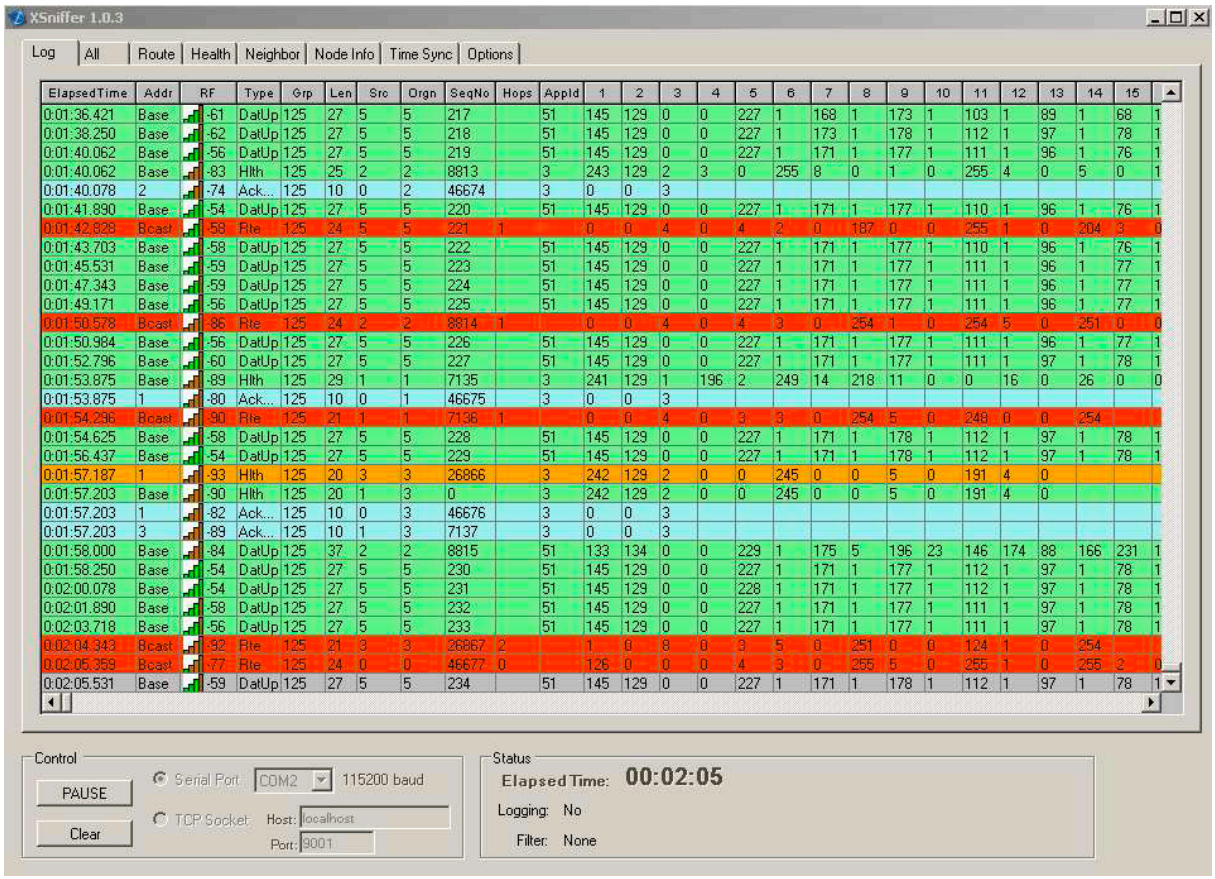


Figura 3.14 – Janela do XSniffer.

Da coluna *RF* é possível observar o nível de sinal que chega à *gateway*. Os nós sensores 1 e 3 estão noutro laboratório, significando que existe uma parede pelo caminho de propagação do sinal. O nó 2 está no mesmo laboratório da *gateway* mas longe da sua posição. O nó 5 está perto da *gateway*.

Após vários testes, criou-se uma pequena rede de monitorização ambiental nos laboratórios de telecomunicações para avaliar o seu desempenho. Usou-se três nós sensores, com placas de sensores MTS400 (que inclui sensor de luz, temperatura, humidade, pressão barométrica e acelerómetro). Um quarto nó programado como *gateway* foi ligado a um computador com acesso à Internet. Para isto desenvolveu-se uma página *Web* para apresentar em tempo real os resultados das medidas. Avaliou-se as condições ambientais, duração das baterias e nível de sinal recebido.

3.2.1 Página *Web*

Em vez de utilizar-se o MoteView ou XSniffer para visualizar a informação, utilizou-se a ferramenta XServe, que recebe os dados da porta série e armazena-os numa base de dados. A página *Web* desenvolvida, usando a linguagem PHP, acede directamente à base de dados e representa os valores registados no formato apresentado na figura 3.15.

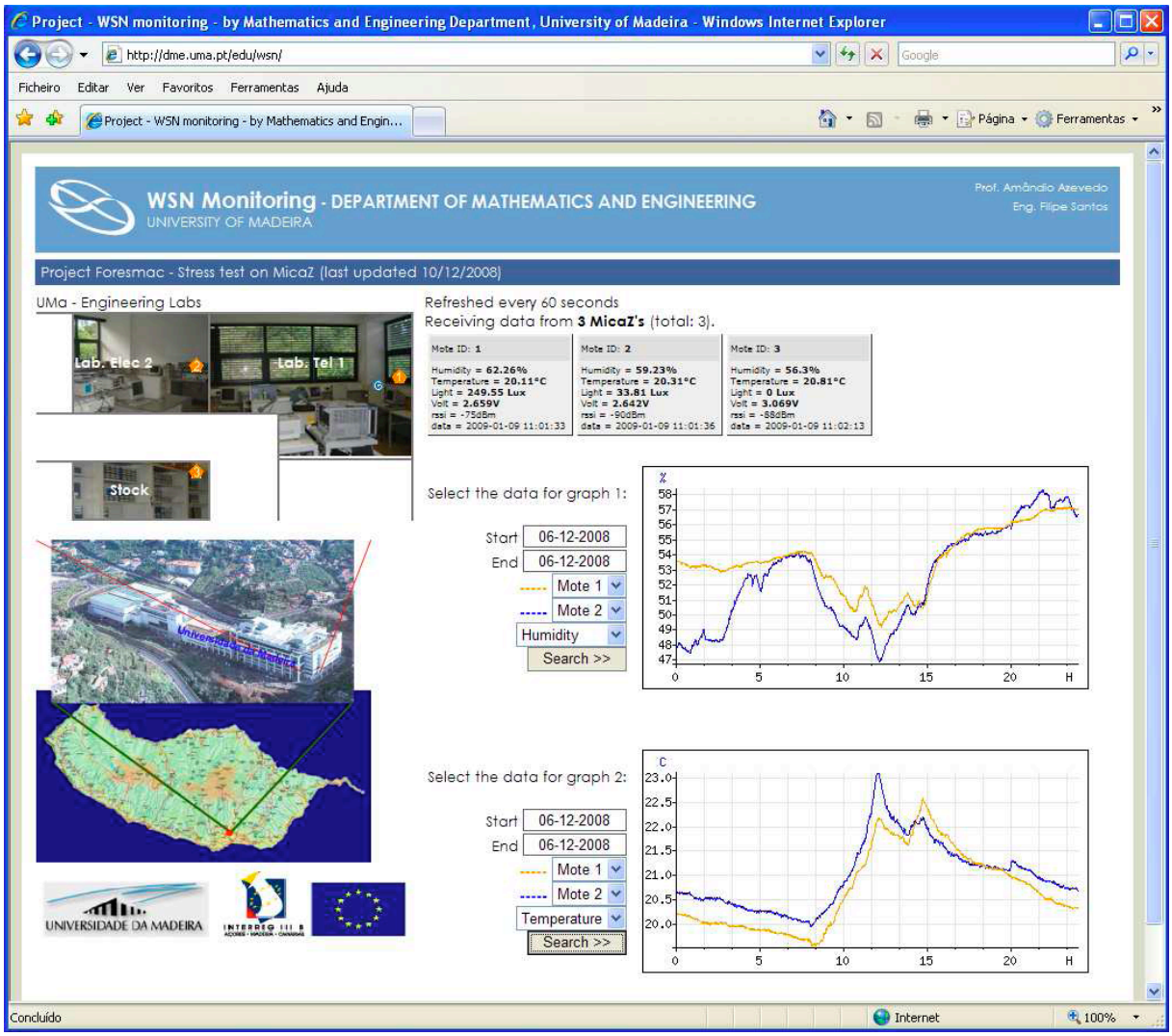


Figura 3.15 – Página *Web* para MicaZ, visualização dos dados.

A base de dados contém as leituras dos sensores. Na tabela 3.1 apresenta-se uma amostra. A primeira coluna apresenta a data de recepção das leituras. A segunda coluna indica o nó sensor a que corresponde as leituras. A terceira coluna é indicação do nó que foi usado para transferir os dados do sensor de origem. As outras colunas são a tensão, humidade, temperatura, pressão, luminosidade, aceleração em x e y, respectivamente.

TABELA 3.1 – AMOSTRA DE LEITURAS DO MICAZ REGISTADAS NA BASE DE DADOS.

Data	Id_no	Id_parente	Tensão	Humidade	Temperatura	Pressão	Luminosidade	Acel. x	Acel. x
2008-11-22-09:46:03	1	0	496	1365	6082	19083	65465	447	448
2008-11-22-09:46:24	2	0	489	1322	6116	18758	65469	389	420
2008-11-22-09:46:42	3	2	408	1360	6175	18680	65408	450	451
2008-11-22-09:47:02	1	0	496	1365	6084	19083	65467	446	448
2008-11-22-09:47:23	2	0	489	1322	6116	18757	65466	389	420
2008-11-22-09:47:40	3	2	408	1359	6176	18681	65408	450	451
2008-11-22-09:48:00	1	0	496	1365	6086	19084	65463	447	448
2008-11-22-09:48:21	2	0	489	1322	6116	18756	65466	389	419
2008-11-22-09:48:39	3	2	408	1362	6175	18680	65408	450	451
2008-11-22-09:48:59	1	0	496	1365	6086	19083	65464	447	448
2008-11-22-09:49:20	2	0	489	1322	6117	18757	65467	389	420

Para avaliar o nível de sinal recebido, foi também guardado na base de dados o parâmetro *Received Signal Strenght Indicator* (RSSI), que representa o valor do nível de sinal recebido. Pois este valor não é entregue na mensagem de dados. A medida do nível de sinal não é feita no mesmo período que a amostra de sensores apresentada na tabela 6.1. Contudo permite visualizar a evolução ao longo do tempo. A tabela 3.2, apresenta uma amostra dos resultados.

TABELA 3.2 – AMOSTRA DE LEITURAS DE RSSI.

Data	Id_node	RSSI
2008-11-22-09:44:40	3	221
2008-11-22-09:45:37	2	211
2008-11-22-09:46:23	1	246
2008-11-22-09:46:40	3	220
2008-11-22-09:47:37	2	211
2008-11-22-09:48:23	1	248
2008-11-22-09:48:40	3	221
2008-11-22-09:49:37	2	211
2008-11-22-09:50:23	1	248
2008-11-22-09:50:40	3	220

Das leituras são criados gráficos para a visualização adequada. A luminosidade é obtida através do seguinte algoritmo:

```
for n=1:length(luminosity_reading))
        if(luminosity_reading>0)
                x=luminosity_reading; y=0;
                Luminosity(n)=0;
                aux1=bitshift(1,(bitshift(bitand(x,112),-4)))-1;
                aux2=bitshift(1,(bitshift(bitand(y,112),-4)))-1;
                if((16.5*aux1+bitand(x,15)*aux1)~=0)
                        Luminosity(n)=(16.5*aux1+bitand(x,15)*aux1)*0.46/exp(3.13*((16.5*aux2
                        +bitand(y,15)*aux2)/(16.5*aux1+bitand(x,15)*aux1)));
                end
        else
                Luminosity(n)=0;
        end
end
```

Para os outros parâmetros, foram utilizadas as seguintes fórmulas para calcular os valores correctos das leituras dos sensores:

$$\begin{aligned}
&\text{Humidity} = (-39.6 + 0.01 * Temp_reading - 25) * (0.01 + +0.00008 * Hum_reading) \\
&\qquad\qquad + 0.0405 * Hum_reading - 2.8\times10^{-6} Hum_reading^2 \\
&\text{Temperature} = -39.6 + 0.01 * Temp_reading \\
&\text{Voltage} = \frac{125.352}{Volt_reading} \\
&\text{RSSI} = Rssi_reading - 300;
\end{aligned} \tag{3.2}$$

A página de Internet indica a posição no mapa do edifício, onde as medidas são recolhidas. O primeiro laboratório, representado na figura 3.16a), contém a *gateway* e o nó sensor 1. O nó sensor está perto da janela, que contém persianas que causam alguma obstrução à incidência de raios solares directamente sobre o sensor. Como pode ser observado, o laboratório está equipado com diverso equipamento que influenciam a propagação do sinal. A figura 3.16b) apresenta o laboratório onde o segundo nó sensor foi colocado. Este laboratório tem uma parede que partilha com o primeiro. O terceiro compartimento, é um pequeno armazém de material electrónico que não possui janelas. Assim, a única luminosidade é da iluminação fluorescente, que só é ligada esporadicamente.

(a) (b) (c)

Figura 3.16 – Laboratórios onde as medidas foram obtidas.

Todos os nós usam antenas monopolo. Após os primeiros resultados foi observado que o nó sensor no armazém necessitava de uma antena de maior ganho. Assim, foi considerada uma antena *biquad* em vez do monopolo, dado que o ganho é da ordem dos 8 dBi. Este nó está ligado a uma fonte de alimentação ligada à rede eléctrica, que fornece uma tensão constante. Os outros sensores são alimentados com baterias.

Na parte superior da página *Web* aparece três tabelas com os últimos resultados recebidos, uma por cada nó sensor, como apresenta a figura 3.17. Nesta página, duas janelas de gráficos fornecem a visualização dos dados por dia ou por vários dias. Como pode ser observado na figura 3.18, o gráfico apresenta os dados para o período indicado nos campos *Start* e *End* para os dois nós seleccionados. O parâmetro ambiental a visualizar é seleccionado no campo de lista apropriado.

Mote ID: **1**	Mote ID: **2**	Mote ID: **3**
Humidity = **49.23%**	Humidity = **51.38%**	Humidity = **48.07%**
Temperature = **22.03°C**	Temperature = **21.45°C**	Temperature = **21.42°C**
Light = **53.13 Lux**	Light = **536.13 Lux**	Light = **0 Lux**
Volt = **2.62V**	Volt = **2.615V**	Volt = **3.069V**
rssi = -55dBm	rssi = -85dBm	rssi = -84dBm
data = 2008-12-11 16:23:07	data = 2008-12-11 16:23:53	data = 2008-12-11 16:23:19

Figura 3.17 – Página *Web* para MicaZ, medidas dos sensores.

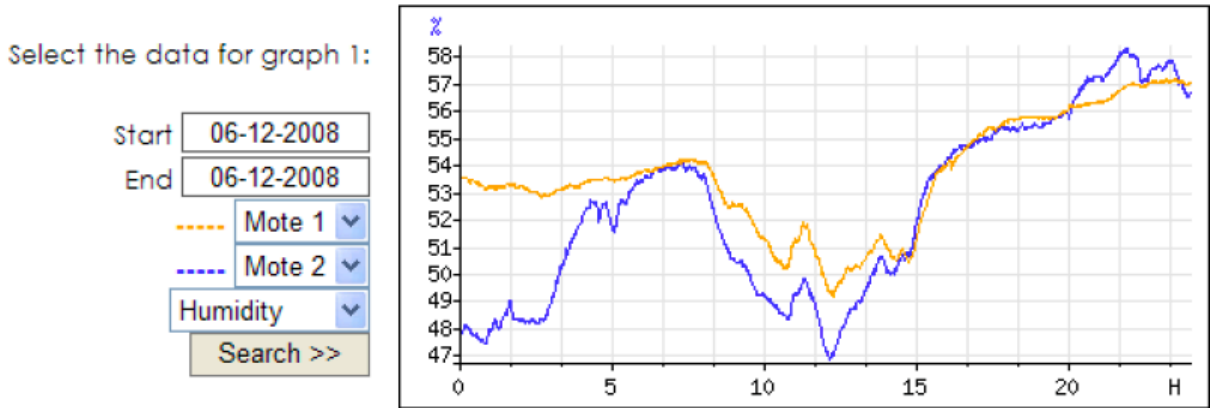


Figura 3.18 – Página *Web* para MicaZ, janela de visualização de gráficos.

3.2.2 Resultados

A rede de sensores sem fios é utilizada para registar os parâmetros ambientais desde Novembro de 2008. Com estes dados avaliou-se o desempenho da rede. O período de amostra para as medidas é de um minuto, que impõe a duração das baterias. Como o sensor 1 está perto da *gateway*, a sua potência de saída é -10 dBm. Os outros dois sensores usam uma potência de saída de 0 dBm. Para o sistema de alimentação, a figura 3.19 mostra a duração das baterias para o sensor 1 e 2. Observou-se que o nó 1, que opera a -10 dBm, tem uma duração até as 100 horas, ou seja, pouco mais que quatro dias. As baterias do nó 2, que opera a 0 dBm, duram apenas 87 horas, ou seja, quase três dias e meio. As baterias são recarregáveis de 2500 mAh de carga. O consumo médio é da ordem dos 60 mW para a periodicidade de medidas e envio de um minuto.

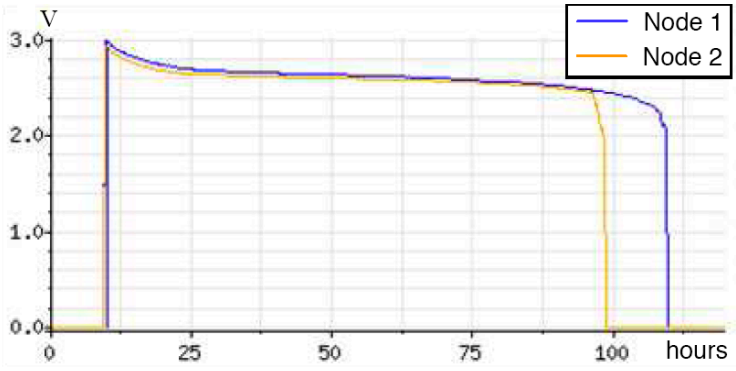


Figura 3.19 – Duração das baterias nos módulos MicaZ.

Para os parâmetros ambientais, a figura 3.20 apresenta a humidade registada durante seis dias de Novembro. A hora da origem é meia-noite. Nos mesmos dias, a figura 3.21 apresenta a temperatura. Das figuras podem-se observar que os valores do nó 3 têm menor variação, visto que está num compartimento interior do edifício. Finalmente, a luminosidade é apresentada na figura 3.22 para os mesmos dias. A luminosidade do nó 2 tem a maior variação, dado que as luzes internas estão usualmente desligadas, e assim, apenas é influenciado pela luminosidade solar. Nos valores do nó 1, pode-se observar que entre as 18H00 e 19H00 as luzes estão ligadas formando o degrau, representado no gráfico, ao fim do dia. É interessante notar que a luz no nó sensor 3 apenas é registada por pequenos períodos, quando a luz do armazém é ligada.

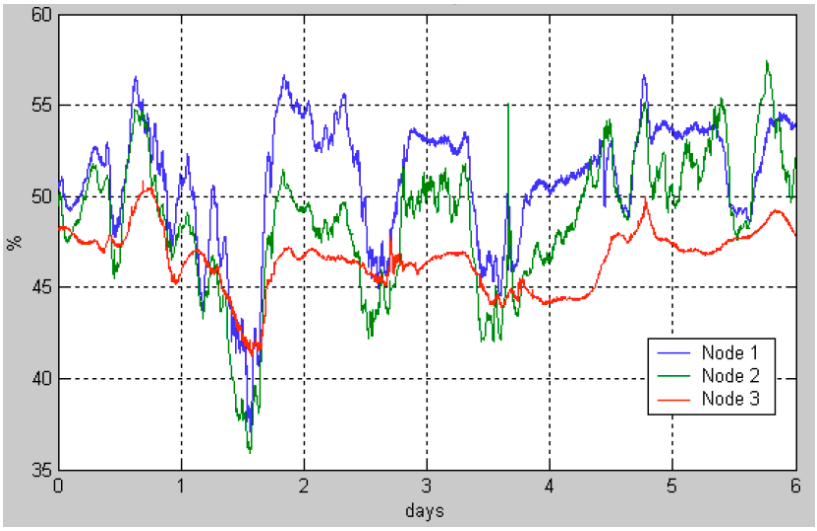


Figura 3.20 – Humidade, durante seis dias.

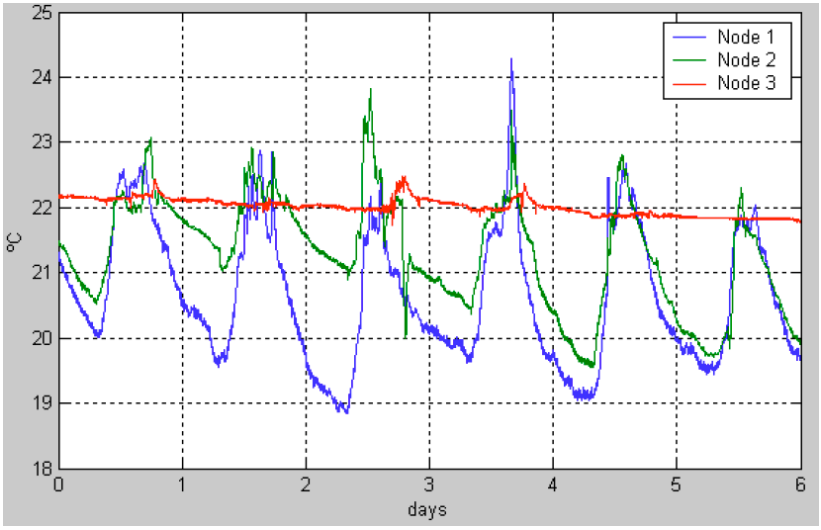


Figura 3.21 – Temperatura, durante seis dias.

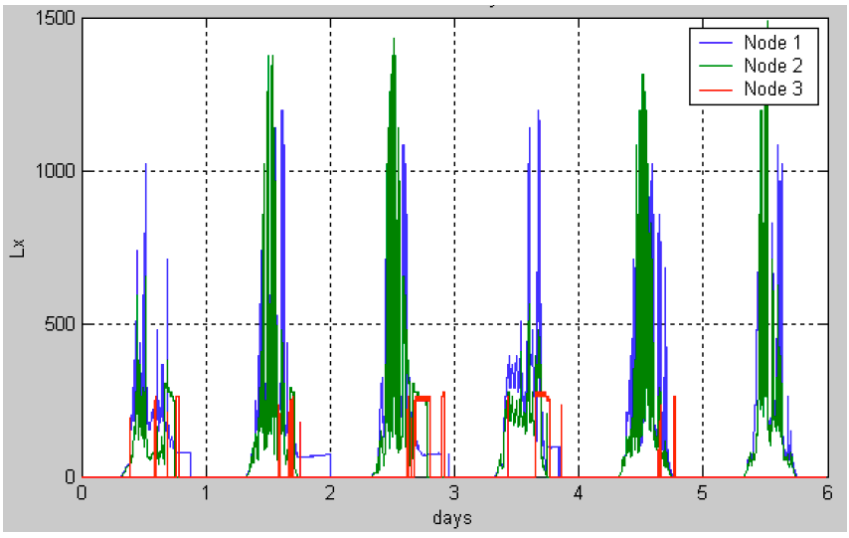


Figura 3.22 – Luminosidade, durante seis dias.

Sobre o nível de sinal recebido em cada ligação entre os nós, na figura 3.23 é apresentado a variação de RSSI para os mesmos seis dias. Embora na mesma posição, o nó sensor sofre a interferência da utilização dos laboratórios, principalmente o nó 1, visto o laboratório ter vários alunos a trabalhar. O nó 1 e 2 ligam-se directamente à base, mas o nó 3 às vezes usa o nó 2 para enviar os seus dados.

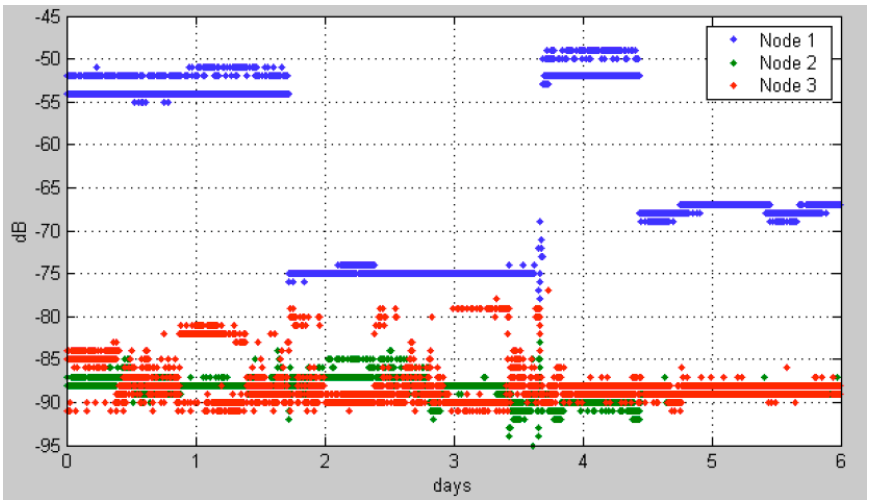


Figura 3.23 – Nível de sinal recebido (RSSI).

A figura 3.24, apresenta os momentos em que o nó 2 foi usado para retransmitir as mensagens do nó 3 (a posição no 0 corresponde a estar directamente ligado à base).

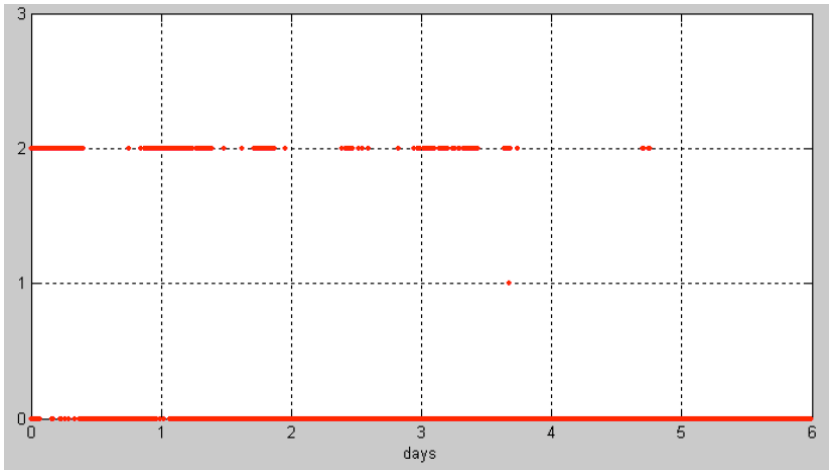


Figura 3.24 – Comunicação do nó 3 com a base, directamente e através do nó 2.

Como já referido anteriormente, as baterias duram à volta de quatro dias. A figura 3.25 ilustra o que acontece quando a tensão está abaixo de certo valor. Esta situação é preciso ter em conta, visto que os erros induzidos nas medidas são elevados.

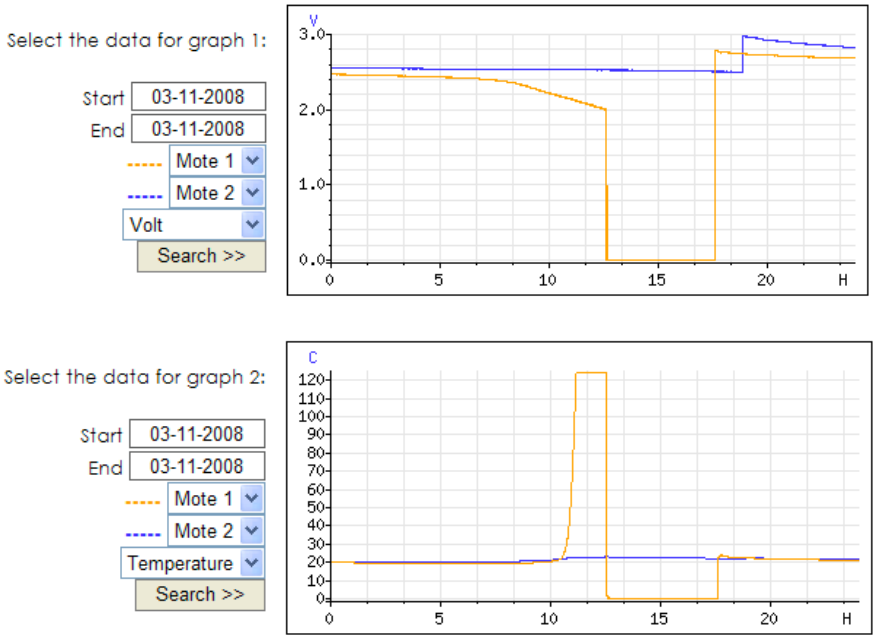


Figura 3.25 – Erro que ocorre no fim da descarga das baterias.

Esta situação, resulta do nó continuar a enviar mensagens de dados quando está a trabalhar a um nível de tensão de bateria inferior ao mínimo de funcionamento dos sensores do módulo. Ponto importante a ter em conta no desenvolvimento dos nós de monitorização que irão trabalhar com energia de baterias.

3.3 XBEE

Módulo RF produzido pela DIGI [22], o XBee, é uma solução em que não se precisa de ser especialista na tecnologia de transmissão de dados, libertando o desenvolvedor para trabalhar apenas na aplicação. O rádio XBEE da Digi destaca-se por: implementar a rede de comunicações *ZigBee*; oferecer uma interface de hardware e software simples; e ter custo aceitável.

3.3.1 Características

Características gerais da plataforma:

- Baixo custo (cerca de 27 Euros por unidade)
- Baixa potência
- Rede de ponto para multiponto / ponto a ponto
- Rede malha avançada
- Até 250Kbps de taxa de transferência de dados
- Comunicação RF logo ao tirar da caixa sem ser preciso configurar
- Encriptação AES de 128-bit
- Gama variada de antenas e conectores
- Aprovado para utilização no espaço Europeu.

Na figura 3.26 apresentam-se os módulos XBEE e as possibilidades de antenas.

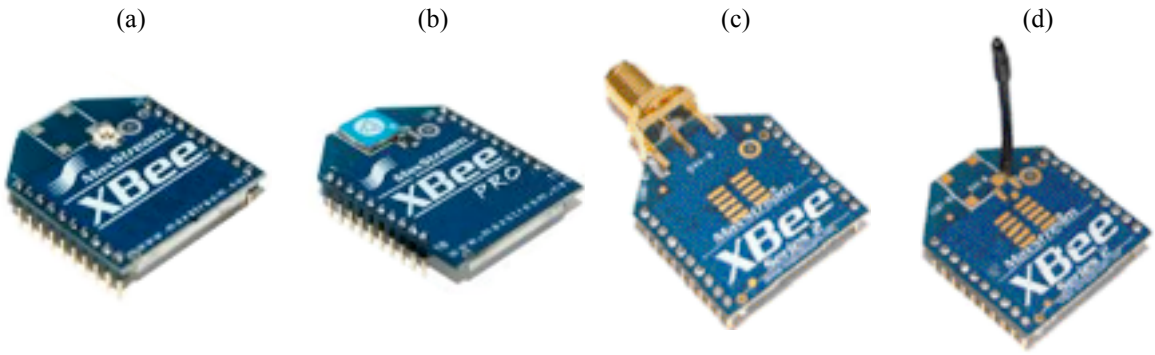


Figura 3.26 – Módulos XBEE e antenas disponíveis: a) conector U.FL.; b) antena chip; c) conector RPSMA; d) antena whip.

O modulo XBee engloba a implementação das camadas do padrão 802.15.4, do *ZigBee* e ainda uma camada API/AT que realiza a interface entre a aplicação e o *ZigBee*. Na figura 3.27 apresenta-se a arquitectura do módulo XBee [22].

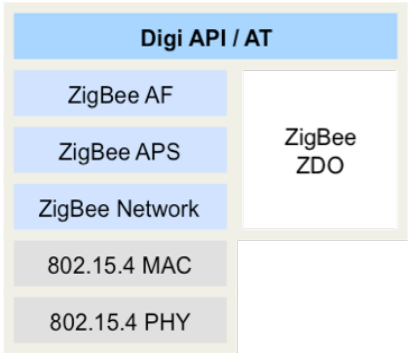


Figura 3.27 – Arquitectura do módulo XBee.

O módulo XBee apresenta várias entradas e saídas digitais e analógicas. No modo mais geral interliga-se aos dispositivos de uma aplicação, através de uma porta série. Na figura 3.28 apresenta-se a configuração genérica de aplicação do módulo XBee entre dois módulos de uma aplicação [22].

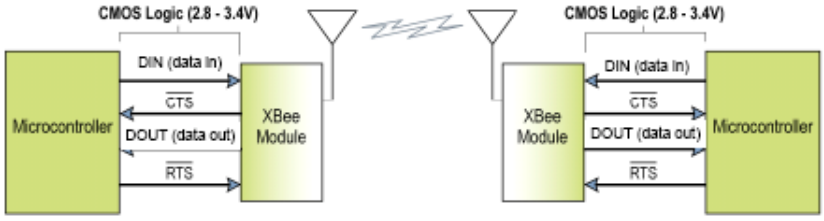


Figura 3.28 – Diagrama de fluxo de dados numa interface UART com o XBee.

As entradas e saídas incluem, além da UART, quatro ADC's, oito pinos digitais, e uma saída em PWM. Estas E/S podem ser configuradas directamente no módulo ou remotamente pela própria ligação rádio. É possível configurar, período de amostragem, estados por defeito, detecção de alteração de estado, alteração de estados e leitura de valores da conversão A/D. Podendo-se, só com o módulo XBee desenvolver certas aplicações, bastando para isso, adicionar os sensores / actuadores pretendidos a estas portas de E/S.

Os módulos Xbee funcionam segundo um *firmware* instalado e pré-configurado. São distribuídos quatro *firmwares*:

XBee 802.15.4 – Primeiro *firmware* disponibilizado com o módulo XBee série 1. Este *firmware* passa mensagens de um só nó para outro nó, com um pacote de resposta entre eles. Não existe saltos ou encaminhamento das mensagens neste protocolo. A rede baseada neste *firmware* pode apenas comunicar até ao alcance máximo de um nó individual. Na figura 3.29 apresenta-se as topologias que podem ser implementadas com este *firmwares*.

Figura 3.29 – Topologias de rede implementadas pelo firmware XBee 802.15.4: a) ponto-a-ponto; b) ponto-a-multi-ponto.

XBee ZNet 2.5 e XBee ZB – O XBee ZNet 2.5 usa uma adaptação do protocolo *ZigBee* pela DIGI, e o XBee ZB é a implementação do próprio protocolo sendo compatível com outros rádios de outros fabricantes que corram o protocolo *ZigBee*. Ambos suportam o encaminhamento em malha (*mesh*), permitindo os pacotes atravessarem múltiplos nós (múltiplo *hops*) para chegarem ao seu destino. Isto permite que os nós sejam espalhados por uma região alargada e mantenham as comunicações entre todos os dispositivos de uma rede. Através destes *firmwares* é possível implementar uma rede que suporta falhas dos nós, numa topologia em malha a rede auto-configura-se utilizando os nós vizinhos, permitindo que as mensagens cheguem ao seu destino, mas obriga que todos os nós da rede estejam sempre activos e tenham funções de *router*, não sendo adequado para situações de baixos recursos energéticos. A segunda topologia, estrela, é formada em torno de nós centrais, *routers*, que depois se interligam entre si. O conhecimento dos *routers* simplifica a gestão do protocolo, permitindo a criação de redes robustas que podem expandir por longas distâncias e densidades. Esta topologia obriga apenas que os nós routers estejam sempre activos, permitindo que os nós sensores utilizem técnicas de poupança energética e logo sejam alimentados por pequenas baterias. Na figura 3.30 apresentam-se as topologias implementadas por estes *firmwares*.

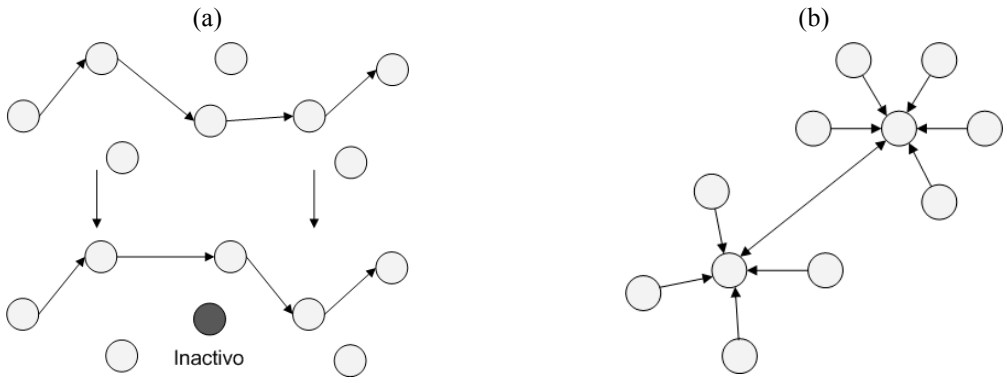


Figura 3.30 – Topologias de rede implementadas pelo firmware XBee Znet 2.5 e ZB: a) malha; b) estrela.

DigiMesh 2.4 – *Firmware* proprietário da DIGI, que implementa uma topologia em malha, em que todos os nós podem adormecer de uma forma sincronizada. O processo de sincronismo afecta o desempenho da rede, diminuindo a velocidade de reencaminhamento, mas permite a aplicação de técnicas de poupança energética em todos os nós da rede.

Estes *firmwares* podem ser aplicados nos módulos XBee consoante o rádio presente no módulo. O *hardware* do XBee é baseado em módulos rádio 802.15.4 de dois fabricantes, Freescale – XBee séries 1; e Ember – XBee séries 2. Estes por sua vez são comercializados nas versões OEM e Pro que diferem pelo XBee Pro possuir um amplificador que aumenta a potência de saída do sinal RF e melhora a sensibilidade, apresentando também um maior consumo. No anexo B tabela B.1 apresenta-se as características detalhadas das várias versões XBee.

Os vários *firmwares* funcionam de dois modos, modo AT e modo API. No modo AT funcionam como uma extensão RS-232 em que os dados que são entregues ao módulo são transmitidos de forma transparente. No modo API é preciso utilizar mensagens segundo a estrutura API do XBee. No Anexo A apresenta-se um resumo desta estrutura. Na figura 3.31 apresenta-se a organização geral da mensagem que a aplicação deverá usar para enviar os seus dados através da rede XBee. A mensagem é constituída por um cabeçalho, onde é indicado o comprimento e o tipo de mensagem, pelo endereço do destinatário, os dados a enviar e pelo resultado em 8 bits do cálculo do checksum.

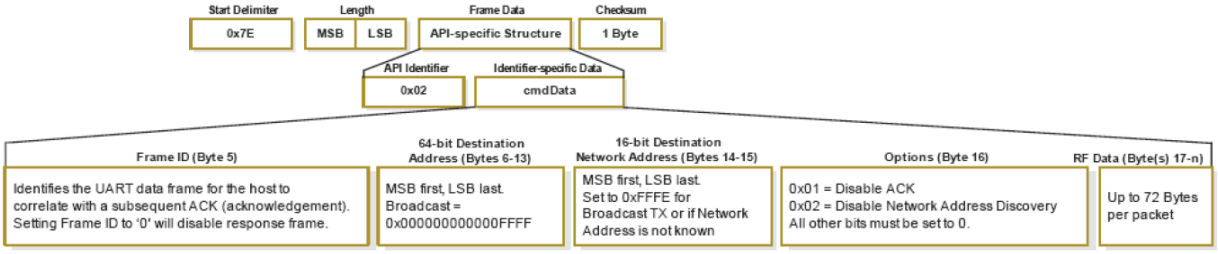


Figura 3.31 – Estrutura de um pacote de transmissão (TX).

Devido à mensagem ser enviada via UART, os dados são divididos em bytes (representados pelo valor em hexadecimal). Na figura 3.32 apresenta-se um exemplo prático de uma mensagem endereçada ao coordenador da rede *ZigBee*:

```
0x7E 0x00 0x1C 0x10 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x12 0x14 0x01 0x42 0x00 0x08 0x00
0xFE 0x00 0x01 0x02 0x0A 0x01 0xE2 0x71
```

Legenda:

0x7E	- Byte de inicio da mensagem
0x00 0x1C	- Comprimento da mensagem (28 bytes excepto checksum);
0x10	- Tipo de mensagem API
0x01	- ID da mensagem
0x0000000000000000	- Endereço de 64 bits do coordenador (pode ser substituído pelo endereço real)
0xFFFE	- Endereço de 16 bits (0xFFFE quando desconhecido)
0x00 0x00	- Número máximo de *hops* que uma mensagem pode dar, e byte de opções
0x12 a 0xE2	- Dados da mensagem (14 bytes de dados enviados pela aplicação)
0x71	- Checksum da mensagem.

Figura 3.32 – Exemplo de um pacote de transmissão (TX).

3.3.2 Criação de um nó sensor com XBee

Uma vez que o módulo XBee não é comercializado com sensores. Necessitou-se de implementar um módulo semelhante aos dois anteriores. Para isso foram adquiridos sensores

iguais aos dos módulos Tmote Sky e MicaZ, procurando uma igualdade de condições de teste. E assim desenvolveu-se um primeiro protótipo de sensor ambiental baseado no módulo XBee.

Utilizando uma placa de desenvolvimento, existente nos laboratórios da Universidade da Madeira, baseada no microcontrolador AT90USB1287 [73], desenvolveu-se uma primeira versão de um módulo de monitorização ambiental. A figura 3.33 apresenta o protótipo desenvolvido.

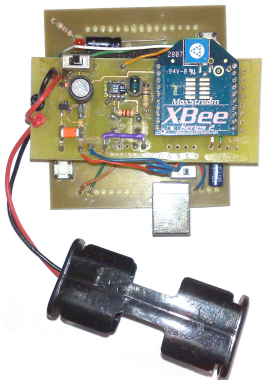


Figura 3.33 – Módulo de monitorização ambiental.

Para alimentar o circuito utilizou-se um par de baterias em série de formato AA. Estas apresentam uma tensão nominal de 1.2 V e 2500 mAh de carga.

Como verificou-se anteriormente as baterias apresentam uma tensão inferior à mínima necessária, e inclusive não se mantém constante em toda a carga útil. Assim, optou-se por utilizar um conversor DC-DC (NCP1400) para converter e regular a tensão das baterias para 3V. Evitando o problema que ocorre com os módulos MicaZ, ilustrado na figura 3.25.

Para avaliar o módulo, desenvolveu-se uma pequena aplicação em Java para o computador receber as mensagens provenientes do nó e armazená-las numa base de dados. Para isso foi também desenvolvido uma pequena base de dados em MySQL. A aplicação, recebe a mensagem (dentro dos moldes da mensagem da figura 3.31), separa os dados dos diferentes sensores e cria uma *query* de SQL para inserir os dados na tabela. A partir do código gerado para a página *Web* do MicaZ, acedeu-se aos dados armazenados e observou-se os resultados obtidos.

Na figura 3.34 é apresentado dois gráficos com as leituras do sensor de humidade e temperatura. Os gráficos correspondem a três dias, sendo a meia-noite o início dos dados. Mais uma vez pode-se reparar que o aumento da temperatura corresponde à descida da humidade.

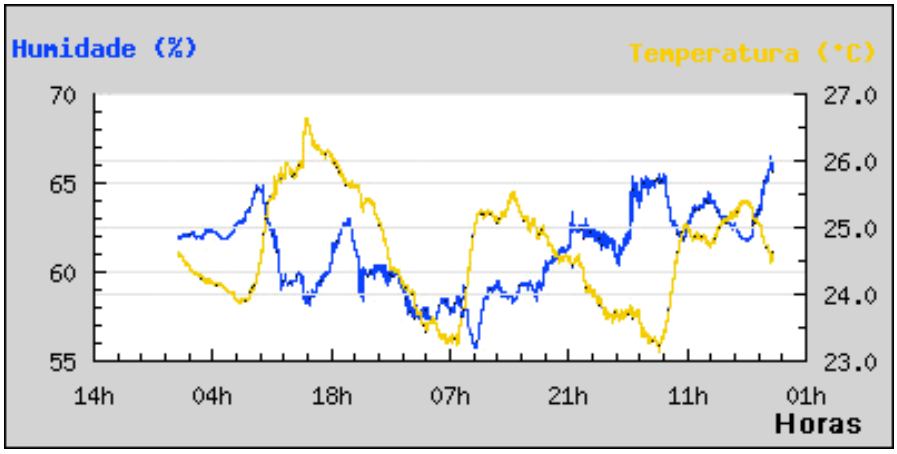


Figura 3.34 – Registo de humidade e temperatura do primeiro módulo de monitorização ambiental.

A figura 3.35 apresenta a evolução da intensidade luminosa durante dois dias consecutivos. O módulo foi colocado no interior do laboratório, junto de uma janela.

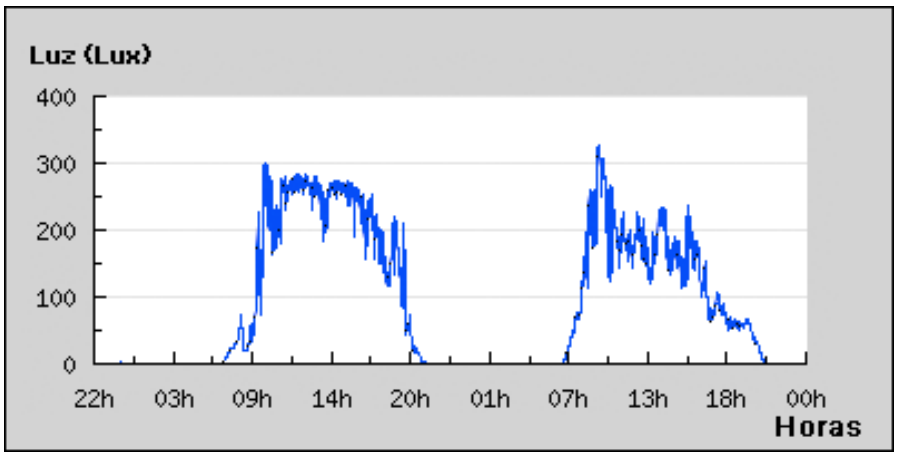


Figura 3.35 – Registo de luminosidade do primeiro módulo de monitorização ambiental.

Um objectivo do trabalho era a obtenção de baixo consumo. Para isso explorou-se a possibilidade de adormecer os componentes do sistema. Foi necessário estudar os tempos necessários de ciclo activo de cada componente. Visto o sensor de humidade e temperatura já activar o baixo consumo automaticamente, ou seja, entra no estado de adormecido sempre que possível. Restou controlar o adormecer do rádio XBee e do microcontrolador.

Após a optimização do programa de controlo do ciclo activo dos componentes, conseguiu-se atingir uma duração de 35 dias de monitorização, a uma periodicidade de amostragem de 1 minuto, para uma carga completa das baterias. Na figura 3.36 apresenta-se a evolução da descarga das baterias para um ciclo completo de energia.

Registou-se com recurso a um amperímetro, um consumo médio de 2,5 mAh. Nesta primeira versão observou-se o consumo de cada componente. Verificou-se, assim, que o conversor DC-DC é dos componentes que mais contribuem para o consumo. Dado que o módulo está adormecido pela maior parte do tempo, e nesta situação, o conversor DC-DC apresenta baixa eficiência, na ordem dos 50%, aumentando o consumo do estado adormecido para 2 mAh. Outro factor importante é o uso de um microcontrolador subdimensionado para as necessidades do módulo que pode ser substituído por um de menor consumo.

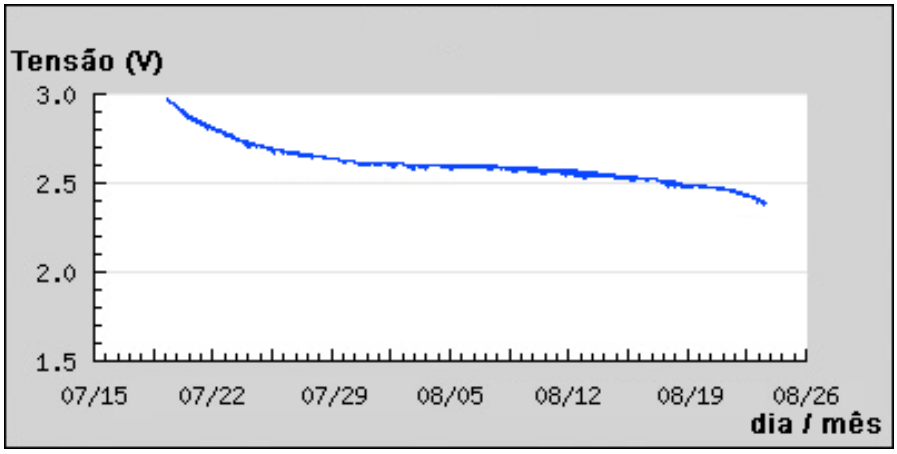


Figura 3.36 – Curva de descarga da tensão das baterias de alimentação.

O objectivo de criar um módulo análogo ao existente nos laboratórios foi atingido, por um custo inferior aos 80 Euros.

Com este módulo conseguiu-se apreender todas as partes intervenientes num sistema embebido de redes de sensores sem fios.

3.4 Estudo de Propagação do Sinal

É importante estudar o comportamento do sinal rádio e o alcance da ligação, de forma a poder-se tomar a decisão de qual topologia de rede usar e como distribuir a rede. Com recurso ao analisador de espectros, modelo FSH6 da Rohde & Schwarz, e uma antena de referência, observou-se a performance dos vários kits estudados.

3.4.1 Construção da Antena Dipolo

Foi necessário construir uma antena de referência para o estudo de propagação. Escolheu-se o dipolo de meio comprimento de onda, por ser uma antena conhecida e apresentar um diagrama de radiação horizontal circular.

Geralmente a guia de onda que interliga a antena aos equipamentos é um cabo coaxial. Como este é um meio inerentemente não balanceado, por apresentar um condutor externo de maior área que o interno, a ligação à antena não será igual, provocando o não balanceamento. Há necessidade de usar um *balun* (*balance to unbalance*). Baseou-se a construção da antena no *balun* coaxial [74]. Na figura 3.37 apresenta-se o esquema da antena e *balun*, adaptado de [74].

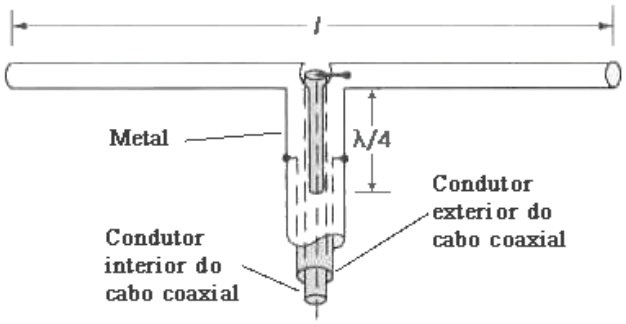


Figura 3.37 – Esquema da antena dipolo de meio comprimento de onda com *balun* coaxial.

A impedância do *balun* é importante que seja igual à do equipamento a que vai ser ligado, que é de 50 Ohm. O cálculo da impedância é dado pela relação do diâmetro exterior do condutor central, com o diâmetro interior do metal exterior [75], dada pela seguinte expressão:

$$Z_0 = \frac{138}{\sqrt{\varepsilon}} \times \log\left(\frac{D_{metal}}{d_{central}}\right) \Omega \tag{3.1}$$

Não havendo a disponibilidade de condutores de todos os diâmetros, utilizou-se a seguinte combinação, que aproxima a relação pretendida: condutor central de 1,6 mm de diâmetro externo, e um tubo com 4 mm de diâmetro interno para formar o metal externo do *balun*. O comprimento *l* da antena é de meio comprimento de onda, como queríamos que a antena funcionasse na gama de frequências de trabalho dos rádios em estudo, definiu-se a frequência de 2420 MHz. O comprimento de onda é dado pela expressão 3.2, assim meio comprimento de onda é 6,2 cm e um quarto do comprimento de onda é de 3,1 cm.

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8}{2{,}42 \times 10^9} = 0{,}124 \ \ metros \tag{3.2}$$

Construiu-se a antena, com uma extensão de cabo coaxial para facilitar a colocação de um conector e permitir o manuseamento da antena. Utilizou-se um cabo coaxial modelo RG58, com 70 cm de comprimento. Este cabo apresenta 50 Ohm de impedância e 0,8 dB de atenuação para os 70 cm na frequência dos 2.4 GHz. À outra extremidade do cabo fixou-se um conector do tipo N, o mesmo tipo que o analisador de espectros. Para os testes desprezou-se a atenuação introduzida pelo conector. Apresenta-se na figura 3.38 a antena desenvolvida.

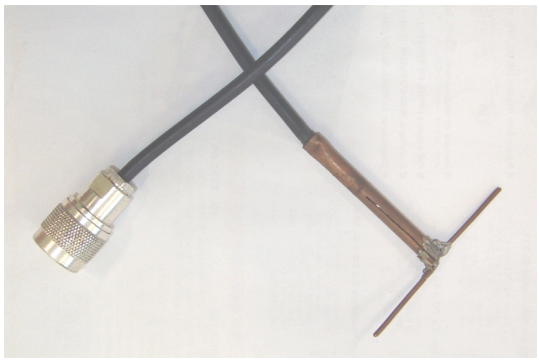


Figura 3.38 – Antena dipolo de meio comprimento de onda desenvolvida.

Com recurso a um medidor de SWR (*Standing Waves Ratio*), o ORITEL RO600, mediu-se a desadaptação da antena em relação à impedância de 50 Ohms. O valor inicial registado foi de 1,55. Diminuindo o comprimento total das hastes da antena, como indicado em [74], foi possível melhorar o SWR da antena para 1,35 o que é considerado um bom valor, visto que assim a desadaptação apenas perde 0,02 dB.

Era importante medir o ganho da antena para poder utilizá-la como referência. Um método seria medir o ganho da antena por comparação de outra. Não havendo antenas de referência no Laboratório foi necessário utilizar um método indirecto. O método das três antenas [76]. Em que com a medição da combinação de três antenas obtém-se o ganho individual delas. Esta medida de nível de sinal recebido combinado, é representada pelo sistema de equações 3.3. Em que *Gx* é o ganho das antenas, *Pta* é a potência no espaço livre e *Prb* é a potência registada na antena à distância *r*.

$$
\begin{aligned}
(G_a)_{dB} + (G_b)_{dB} &= 20\log_{10}\left(\frac{4\pi R}{\lambda}\right) + 10\log_{10}\left(\frac{P_{rb}}{P_{ta}}\right) \\
(G_a)_{dB} + (G_c)_{dB} &= 20\log_{10}\left(\frac{4\pi R}{\lambda}\right) + 10\log_{10}\left(\frac{P_{rc}}{P_{ta}}\right) \\
(G_b)_{dB} + (G_c)_{dB} &= 20\log_{10}\left(\frac{4\pi R}{\lambda}\right) + 10\log_{10}\left(\frac{P_{rc}}{P_{tb}}\right)
\end{aligned}
\tag{3.3}
$$

Do sistema de equações retira-se $(G_a)_{dB}$, $(G_b)_{dB}$ e $(G_c)_{dB}$. Utilizou-se três antenas: um monopolo de meio comprimento de onda, o dipolo e uma antena *yagi*. As medidas foram realizadas no exterior a cinco metros de altura, afastando assim as antenas do efeito de proximidade ao solo, no sentido de aproximar às características do espaço livre. O sistema de teste prático utilizado é apresentado na figura 3.39.

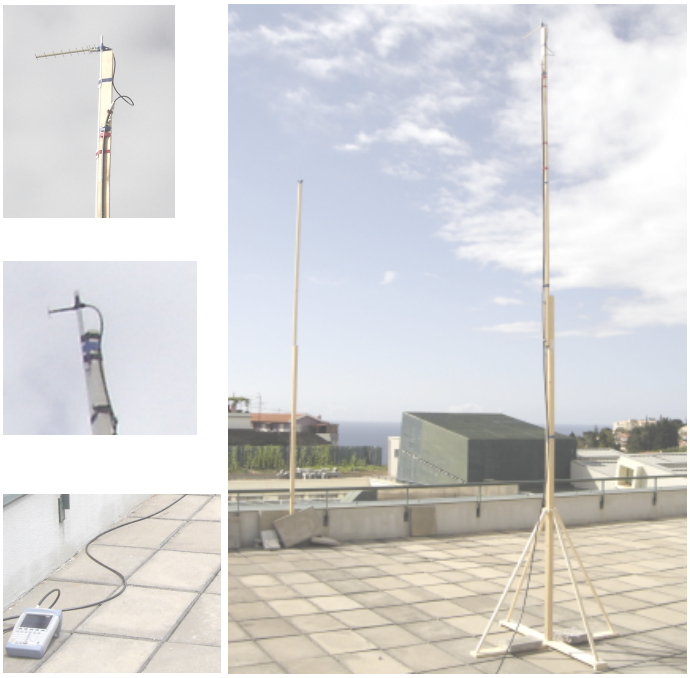


Figura 3.39 – Sistema de medição das características da antena dipolo.

Registou-se os valores de potência as várias combinações de antenas desde 0,5 m até 8 m. Para minimizar os erros do sistema, dos vários valores escolheu-se a distância para o qual as curvas de atenuação aproximam-se do comportamento do espaço livre. Registou-se um nível de sinal de -37,8 dBm para a relação dipolo-*yagi*, -47,1 dBm para a relação monopolo-dipolo e -37,4 dBm entre o monopolo-*yagi*. Resolvendo o sistema de equações de 3.3, obteve-se o ganho das várias antenas: monopolo – 1,8 dBi; dipolo – 1 dBi e a *yagi* – 9,8 dBi. Tendo em conta as perdas do cabo coaxial, o ganho da antena dipolo é de 1,8 dBi. Valor que foi usado para normalizar as medidas realizadas no estudo de propagação do sinal.

3.4.2 Diagramas de Radiação dos Nós

O ideal num nó sensor sem fios móvel é apresentar um diagrama de radiação omnidireccional, para que a comunicação com a rede seja independente da sua orientação.

O MicaZ utiliza uma antena monopolo de ¼ de λ (comprimento de onda). O diagrama de radiação teórico de um monopolo é igual ao do dipolo de meio comprimento de onda contudo quando introduzido no componente o seu diagrama é ligeiramente alterado. A figura 3.40 mostra o padrão medido numa câmara anecónica com a antena na vertical [18]. Como se pode observar o principal nulo está 9 dB abaixo do máximo do diagrama de radiação.

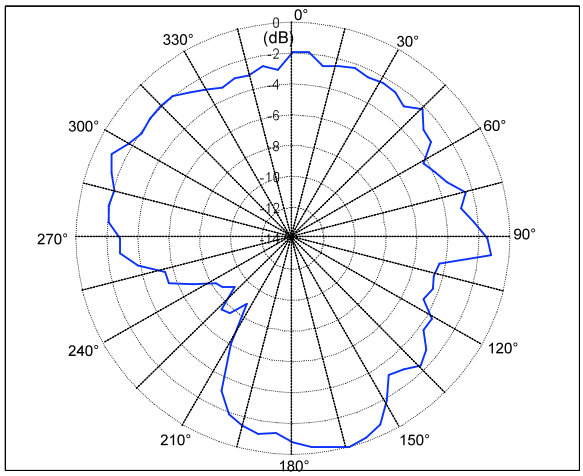


Figura 3.40 – Diagrama de radiação do MicaZ.

O Tmote Sky, utiliza a antena F invertido. A figura 3.41 apresenta o diagrama de radiação com o módulo na horizontal e na vertical [69], como se pode observar o diagrama de radiação não é omnidirecional em ambos os planos. Sendo assim, o nível do sinal recebido depende da orientação da antena do módulo.

(a) (b)

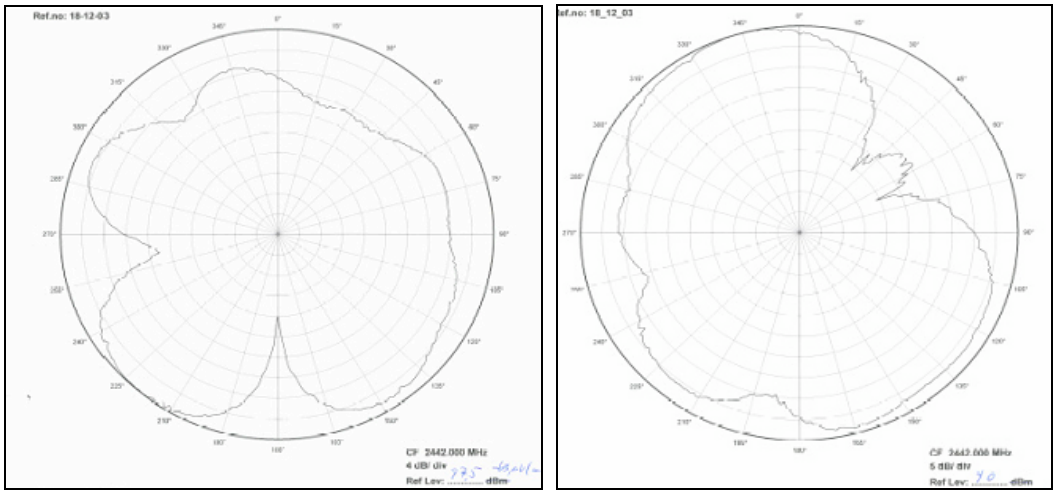


Figura 3.41 – Diagrama de radiação do Tmote: a) montagem horizontal; b) montagem vertical.

Os módulos XBee são comercializados com antena chip, antena monopolo ou conector SMA para conectar a antenas externas. Enquanto o monopolo apresenta um diagrama de radiação próximo de circulo no plano perpendicular à antena, a antena chip não é tão uniforme. Por isso apresenta melhor resposta em certas orientações. A figura 3.42 a) apresenta o diagrama de radiação para a antena monopolo e a figura 3.42 b) para a antena chip. Para comparação ambos os gráficos estão normalizados para uma antena dipolo conectada ao módulo XBee.

Com recurso ao analisador de espectro e uma antena dipolo de meio comprimento de onda registou-se a resposta da antena *chip* do módulo XBee embebido no módulo de monitorização ambiental, versão 1. As medidas foram feitas a 5 metros de altura e 4 metros de distância entre módulo XBee e antena receptora, de forma a aproximar o sistema de medida às condições do espaço livre. A figura 3.43 apresenta os resultados obtidos para a montagem horizontal, figura 3.43 a), e vertical na figura 3.43 b).

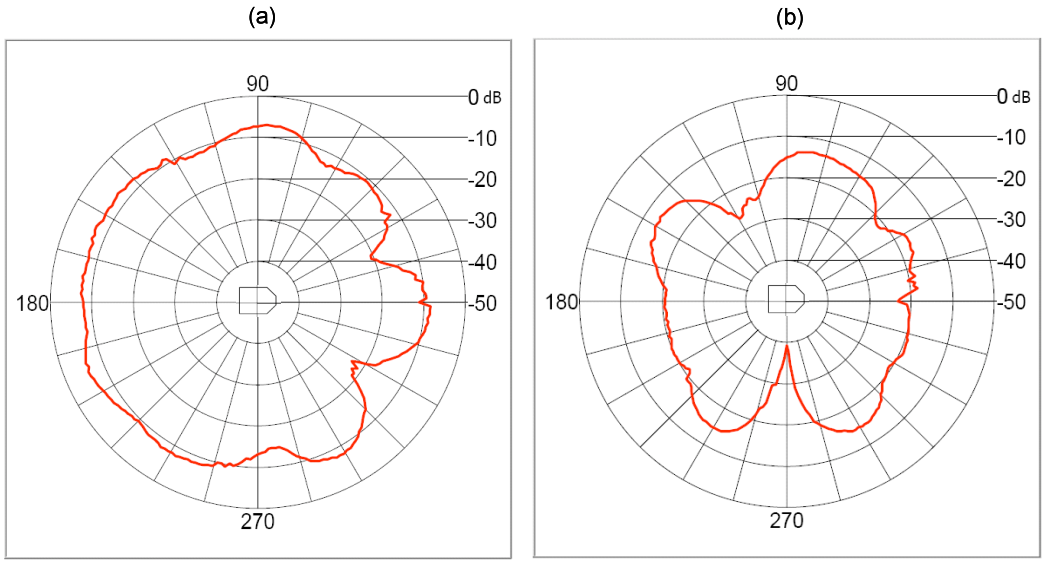


Figura 3.42 – Diagrama de radiação do XBee: a) antena monopolo; b) antena chip

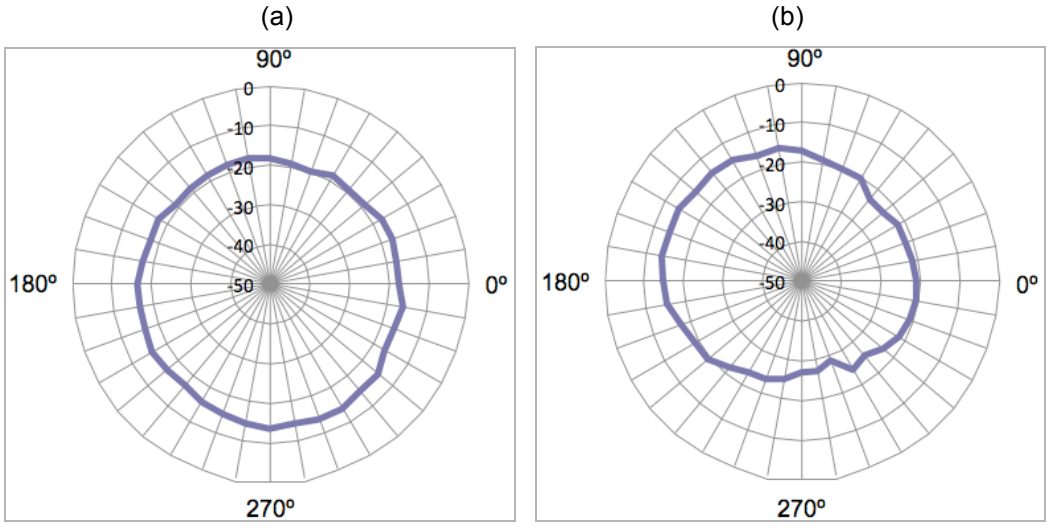


Figura 3.43 – Medição da resposta da antena chip: a) montagem vertical; b) montagem horizontal.

Dos gráficos 3.42 b) e 3.43 b) verifica-se que é na zona de nulo os valores mais baixos registados com o sistema de medida, mas o padrão de radiação da antena chip altera-se quando montado no módulo de monitorização ambiental.

3.4.3 Relação da Potência de Sinal Recebido e RSSI com a Distância

As medidas da potência de sinal recebido realizadas neste trabalho foram obtidas com o analisador de espectro. Outra possibilidade é usar o RSSI (*Received Signal Strength Indicator*) parâmetro disponível nos módulos *ZigBee* testados. Segundo a folha de características do rádio CC2420 utilizado pelo MicaZ e Tmote a leitura de RSSI apresenta um

erro de nível na ordem dos ±6 dB e uma diferença de linearidade de ±3 dB. A folha de características do rádio Ember250 usado pelo XBee não define esses valores apenas indica que a resposta do RSSI é linear na gama de medida.

Realizou-se um conjunto de medidas para obter a potência do sinal recebido em relação à distância ao nó sensor. Foi utilizado o módulo XBee com antena monopolo como emissor e as medidas foram registadas com um analisador de espectro em conjunto com a antena dipolo de meio comprimento, e também pelo valor de RSSI com um outro módulo XBee como receptor. Emissor e receptor foram colocados a 1 metro de altura, numa situação de linha de vista, mas com a presença de automóveis e pessoas no ambiente circundante. O ambiente descrito foi escolhido no sentido de obtermos a relação de atenuação com a distância numa situação próxima da real. As medidas foram registadas a cada 2 metros de distância.

A figura 3.44 apresenta os valores medidos, já sem o contributo dos ganhos das antenas e considerando uma potência de emissão de 0 dBm. A linha ponteada, mostra os valores medidos com o analisador de espectro. Para o RSSI foram registados cinco valores em cada posição, e observou-se que pequenas oscilações na posição do componente ou no ambiente circundante os valores alteravam-se significativamente. A linha contínua mostra os valores médios de RSSI que, embora sigam os valores registados pelo analisador, são em média 2,5 dB superiores. As duas linhas em torno do valor médio de RSSI do gráfico representam a variação máxima e mínima de RSSI obtida em cada ponto, que chega a atingir a certas distâncias os 14 dB de amplitude, com um desvio padrão de 12,5 dB.

A função para o decaimento do sinal com a distância pode ser obtida a partir do modelo log-normal [77] dada por,

$$P_L(d) = P_L(d_0) + 10n\log_{10}\left(\frac{d}{d_0}\right) + X_\sigma \qquad (3.1)$$

onde n é o expoente de perda e indica a taxa à qual o sinal é atenuado com a distância (para o espaço livre n=2). $P_L(d_0)$ é as perdas no espaço a uma distância conhecida d_0 que está no campo distante da antena (tipicamente 1 m para os sistemas análogos ao em análise) e X_σ representa uma média com distribuição Gaussiana em torno do zero (em dB) com desvio padrão σ, e reflecte a variação esperada em torno do valor médio da potência recebida. Na figura 3.44 é ainda apresentado, pela linha tracejada, o decaimento do sinal no espaço livre.

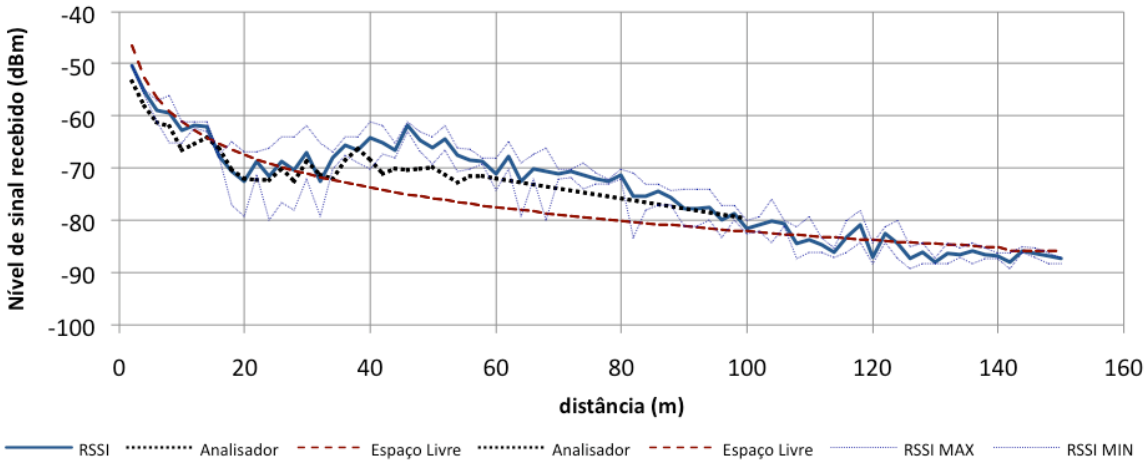


Figura 3.44 – Variação do sinal recebido com a distância ao nó num ambiente com reflexões.

Comparando os valores registados observa-se que embora entre os 35 e 100 metros o sinal recebido seja superior ao do espaço livre, justificado pelas reflexões do ambiente circundante, no início e fim a taxa de atenuação é superior ao do espaço livre apontando para um *n* em média igual a 2.

Com a linha de tendência logaritmica sobre os valores obtidos, é possível extrair o expoente de perdas no espaço. O resultado é *n*=2,01. O resultado está em consonância com a literatura, para espaços rodeados de edifícios mas com linha de vista entre emissor e receptor.

Baseado na equação (3.1) quando o receptor mede o valor P_L(d), a distância estimada ao emissor é dada por,

$$d = d_0 \times 10^{\frac{P_L(d_0)-P_L(d)}{10n}} \tag{3.2}$$

Como a sensibilidade do XBee é de -96 dBm, idealmente seria atingida aos 450 metros. Isto pode-se compreender com a existência de linha de vista, mas como existe uma flutuação do sinal de ±12 dB o receptor pode perder o sinal aos 120 metros. Situação que foi observada durante as medições.

3.4.4 Relação da Potência de Sinal Recebido com a Altura

A proximidade da antena ao solo altera a sua performance e, consequentemente, afecta o nível de sinal recebido. Para medir o efeito, registou-se a variação do nível de sinal recebido com a variação da altura da antena emissora em relação ao solo, considerando a antena receptora a 93 cm acima do solo e a uma distância de 5,4 m do emissor. O nó foi elevado desde os 2 cm até os 236 cm de altura. Os resultados registados são apresentados na figura 3.45. Mais uma vez, é claro o efeito das reflexões. Observou-se uma grande flutuação do sinal. Isto é devido à variação do diagrama de radiação com a distância ao solo para pequenas alturas e também em relação ao tecto para distâncias maiores. Para confirmar a influência do solo, a deflexão devido a um solo perfeito é dada por,

$$F = 2\left|\sin\left(\frac{\beta h_1 h_2}{d}\right)\right| \tag{3.3}$$

onde $\beta=2\pi/\lambda$, λ é o comprimento de onda, h_1 é a altura da antena emissor, h_2 é a altura da antena receptora, e d é a distância entre antenas. Para comparação, o factor definido pela expressão anterior é destacado pela linha tracejada na figura 3.44, onde os máximos foram movidos para -45 dB para permitir a comparação.

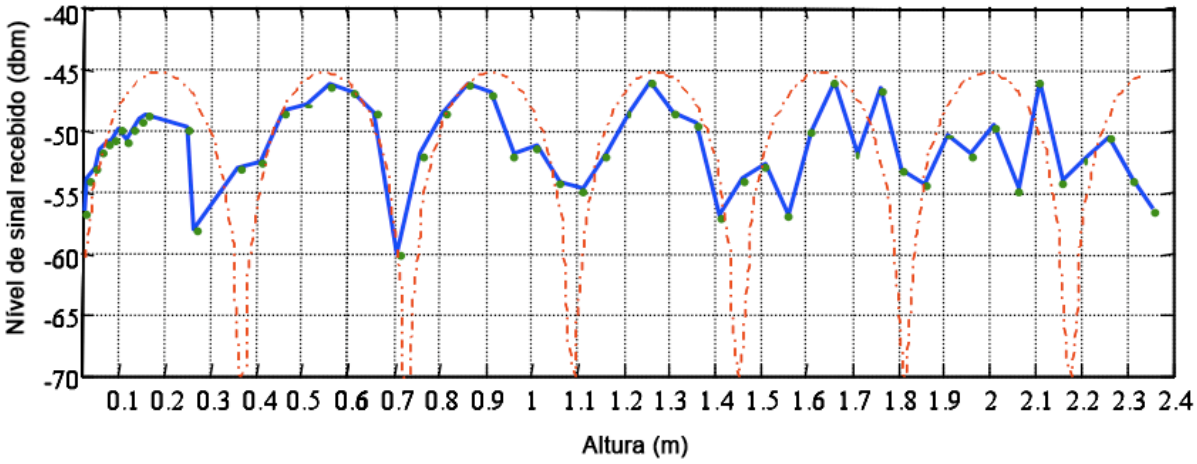


Figura 3.45 – Variação do sinal recebido com a altura do nó sensor.

Na figura 3.46 apresenta-se a variação do sinal recebido ao variar a altura de ambas as antenas (recepção e emissão) em relação ao solo, desde os 2 cm até os 26,5 cm.

A linha tracejada corresponde à linha de tendência das medidas. A 25 cm de altura a média do sinal recebido é 10 dB superior que a 2 cm.

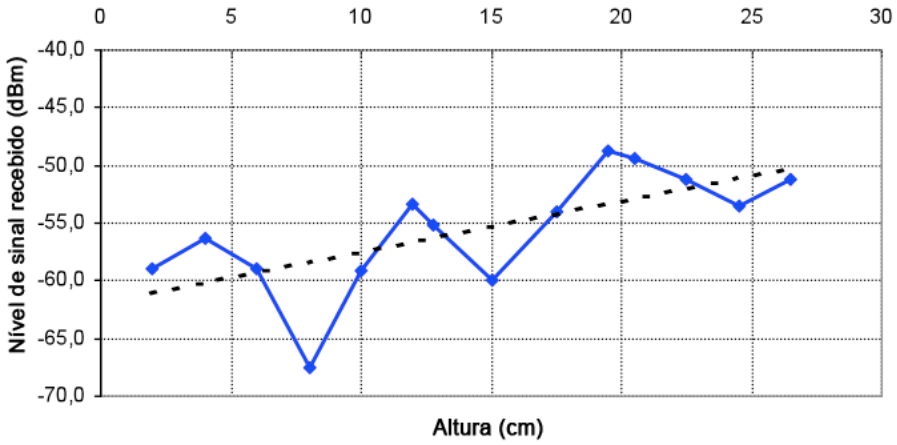


Figura 3.46 – Variação do sinal recebido com a altura do nó sensor e da antena receptora.

Num meio real rodeado de objectos, os nulos são na maior parte das vezes preenchidos devido às reflexões.

3.5 Comparação dos Protótipos de RSSF

De forma geral as redes testadas permitiram a criação de redes *ZigBee* completamente funcionais. Aprendeu-se, não só, como funciona as redes de sensores sem fios baseadas em *ZigBee*, como também a identificar requisitos importantes de uma rede de sensores.

Com os resultados dos testes elaborou-se a tabela 3.3, de comparação das características registadas, que distinguiram as várias plataformas.

TABELA 3.3 – CARACTERÍSTICAS QUE DISTINGUEM AS REDES IMPLEMENTADAS.

	Autonomia	Fiabilidade	Alcance	Preço
Tmote Sky	4 dias (sempre ligado)	Baixa	50 a 125 m	200 €
MicaZ	4 dias (sempre ligado)	Boa	30 a 100 m	350 €
XBee	45 dias	Boa	40 a 120 m	80 €

O Tmote Sky utiliza o mesmo rádio que o MicaZ, o CC2420 da Chipcom, e o mesmo sistema operativo, TinyOS, mas distingue-se por já englobar os sensores e antena de melhor desempenho na mesma placa, e ainda apresentar um interface de programação de acesso rápido via USB. Estas características apontavam para um melhor desempenho em relação ao MicaZ, mas durante os testes o *software* fornecido apresentou problemas de fiabilidade, pois os nós necessitaram de ser reiniciados ao fim de um número aleatório de dias de trabalho.

O MicaZ, apresenta uma plataforma com uma estrutura modular de componentes e de *software*, direccionados para os testes. Pensando na aplicação final do tema deste trabalho, registou-se os seguintes pontos fracos de desempenho: o módulo MicaZ emite à mesma potência de saída que os Tmote Sky, mas obteve um valor na recepção sempre inferior, da ordem dos 10 dB. O que se traduz num menor alcance. Verificou-se também um tempo de registo do nó na rede, irrazoável, atingindo por vezes um minuto.

O XBee apresentou um desempenho constante, reflectindo as características do fabricante.

Ao analisar os vários módulos quanto ao custo de desenvolvimento, constatou-se que: o MicaZ apresenta uma plataforma, proprietária, rígida, que embora venha preparada para a ligação de vários sensores, os parâmetros de controlo, como período de amostragem, controlo energético dos vários componentes, são limitados às possibilidades apresentadas. O Tmote Sky é uma plataforma de código aberto, por outro lado força a que o desenvolvedor tenha de ser um especialista em todas as componentes que envolve o módulo, desde o *Hardware*, sistema operativo, protocolo *ZigBee* até a interface com o computador. O XBee, embora seja uma solução sem a integração de sensores, a implementação completa da rede e a simplicidade da API, permite ao desenvolvedor preocupar-se apenas com a aplicação, apresentando-se como o módulo com o menor custo de aprendizagem.

Verificou-se as aplicações e *firmware* que acompanham os *kits* de RSSF permitem configurar parâmetros básicos da rede e do funcionamento dos nós. É possível configurar periodicidades de amostras e até alarmes a certos níveis dos parâmetros monitorizados. Mas actualmente ainda não oferecem uma forma amigável de programar o *hardware* com inteligência adicional e automatização. Requerendo sempre dos desenvolvedores de aplicações um conhecimento enorme da programação de sistemas embebidos, como referia Kristi Hobs (National Instruments) [78]. Por exemplo se a aplicação requerer elevada taxa de amostragem de um parâmetro, pode ser embebido um certo nível de processamento e passar apenas para a rede alguma parametrização dessa informação.

A localização automática e o baixar do preço podem permitir substituir as etiquetas RFID que necessitam da proximidade do leitor, ou seja, é necessário conhecer a sua localização. Embora para entrar no mundo industrial ainda é necessário comprovar o tempo de funcionamento de 99,999% [79].

4 DESENVOLVIMENTO DO SISTEMA

Do estudo realizado sobre sistemas de monitorização de atletas e rede de sensores sem fios, apresenta-se neste capítulo o desenvolvimento de uma proposta para um sistema monitorização de provas de educação física. Descreve-se a arquitectura do sistema, os diferentes módulos que o integram, e as diferentes interfaces de utilizadores.

O sistema apresentado, integra a monitorização ambiental com a monitorização de atletas numa solução sem fios, de baixo custo e em tempo real. O sistema integra duas tecnologias sem fios, *ZigBee* e *Bluetooth* oferecendo a agilidade de interfaces com o utilizador. Sendo um passo em direcção às redes ubíquas.

4.1 Requisitos do Sistema

Do estudo realizado sobre redes de sensores sem fios e da observação de várias provas de educação física, identificou-se os principais requisitos para um sistema de monitorização de atletas durante o exercício físico, os quais enumeram-se de seguida:

O sistema deverá ser compreendido por um conjunto de módulos sem fios capazes de monitorizar, em tempo real, provas de educação física;

O sistema deverá monitorizar parâmetros físicos de cada atleta;

O sistema deverá monitorizar parâmetros ambientais da área de prova;

O sistema deverá reencaminhar os parâmetros monitorizados para um computador central onde serão armazenados;

Os dados deverão ser apresentados no computador através de um *software* adequado;

Estes dados deverão estar acessíveis de uma forma simplificada na Internet;

Os módulos de atletas deverão monitorizar os seguintes parâmetros: velocidade; aceleração; batimento cardíaco e posição.

Os módulos de monitorização ambiental deverão registar os seguintes parâmetros: temperatura; luminosidade; humidade; monóxido de carbono; dióxido de carbono e oxigénio;

O sistema deverá ser capaz de medir a amplitude de interesse dos parâmetros a monitorizar;

O sistema deverá abranger toda a área de prova;

Os módulos do sistema deverão apresentar dimensões reduzidas, permitindo a sua portabilidade, não influenciando assim a actividade do atleta;

Os módulos do sistema deverão apresentar baixo consumo, permitindo ser alimentado por pequenas baterias e oferecer uma autonomia mínima da duração de uma prova;

O sistema deverá ser escalável, ou seja, permitir o aumento de atletas monitorizados;

Os módulos para monitorização ambiental deverão ser dotados com interface *Bluetooth*, fornecendo aos atletas / treinadores informação sobre o ambiente de treino através de uma aplicação adequada ao telemóvel;

4.2 Opções Tomadas

Neste ponto apresentam-se as opções tomadas que suportam a arquitectura definida para o sistema.

A disponibilidade dos módulos XBee no laboratório e os bons resultados, apresentados no capítulo anterior, determinaram a eleição do XBee como módulo base para o desenvolvimento do sistema de monitorização de provas de educação física. Nestes é usado o protocolo *ZigBee*. Optou-se à partida usar a topologia em malha em que todos os nós são *FFD*, para não impor limitações no estabelecimento da rede, ou seja, os atletas poderão ligar ou desligar os seus módulos de monitorização sem preocupações de como funciona a rede.

Tendo em conta, que cada malha da rede tem um alcance máximo (com garantia de recepção) de 120 metros no exterior, em linha de vista, como demonstrado pelo ponto 3.4.3. No caso da área de treino ter um comprimento superior aos 120 metros, torna-se necessário garantir a distância máxima entre nós, não sendo admissível limitar os atletas, especialmente se a actividade física for realizada num longo percurso. Definiu-se assim, a necessidade de espalhar torres de encaminhamento pelo percurso à distância máxima de 120 entre cada uma, desde a *gateway* do sistema.

O sistema deverá monitorizar atletas durante a actividade física e ambiente circundante. Para isso definiu-se os seguintes parâmetros de monitorização: o ritmo cardíaco, a posição, a velocidade de cada atleta, a temperatura ambiente, humidade, luminosidade, concentração de CO_2, O_2 e CO.

A posição e a velocidade são obtidas através de um GPS. Nos casos de não haver recepção de sinal GPS, situação do exercício físico ser realizado dentro de edifícios, é usado um acelerómetro para obter a velocidade e um sistema de posicionamento relativo, utilizando como referência às torres de encaminhamento. O acelerómetro é utilizado na cintura devido a uma convergência de condicionantes: alcance ao sensor de batimento cardíaco, elevação do rádio em relação ao solo permitindo uma menor afectação do solo como demonstrado pela figura 3.38, e redução do hardware em relação ao uso de um pedómetro no calçado, já que este necessita de um segundo rádio para comunicar com o módulo principal.

Foram integradas várias tecnologias de comunicação no sistema, de forma a oferecer várias interfaces para os utilizadores. Isto, na procura da implementação do axioma das redes ubíquas. Para isso o sistema integra as RSSF, a Internet e *Bluetooth*, maximizando o acesso à informação, tendo em conta a elevada disponibilidade de equipamentos que incorporam estas tecnologias, como por exemplo o telemóvel.

4.3 Arquitectura do Sistema

O sistema desenvolvido tem por base a tecnologia de rede de sensores sem fios *ZigBee*. É usada a topologia de rede em malha.

O sistema é composto por quatro tipos de nós:

- Coordenador – Nó composto por um rádio *ZigBee* e interface USB ligado ao computador. Este nó coordena a rede *ZigBee* e serve de *gateway* para a toda a informação da rede;
- Nós de referência ou Routers – Estes nós serão colocados ao longo do curso da prova, servindo de marcos para o mesmo. São compostos por um GPS e um rádio *ZigBee*. O rádio *ZigBee* destes módulos é mantido sempre activo e no alcance dos routers anteriores e posteriores, por forma a garantir a existência de

pelo menos um caminho de rede para a *gateway*, para assim dar acesso à rede *ZigBee* aos nós de atletas que circulam por todo o percurso da prova;

- Nós de atletas – São compostos por um acelerómetro, sensor de batimento cardíaco, GPS e rádio *ZigBee*. Estes registam e comunicam para a *gateway* a aceleração, batimento cardíaco e posição do atleta a cada segundo;
- Nós de monitorização ambiental – São compostos pelos sensores de: temperatura; luminosidade; humidade; monóxido de carbono; dióxido de carbono e oxigénio. E ainda pelo rádio *ZigBee*, rádio *Bluetooth* e um pequeno ecrã para visualização dos dados directamente no local. Estes nós registam e comunicam, a cada minuto, os diversos parâmetros ambientais, e ainda permitem ser colocados em qualquer ponto do percurso da prova.

O sistema funciona da seguinte forma: todas as mensagens dos nós de monitorização de atletas e de monitorização ambiental são reencaminhadas através da rede *ZigBee* para um computador. Este regista os diversos dados numa base de dados MySQL. Uma aplicação Java apresenta em tempo real sobre um mapa a localização dos vários atletas e respectivos parâmetros, incluindo os dados ambientais. Neste computador, também existe um servidor de páginas *Web* que aloja uma aplicação que permite o acesso remoto aos dados de monitorização ambiental. Para completar o sistema recorre-se a uma aplicação em Java ME para telemóveis que permite, aos atletas ou treinadores, acederem aos dados dos parâmetros ambientais bem como a posição onde se encontram.

A figura 4.1 apresenta a arquitectura proposta para o sistema de monitorização de atletas.

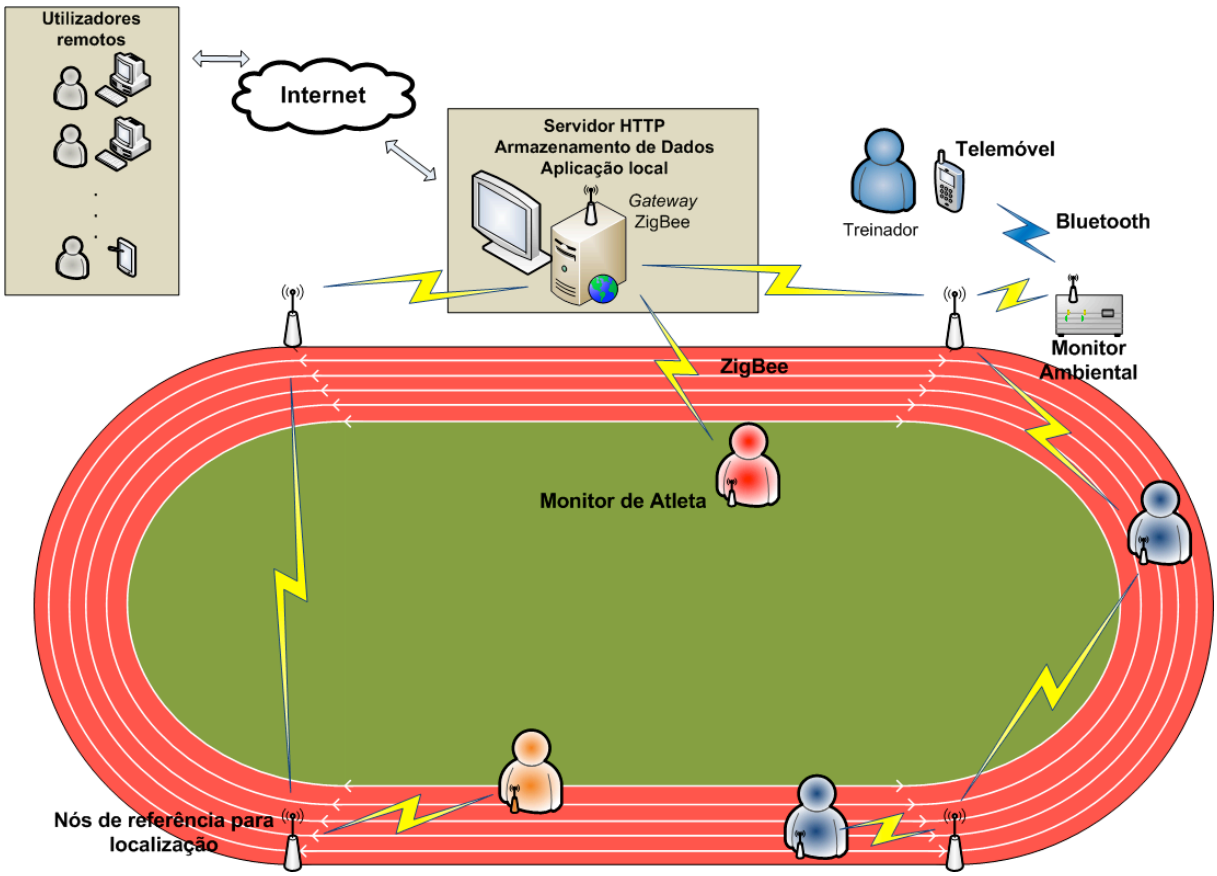


Figura 4.1 – Arquitectura do sistema de monitorização de atletas.

4.4 Cálculo da Velocidade e Posição do Atleta

No exterior, a velocidade e a posição actual de um atleta podem ser determinadas pela informação do GPS. No interior de edifícios, tal não acontece. Daí a necessidade de integrar outra solução, para cálculo da velocidade e posição do atleta no sistema, para o caso da actividade se realizar no interior de edifícios.

Já se encontram diversos dispositivos no mercado baseados num acelerómetro [30][31], que oferecem a funcionalidade de pedómetro com erros inferiores a 3%. Estudos também demonstram que o acelerómetro é adequado a aplicações de monitorização da actividade física de pessoas. Optou-se, assim, por utilizar um acelerómetro para a obtenção da velocidade do atleta dentro de edifícios.

O acto de andar de uma pessoa pode parecer bastante complicado se o analisarmos de um referencial fixo ao solo: as pernas vão sempre para a frente, com velocidade variável; durante algum tempo o pé desloca-se, com maior velocidade que a média da velocidade da pessoa e, ao tocar o solo, pára; quando o pé está parado em relação ao solo, a perna faz um movimento para a frente, como que rodando em torno dele; além disso, as pernas dobram-se ligeiramente quando andamos.

Medir a velocidade com precisão integrando continuamente a aceleração longitudinal, é complexo devido a erros inerentes ao sinal da aceleração ao longo do tempo. Para reduzir esses erros, as medidas de tempo e de aceleração do acto de andar podem ser divididas em passos. Os passos causam um movimento periódico da cintura humana, como representado na figura 4.2 [80]. O perfil de forma de arco apresentado assemelha-se ao funcionamento de um pêndulo simples invertido.

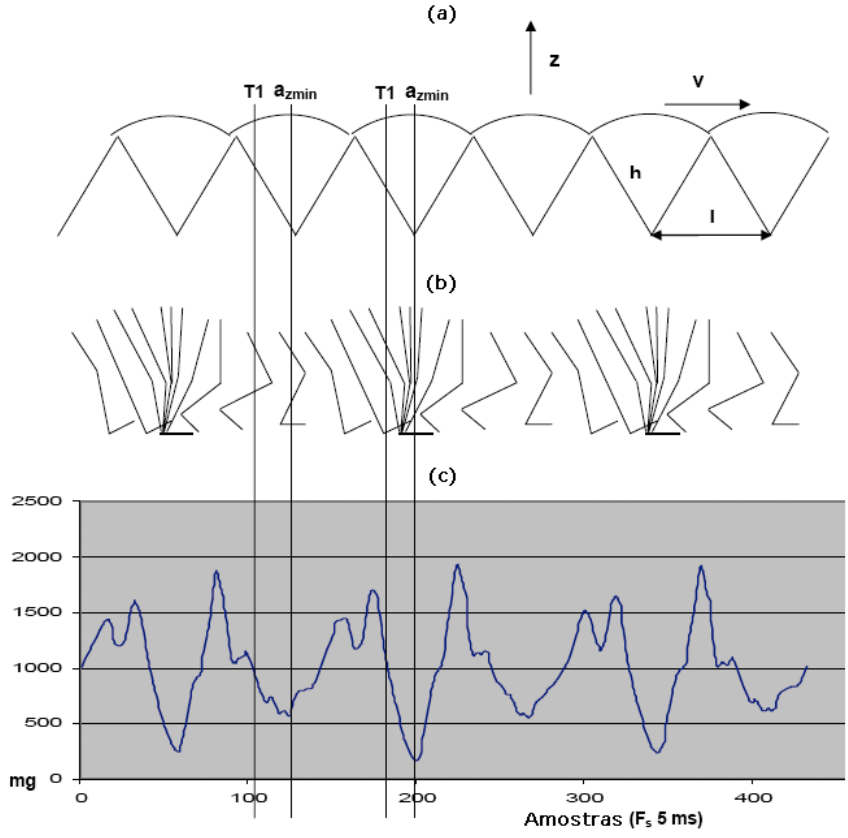


Figura 4.2 – Dinâmica do acto de andar: a) movimento em arco da cintura; b) perfil do passo do pé direito; c) aceleração durante os passos.

Assim, o comprimento de um passo pode ser integrado da velocidade angular do movimento análogo do pêndulo. A aceleração vertical da cintura e o comprimento efectivo da perna definem a velocidade angular e, logo a velocidade do passo (*v*), comprimento do passo (*l*) e distância total percorrida (*S*) [80], definido pelas seguintes expressões:

$$v = Velocidade = \sqrt{h} \times \sqrt{(1g - a_{z\min})} \tag{4.1}$$

$$l = Comprimento\ de\ 1\ \ passo = T \times v = (T_{1passo\ n+1} - T_{1passo\ n}) \times v \tag{4.2}$$

$$S = Distância\ \ percorrida = \sum_{i=2}^{n} l_i \tag{4.3}$$

Nas redes de sensores sem fios os métodos de localização incluem, além do GPS, nós balizas (ou âncoras) de referência e métodos baseados na proximidade dos vizinhos. GPS é uma solução simples, mas no caso de obstrução o sistema pode não funcionar. O método de balizas faz uso de nós de referência que conhecem a sua própria posição, que assim, permitem ao nó sensor inferir a sua posição. Este método apresenta problemas numa rede de grandes dimensões. A localização baseada na proximidade faz uso dos nós vizinhos para determinar a sua posição relativa e servirem de referência para os outros nós. Existem uma série de outras técnicas e algoritmos que fazem uso, de cálculos de probabilidade estatística, da medição da fase do sinal recebido em relação a um sinal de referência, de sistemas centralizados com laser e sistemas com ultra-sons [11]. Embora estes métodos ofereçam elevada precisão, como o sistema que utiliza o laser, eles exigem linha de vista com todos os nós da rede, ou elevados recursos de processamento.

No caso da aplicação e monitorização de atletas, em que se pretende colocar nós routers e dadas as dimensões da rede não serem enormes optou-se pela utilização de balizas de referência para inferir a posição dos nós sensores de atletas.

Do estudo de propagação do sinal, verifica-se que é possível, em certas condições, inferir a distância entre dois rádios pela medição do nível de sinal de um em relação ao outro, utilizando para isso a equação (3.2). Num local real longe das condições do espaço livre, o sinal rádio é afectado por diversos factores, como demonstrado pelas medidas da atenuação do sinal com a distância, figura 3.44. O sinal pode ter o mesmo nível para distâncias diferentes. Assim, requer que sejam tiradas várias medidas do nível de sinal, em cada posição, prevalecendo a média para o cálculo da distância.

Com a utilização de nós de referência pode-se medir a distância do atleta a cada um deles e, aplicando a trilateração [81], determinar a posição relativa do atleta.

Para isso faz-se uso das mensagens de *broadcast* do *ZigBee*, para as balizas emitirem um sinal de referência aos nós no seu alcance, e da potencialidade do XBee medir o nível de sinal correspondente a cada mensagem, denominado por RSSI. A mensagem é definida para dar apenas um salto (*hop*), chegando apenas ao primeiro nível de nós vizinhos. Os atletas que escutarem o *broadcast* medem o nível do sinal recebido e de seguida enviam esta informação para a *gateway*. No computador, com a informação da posição de cada nó de referência e com as medidas do nível de sinal dos vários *broadcasts*, através da trilateração, é calculada a posição do atleta, como representado na figura 4.3.

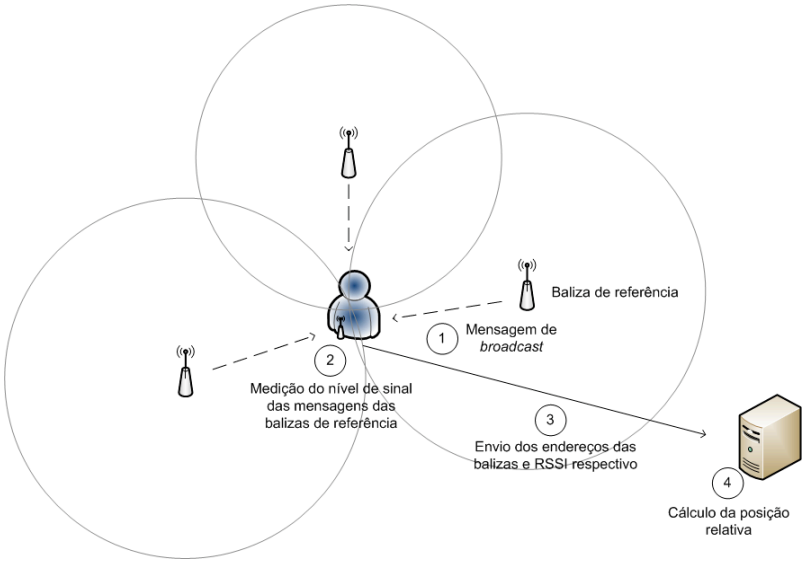


Figura 4.3 – Trilateração, processo para obtenção da posição.

A posição dos atletas pode ser obtida por três ou mais pontos de referência. Quanto maior o número de nós de referência, maior será o número de intersecções aumentando a precisão do cálculo do posicionamento.

4.5 Configuração da Rede XBee

Neste ponto descreve-se a configuração dos módulos XBee que implementam a rede *ZigBee*.

Para todos os módulos do sistema comunicarem com o computador necessitam de integrar um módulo XBee configurado adequadamente. No sistema identificam-se três tipos de nós: o coordenador da rede que serve de *gateway*, ligado directamente ao computador via uma porta série; os routers que são usados nos módulos de monitorização de atletas e nas torres de encaminhamento (também com a função de balizas de localização); e os *end devices* (ou nós sensores) que são aplicados aos módulos de monitorização ambiental. A configuração dos nós é feita através do programa X-CTU da DIGI. Utilizou-se módulos XBee série 2 em conjunto com o *firmware* XBee ZB (*ZIGBEE*), por este ser o *firmware* mais recente que implementa o protocolo *ZigBee*. Para configurar cada nó é necessário seleccionar os parâmetros adequados no *firmware* e actualizar o módulo XBee respectivo. Na tabela, 4.1, lista-se os parâmetros que são necessários configurar conforme os tipos de nós utilizados na rede.

TABELA 4.1 – CONFIGURAÇÃO DE PARÂMETROS DA REDE XBEE.

Parâmetros	Coordenador	Router	Sensor	Observação
PAN ID	125	125	125	Identificação do grupo de trabalho.
Power level	*Highest*	*Highest*	*Highest*	Potência de saída de 1 dBm.
Power mode	*Boost*	*Boost*	*Boost*	Aumento da potência de saída em 2 dB e da sensibilidade em 1 dB.
Node join time	0xFF	0xFF	-	0xFF – permite os nós registarem-se na rede a qualquer momento.
Pin PWM	-	RSSI	-	Activa a saída, do nível do sinal recebido, em forma de sinal PWM.

Na figura 4.4, apresenta-se a janela do programa de configuração e programação dos módulos XBee.

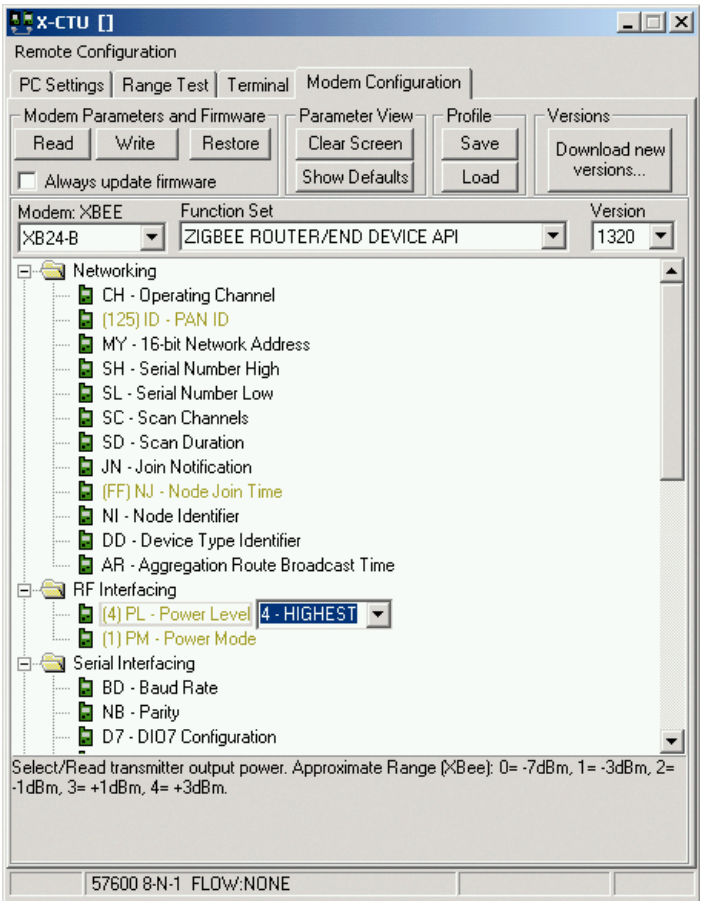


Figura 4.4 – X-CTU, programa de configuração do XBee.

Quanto às opções de *hardware* disponíveis, define-se as opções de antenas utilizadas e a escolha entre XBee OEM e PRO. Para as antenas, optou-se por utilizar no coordenador, nas torres de referência e no módulo de monitorização ambiental, módulos XBee com conexão SMA em conjunto com antenas monopolo. Assim, com as antenas aumentou-se o nível do sinal e excede-se assim as condições que garantiam pelo menos 120 metros de alcance. Para os módulos de atletas, utilizam-se módulos com a *chip* antena, sendo esta a antena disponível de menor perfil, contribuindo assim, para as dimensões reduzidas do módulo monitor de atleta. Entre a versão XBee, optou-se pela OEM, considerando o alcance entre nós de 120 metros capaz de abranger a monitorização de atletas num recinto fechado. No caso de áreas maiores, é sempre possível trocar pela versão PRO, incrementando a distância para pelo menos 300 metros em linha de vista [23].

Definida a versão de *firmware* do XBee utilizada, define-se o formato das mensagens que circulam na rede *ZigBee* que seguem a estrutura API descrita na figura 3.30. Os vários nós, para enviar os seus dados para o computador, têm de compreender os dados num pacote de transmissão de dados. Estes dados seguem no campo *RF Data*, que no caso da API do XBee permite até o máximo de 72 bytes de dados.

Não havendo nenhum campo para distinguir as diferentes aplicações dos nós da rede, definiu-se que o primeiro byte indica o tipo de módulo do sistema (do atleta, torre ou monitorização ambiental) e ainda estipula os dados que vêm na mensagem, para que assim, a aplicação de recepção possa interpretar os dados recebidos. Na tabela 4.2 listam-se as estruturas das diferentes mensagens que circulam no sistema.

TABELA 4.2 – ESTRUTURA DOS DADOS DAS MENSAGENS DE REDE DO SISTEMA

Módulo	Tipo	Dados (bytes)	Destinatário (endereço)	Descrição
Atleta	0xA0	GPS – Lat,N/S,Long,E/O (10) + Ritmo Cardíaco (1) + Endereço de balizas e RSSI (9x4)	Coordenador (0x00000000)	Envio da posição do atleta, ritmo cardíaco, e o endereço e respectivo nível de sinal de até quatro referências.
Atleta	0xA1	Aceleração (70)	,,	Envio da aceleração vertical ao longo de um segundo. 100 amostras de 16 bits, ou seja, 200 bytes.
Atleta	0xA2	Aceleração (70)	,,	
Atleta	0xA3	Aceleração (60)	,,	
Nó referência	0xB0	GPS (10) + Endereço dos nós e RSSI (9x4)	,,	Envio da posição da referência, e o endereço e respectivo nível de sinal de até quatro balizas.
Nó referência	0xB1	-	*Broadcast* (0x0000FFFF)	Mensagem de *broadcast* para referência.
Ambiental	0xC0	Temperatura (2) + Humidade (2) + Luz (2) + O_2 (2) + CO (2) + CO_2 (2) + Bateria (2)	Coordenador (0x00000000)	Envio das medidas dos parâmetros ambientais.

A aplicação no computador, organiza os dados segundo o endereço único do remetente, que vem explícito no pacote de recepção de dados entregue ao computador, pelo coordenador e *gateway* da rede.

4.6 Métodos e Ferramentas de Desenvolvimento

Para desenvolvimento deste projecto foi necessário recorrer a uma panóplia de ferramentas de *software* e *hardware*. De seguida resume-se os processos de desenvolvimento utilizados e as ferramentas intervenientes.

Inicialmente desenhou-se circuitos de teste e através da simulação verificou-se se o funcionamento correspondia ao projectado. Para isso utilizou-se o *software* Multisim [82] da National Instruments.

Para obter uma versão funcional dos protótipos verificados na simulação desenhou-se o circuito impresso com recurso ao *software* Eagle [83].

Já com o desenho terminado procedeu-se ao fabrico do mesmo com recurso aos equipamentos e materiais disponíveis nos laboratórios da Universidade da Madeira. O processo de fabrico segue a diagrama apresentado na figura 4.5.

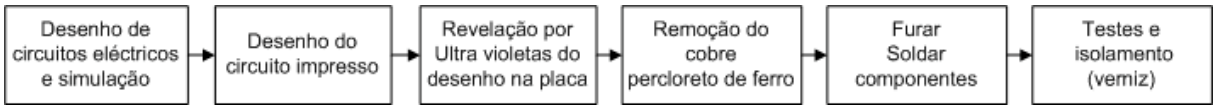


Figura 4.5 – Procesos de fabrico do protótipo *hardware*.

Posteriormente realizou-se a programação do controlo a embeber nos microcontroladores. Este controlo foi desenvolvido sob as ferramentas fornecidas pelo fabricante do microcontrolador, ATMEL. Utilizou-se o editor AVR Studio [84] e o compilador Win AVR [85]. Para programar fisicamente os microcontroladores foi utilizado a ferramenta FLIP [86] para o microcontrolador AT90USB1287 e no caso da versão ATMega foi necessário recorrer ao programador ISP500 [87] da Olimex.

Para configurar a rede *ZigBee* nos rádios da XBee utilizou-se o programa X-CTU [88] fornecido pelo fabricante DIGI. O X-CTU oferece um interface de texto para a porta série, meio pelo qual a rede se liga ao computador, que nos permitiu observar os primeiros testes de dados enviados pela rede. Com uma ideia dos dados a serem registados criou-se uma base de dados em MySQL. Utilizando a linguagem Java, através do uso do *Integrated Development Environment* (IDE) Netbeans [89] concebeu-se um pequeno programa para receber e armazenar os dados na base de dados.

Os primeiros resultados foram avaliados através de gráficos produzidos com o *software* Matlab [72]. Para isso aplicou-se as respectivas expressões matemáticas convertendo os valores recebidos em informação útil. O Matlab permitiu uma análise rápida dos resultados sem preocupações sobre o desenho gráfico.

À aplicação Java adicionou-se interface gráfica e assim substituiu-se o trabalho realizado em Matlab com vista a disponibilizar uma aplicação que pudesse correr em qualquer máquina sem custos adicionais.

Para uma solução que fosse acessível remotamente, desenvolveu-se uma aplicação para a Internet. Com o pacote XAMPP [90] instalou-se um servidor de Internet no computador local como plataforma de desenvolvimento. Este pacote inclui o servidor Apache, motor PHP e sistema de base de dados MySQL. Utilizou-se o editor PHP Designer [91], para programar a aplicação na linguagem PHP.

Dos recursos utilizados, sublinha-se as livrarias externas utilizadas nos diferntes programas desenvolvidos:

- Livraria de controlo do display HD44780 [92] utilizado no *firmware* para o módulo de monitorização ambiental;
- Livraria de desenho de gráficos JPGRAPH [93] para a aplicação de Internet em PHP;
- Livraria de acesso á base de dados MySQL para PHP MYSQL4 [94] da phpBB;
- Livraria de acesso à base de dados MySQL para Java, MySQL connector [95] da SUN;
- Livraria de interface à porta série para Java, RXTXCOM [96];
- Livraria de acesso ao interface Bluetooth do telemóvel para Java ME, Bluecove [97].

4.7 Módulo de Monitorização Ambiental

A monitorização ambiental é realizada por um módulo dedicado para o efeito.

4.7.1 Descrição Funcional

Baseado no estudo e desenvolvimento do primeiro protótipo descrito no ponto 3.3.2, definiu-se o diagrama funcional do protótipo final apresentado na figura 4.6. O módulo realiza tarefas periódicas e tarefas a pedido. As tarefas periódicas consistem em: amostrar os sensores, compilar os valores numa mensagem *ZigBee* e enviá-la para a *gateway* do sistema, a cada minuto. As tarefas a pedido incluem a visualização dos parâmetros medidos num ecrã, directamente no módulo, e na resposta a um pedido por *Bluetooth*, com os últimos valores monitorizados e com a indicação da localização do módulo, incluindo o nome da rua ou local.

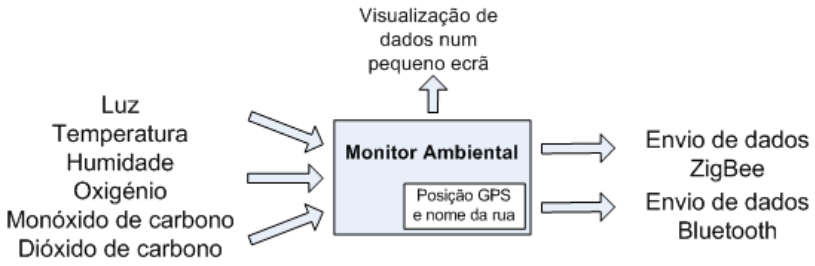


Figura 4.6 – Diagrama funcional do módulo de monitorização ambiental.

4.7.2 Componentes Utilizados

Iníciou-se o desenvolvimento pela pesquisa de mercado sobre sensores de CO_2, CO e O_2, dando atenção especial às dimensões, baixo consumo e custo.

Adquiriu-se dois sensores de dióxido de carbono (CO_2): o módulo TGS4160 produzido pela Figaro e o C20 da Gas Sensing Solutions. O TGS4160 é um sensor que mede a concentração de CO_2, baseado num electrólito de estado sólido [98]. O C20 por sua vez, detecta a concentração de CO_2 medindo a absorção de luz infravermelha [99]. Esta tecnologia embora de maior custo (160 contra 120 Euros), oferece inúmeras vantagens em relação à do TGS4160: alta velocidade (arranque em 2 s contra 2 h da primeira opção), ampla gama de medida desde 0% (mínimo de 200ppm) a 65%, imunidade à luz solar, precisão e baixo consumo (< 100 mW contra os 300 mW do TGS4160). O tempo de arranque rápido do C20 permite que seja ligado apenas no momento da medida, contribuindo assim para uma maior eficiência energética. Decidiu-se assim utilizar o sensor C20. Este é um sensor comercializado num módulo completo, calibrado de fábrica, que inclui o condicionamento do sinal medido (conversão e linearização), fornecendo o valor final através de uma porta UART. Na figura 4.7 ilustra-se a constituição do sensor de C20 [99].

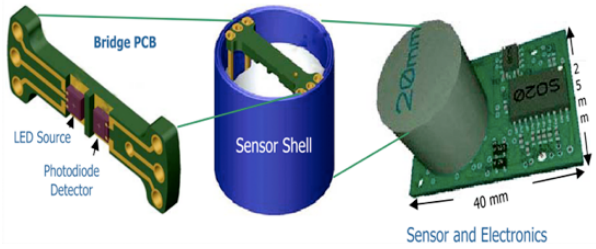


Figura 4.7 – Sensor de dióxido de carbono, C20.

O módulo é autónomo e começa a enviar dados a cada meio segundo, 2 segundos após ser ligado. Este módulo pode disponibilizar uma medida em tempo real ou uma medida resultante de uma média, configurando para isso, um parâmetro entre 1 a 128. Este parâmetro permite diminuir o ruído da medida mas reduz o tempo de resposta. O sensor C20 vem calibrado de fábrica em toda a sua escala, mas oferece vários meios para calibrar tanto o desvio da escala como o declive da recta de conversão, sendo aconselhado para isso, o uso de um gàs de calibração.

O sensor de monóxido de carbono (CO) utilizado é o TGS5042 da Figaro. É um sensor electroquímico que apresenta uma saída em corrente que varia linearmente com a concentração de CO no ar [100]. É indicado para dispositivos que trabalham a baterias, desde aplicações residenciais a industriais. Apresenta uma gama de medida desde os 0 até 10 000 ppm, alta repetibilidade e sensibilidade de CO, mas necessita de calibração. O fabricante fornece os dados de calibração que incluem: inclinação da recta de conversão e compensação

em temperatura. Na figura 4.8 apresenta-se o sensor, onde pode-se observar a inscrição dos dados de calibração na lateral do encapsulamento [100].

Figura 4.8 – Sensor de monóxido de carbono, TGS5042.

Para monitorizar o CO é necessário converter a saída em corrente do sensor numa tensão, e então pode ser medida pelo microcontrolador. Isto pode ser feito de duas maneiras, através de sensor de corrente ou de resistência de carga. O fabricante disponibiliza o desenho de modelos de circuitos, desde um simples até um circuito compensado em temperatura com prevenção de ruído eléctrico. A utilização de um microcontrolador traz vantagens, nomeadamente o cálculo de conversão e a compensação de temperatura. Assim escolheu-se o circuito básico, sensor de corrente, apresentado na figura 4.9 [100].

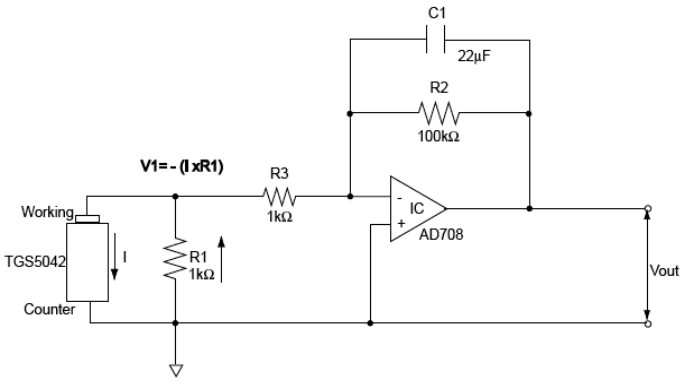


Figura 4.9 – Circuito de conversão e amplificação para o sensor de CO TGS5042.

Este circuito converte a corrente gerada pelo sensor segundo a seguinte equação:

$$Vs = Isensor \times R1 \times (R2/R3) \tag{4.5}$$

Calibração: na lateral do sensor vem impresso o valor de corrente por cada ppm, que neste caso é 1378, o que corresponde a 1,378 nA/ppm. Utilizando este valor o fabricante garante ±15% de precisão e que é possível atingir os ±5% de precisão realizando a calibração através de concentrações conhecidas. À conversão é necessário multiplicar o factor de correcção de temperatura, apresentando no Anexo C sob a forma de curva. Assim, a concentração de CO é dada por

$$CO = \frac{Vs}{R1 \times (R2/R3) \times 1{,}378 \times 10^{-9}} \times T_{coef}(ppm) \tag{4.6}$$

Apresentando, o sensor em conjunto com o amplificador operacional, uma tensão de desvio é necessário realizar a compensação de *offset* (ou ajuste a zero). Isto pode ser feito medindo a tensão de saída num ambiente de ar limpo (0 ppm de CO), e subtraindo este valor à

saída do amplificador. Para maior precisão a temperatura deve de estar em torno dos 20±10ºC. Por fim, a colocação do sensor na placa de circuito impresso, deve ter em conta a proximidade ao contacto com o ar (junto da abertura da caixa) para, assim, melhorar o tempo de reposta do sensor e a proximidade do sensor de temperatura para a compensação em temperatura.

Para monitorização do oxigénio adquiriu-se o KE-25, um sensor do tipo célula galvânica produzido pela Figaro, que se destaca de outros sensores do mercado por apresentar uma saída linear e não ser influenciado por gases como: CO_2, CO, H_2S, NOx, H_2 [101]. O sensor apresenta uma gama de medida desde os 0 aos 100% de concentração de O_2, uma precisão de ±1% e uma duração de cinco anos. Na figura 4.10 apresenta-se o sensor de oxigénio utilizado [101].

Figura 4.10 – Sensor de oxigénio, KE-25.

O sensor não requer qualquer fonte de alimentação externa, mas devido à escala reduzida de tensão à saída do sensor, utilizou-se um circuito baseado num AMPOP para amplificar a saída do sensor. O ganho foi definido em 40, no sentido de usar a máxima gama de medida do ADC do microcontrolador que é de 2,56 V, tendo em conta que a tensão máxima aos terminais do sensor é de 65 mV. Considerando a recta de conversão da percentagem de oxigénio em relação à tensão de saída, apresentada no anexo C, a concentração de O2 é dada por

$$O_2 = \frac{Vs}{40} \times 1{,}78(\%) \tag{4.7}$$

A saída do sensor é linear, mas decresce ao longo do tempo, havendo necessidade de calibração periódica. O sensor KE-25, também, apresenta uma tensão de *offset* à qual é necessário realizar a respectiva compensação. Isto pode ser feito medindo a tensão de saída na presença de uma concentração conhecida, e subtrair este valor à tensão registada. Na montagem é necessário ter atenção, que devido ao sensor ser composto por um líquido, deve ser montado verticalmente segundo a seta impressa na lateral do mesmo.

Para a temperatura e humidade foi utilizado o SHT15, que é um sensor duplo num só circuito integrado, produzido pela Sensirion [102]. Como foi referido no ponto 3.3.1, utilizou-se este sensor por ser largamente utilizado nas redes de sensores sem fios, especialmente devido ao seu baixo consumo e pequeno tamanho. Este sensor já vem calibrado de fábrica e produz uma saída digital. O sensor é produzido usando um processo CMOS e está acoplado com um conversor A/D de 14-bit. Este sensor ainda apresenta excelente qualidade de sinal, como demonstrado pelas suas curvas de precisão apresentadas no Anexo C, e boa rejeição de ruído [102], permitindo que possa ser utilizado em diversas aplicações de monitorização ambiental. Na figura 4.11 apresenta-se o encapsulamento do sensor [102].

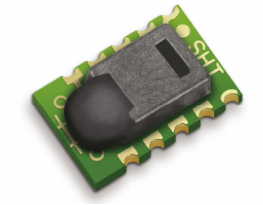


Figura 4.11 – Sensor de humidade e temperatura, SHT15.

O interface é semelhante ao I2C mas não é padrão. Então é necessário embeber o protocolo de comunicação no programa do microcontrolador. A saída do sensor necessita de conversão e compensação consoante o modelo do sensor. Assim para o modelo utilizado o valor de temperatura é dado por

$$Temperatura = Amostra \times 0{,}01 - 40 \ (^{\circ}C) \tag{4.8}$$

Coeficientes necessários para calcular a humidade [102]:

$C1 = -4{,}0$

$C2 = 0{,}0405$

$C3 = -0{,}0000028$

$T1 = 0{,}01$

$T2 = 0{,}00008$

$$rh\_lin = C3 \times Amostra^2 + C2 \times Amostra + C1 \tag{4.9}$$

A humidade é obtida pela seguinte equação:

$$Humidade = (Temperatura - 25) \times (T1 + T2 \times Amostra) + rh\_lin \ (\%) \tag{4.10}$$

Para medição da luz ambiente, foram escolhidos, pelas mesmas razões que o sensor anterior, dois fotodíodos da Hamamatsu. O S1087 e S1087-01, para medir a radiação fotossintética activa e a radiação solar total respectivamente [103]. Este fotodíodo é utilizado no Tmote Sky em paralelo com uma resistência de 100 kΩ, mas é demonstrado que nesta configuração obtém-se uma resposta não linear. É sugerido por [104] a utilização de uma amplificador de transimpedância, apresentado na figura 4.12 [104].

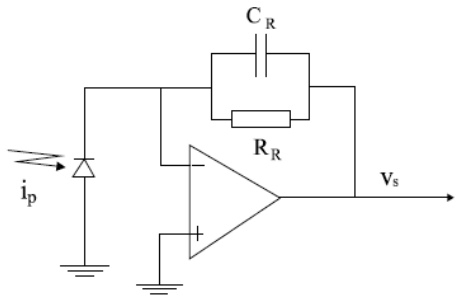


Figura 4.12 – Amplificador de transimpedância para monitorização da corrente do fotodíodo.

Nesta configuração, a relação de tensão de saída do circuito com a corrente do fotodíodo, é dada pela seguinte equação:

$$Vs = R_r * I_p \tag{4.11}$$

Tendo em conta a recta de conversão, apresentada no Anexo C, a medida em Lux é dada por

$$Luz = \frac{Vs}{R_r} \times 10^9 \tag{4.12}$$

Após escolha dos sensores, pesquisou-se por uma forma de adicionar interface *Bluetooth* ao módulo de monitorização ambiental. Para isso adquiriu-se três módulos de *Bluetooth*: o BISMS02BI-01 da Ezurio [105], o WT11 e WT12 da BlueGiga [106]. Todos os módulos apresentam interface UART e permitem funcionar com o perfil SPP (*Serial Port Profile*), que permite de uma forma simples criar uma extensão baseada no protocolo RS232. O módulo WT12 foi excluído por apresentar o menor alcance de entre os módulos. Os outros dois módulos apresentam um consumo muito similar cerca de 3 mA para o estado de escuta e de 30 mA para o momento de comunicação. O modelo da WT11 BlueGiga apresentou um funcionamento mais transparente em relação ao do Ezurio, já que este necessita de um conjunto de comandos AT para funcionar, e as suas dimensões inferiores, ditaram o uso do WT11 para a realização do interface *Bluetooth*. Na figura 4.13 apresenta-se o módulo *Bluetooth* utilizado [106].

Figura 4.13 – Módulo *Bluetooth* WT11.

Para completar o conjunto de componentes necessários para o módulo de monitorização ambiental, e tendo em conta os resultados obtidos no protótipo do ponto 3.3.1, pesquisou-se por um microcontrolador de menor consumo e um conversor DC-DC mais eficiente.

O sistema inclui diversos componentes que precisam de ser interligados e controlados, para isso utilizou-se um microcontrolador. Existem um diverso número de opções no mercado, nomeadamente dos fabricantes: Atmel, Texas Instruments e Microchip. A experiência já adquirida com microcontroladores da família AVR da Atmel, foi a razão de maior peso para a escolha de um modelo deste fabricante.

Os diversos componentes escolhidos para o sistema e a aplicação pretendida impõem determinados requisitos para o microcontrolador, estes são:

- 2 UART's assíncronas – para comunicar assincronamente com o XBee e com o módulo *Bluetooth*;
- 5 ADC's - para converter o sinal dois sensores de luz, oxigénio, monóxido de carbono e ainda monitorização do nível da bateria;
- Duas dezenas de pinos I/O digitais - para comunicação com o sensor de humidade e temperatura, o SHT15, comunicar com o sensor de CO_2, enviar dados para o display, led de presença e controlo da alimentação.
- *Real Time Clock* (RTC) – para a programação de eventos periódicos como a amostragem dos sensores e envio dos dados para a rede *ZigBee*;
- Interrupções para a actuação com eventos assíncronos, como a recepção de dados dos módulos XBee e *Bluetooth* ou comandos de um botão em qualquer altura;

- Baixo consumo energético.

A escolha recaio no modelo ATmega324, doravante denominado por ATmega324 ou simplesmente microcontrolador. O ATmega324 é um microcontrolador de 8 bit de tecnologia CMOS e arquitectura RISC. Apesar de ser RISC, possui um grande número de instruções (131), o que permite melhor optimização do código de alto nível em linguagem C. Apresenta 32KB de memória Flash para programa e dados, 3 temporizadores/contadores, 2 de 8 bit e 1 de 16 bit, RTC com oscilador separado, 6 canais PWM, 8 canais ADC multiplexados para um ADC de 10 bit, duas UART programáveis, interrupção e *Wake-up* com alteração do nível nos pinos digitais, tensão de operação desde os 2,7 V aos 5,5 V, velocidade de 0 a 8MHz para uma tensão de 2,7-5,5 V, oscilador integrado, consumo de 8mA no modo activo a 5V, e vários modos de repouso que permitem baixar o consumo para valores inferior ao 1mA, cada porta consegue fornecer 40 mA por pino e um total de 200 mA, e uma gama de temperatura de funcionamento desde os -40 ºC até +125 ºC permitindo a sua utilização à temperatura ambiente [107].

Os diversos componentes escolhidos operam a dois níveis de tensão, 3,3 V e 5 V. Definiu-se como fonte de energia um par de baterias do tipo AA recarregáveis, devido à sua capacidade e dimensões. Como as baterias recarregáveis apresentam uma variação do nível de tensão durante a descarga como mostra a figura 4.14 [108], um par de baterias em série apresentarão uma carga útil dos 2,8 V aos 2,2 V, sendo assim é necessário converter a tensão para os níveis desejados e estabilizá-la. Isto foi realizado com recurso a dois conversores DC-DC. Um para os 3,3 V e outro para os 5 V.

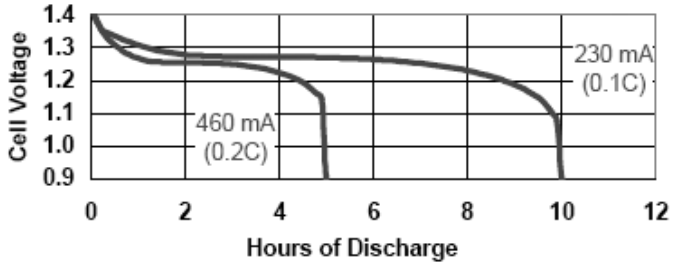


Figura 4.14 – Característica de descarga da bateria Energizer 2500.

A escolha recaiu no *step-up* MAX1675 da Maxim que embora em relação ao NCP1400, utilizado no primeiro protótipo, apresenta uma menor gama de tensão de entrada (de 1,1 a 3.3 V contra 0,7 a 3,3 V), apresenta maior eficiência em toda a gama de corrente de carga, especialmente nos valores mais baixos para 1 mA de carga o MAX1675 apresenta 92% de eficiência enquanto que o NCP1400 não ultrapassa os 70%.

O fabricante do conversor DC-DC indica vários circuitos típicos de aplicação. Apresenta-se na figura 4.15 o esquemático eléctrico da configuração utilizada para a conversão dos 3,3 V [109]. Para 5 V é necessário colocar o pino FB (*FeedBack*) ligado à massa.

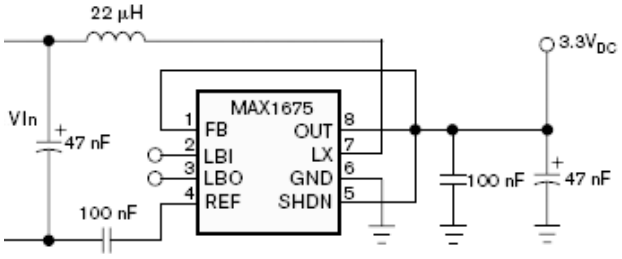


Figura 4.15 – Circuito do *Step-Up* MAX1675 para 3.3V.

Os componentes periféricos são poucos, mas a bobina e o condensador de saída de 47uF afectam directamente a corrente máxima de saída, *ripple* e eficiência do circuito. Utilizou-se modelos indicados pelo fabricante, que apresentam principalmente baixo ESR (*Equivalente Series Resistance*), necessários para obter a eficiência pretendida.

4.7.3 Arquitectura

Após escolhidos os vários componentes, definiu-se a arquitectura do protótipo final do nó de monitorização ambiental, análogo aos módulos de monitorização expostos. Cada nó é então constituído por quatro partes:

- Controlo do nó (microcontrolador);
- Sensores (Humidade, Temperatura, Luz, CO, CO_2 e O_2);
- Periféricos de interface (*ZigBee*, *Bluetooth* e *Display*);
- Alimentação (bateria e conversores DC-DC).

Na figura 4.16 apresenta-se a arquitectura final do nó de monitorização ambiental desenvolvido.

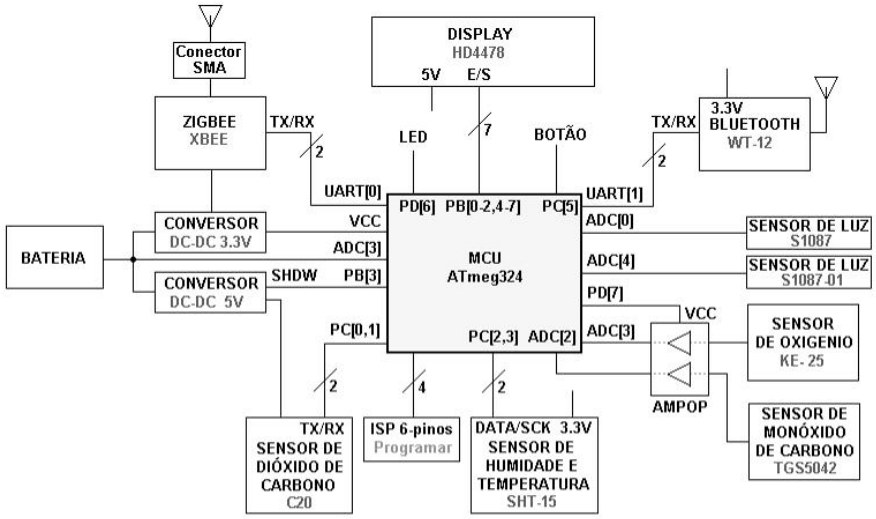


Figura 4.16 – Arquitectura do nó sensor ambiental.

Destaca-se a opção tomada, do nó utilizar dois conversores DC-DC, para alimentar os diferentes componentes que trabalham a tensões distintas. Então, entre a bateria e o conversor para 5V, foi intercalado um transístor de tecnologia MOSFET para controlar a activação do conversor, para assim, ligar os componentes que trabalham a 5 V apenas quando necessários contribuindo para a poupança de energia, nomeadamente o sensor de CO_2.

4.7.4 Protótipo Implementado

Tendo em conta os detalhes, indicados pelos fabricantes, da aplicação dos diversos componentes num circuito impresso, construiu-se o protótipo final em circuito impresso. O resultado final é apresentado na figura 4.17. No Anexo D apresenta-se o esquemático eléctrico e o desenho do circuito.

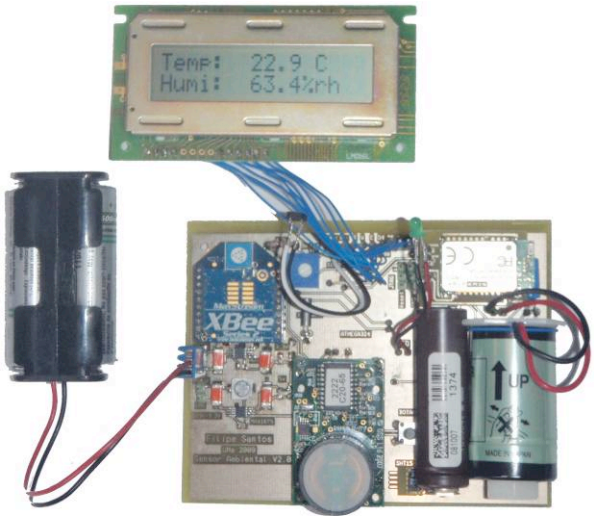


Figura 4.17 – Módulo de monitorização ambiental, versão 2.

Foi adicionado um pequeno conector ligado em paralelo com as baterias, para permitir a recarga externa das baterias sem ter de retirá-las. Para finalizar foi adquirida uma caixa estanque à medida, apresentada na figura 4.18, e feito os respectivos orifícios para a colocação do sensor de luz, *display*, botão, led, conector de antena *ZigBee* e conector para alimentação externa. No fundo da caixa foram feitas diversas aberturas para a circulação de ar no interior da caixa, por forma a expor os sensores ao ar ambiente.

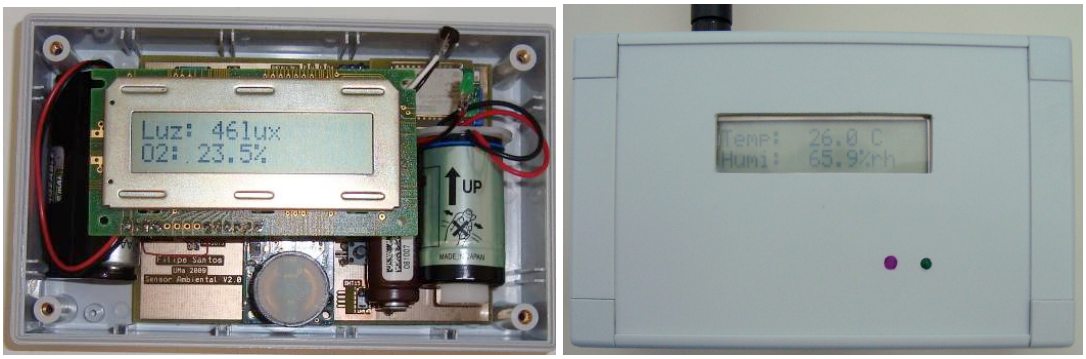


Figura 4.18 – Protótipo final do monitor ambiental com caixa.

Deste trabalho compilou-se um artigo apresentado na conferencia AMIES 2009, apresentado no Anexo J, em que é descrito o desenvolvimento do protótipo no âmbito de monitorização de ambientes urbanos.

4.7.5 Controlo Embebido

O programa embebido no microcontrolador do nó, desenvolvido em C, é responsável pelo funcionamento autónomo do mesmo. A programação resume-se à configuração de uma

interrupção temporal que controla uma variável de tempo. Em função do valor desta variável são executadas as tarefas do módulo.

O programa tem duas secções, a responsável pelas tarefas periódicas e a responsável pelas tarefas assíncronas. Pela amostragem dos vários sensores e de enviar os dados, e por responder aos telemóveis e mostrar os dados no *display*. Ao ligar o módulo, dá-se inicialização de variáveis e do *real time clock* (RTC). De seguida o programa coloca o nó em modo adormecido: activa o modo adormecido do XBee, desliga o conversor DC-DC de 5V e desliga o amplificador, por forma a diminuir o consumo durante o tempo que não realiza qualquer tarefa. O período escolhido de amostragem é de 1 minuto, pelo qual a cada minuto é repetido o processo de amostragem e recolha de dados, compilação da mensagem e envio da mesma para o coordenador da rede através do XBee. A segunda secção é encarregue por responder, assincronamente, a pedidos chegados através do *Bluetooth*. Responde com a localização do módulo, previamente gravada na memória do microcontrolador, seguida dos dados mais recentes das variáveis monitorizadas, e, ainda, a mostrar os dados no *display* quando o botão presente no módulo é actuado. Neste caso, as fórmulas de conversão da amostra para o valor final são aplicadas no microcontrolador, e assim são apresentados os valores nas unidades adequadas. Na figura 4.19 apresenta-se o diagrama de fluxo de dados do programa de controlo.

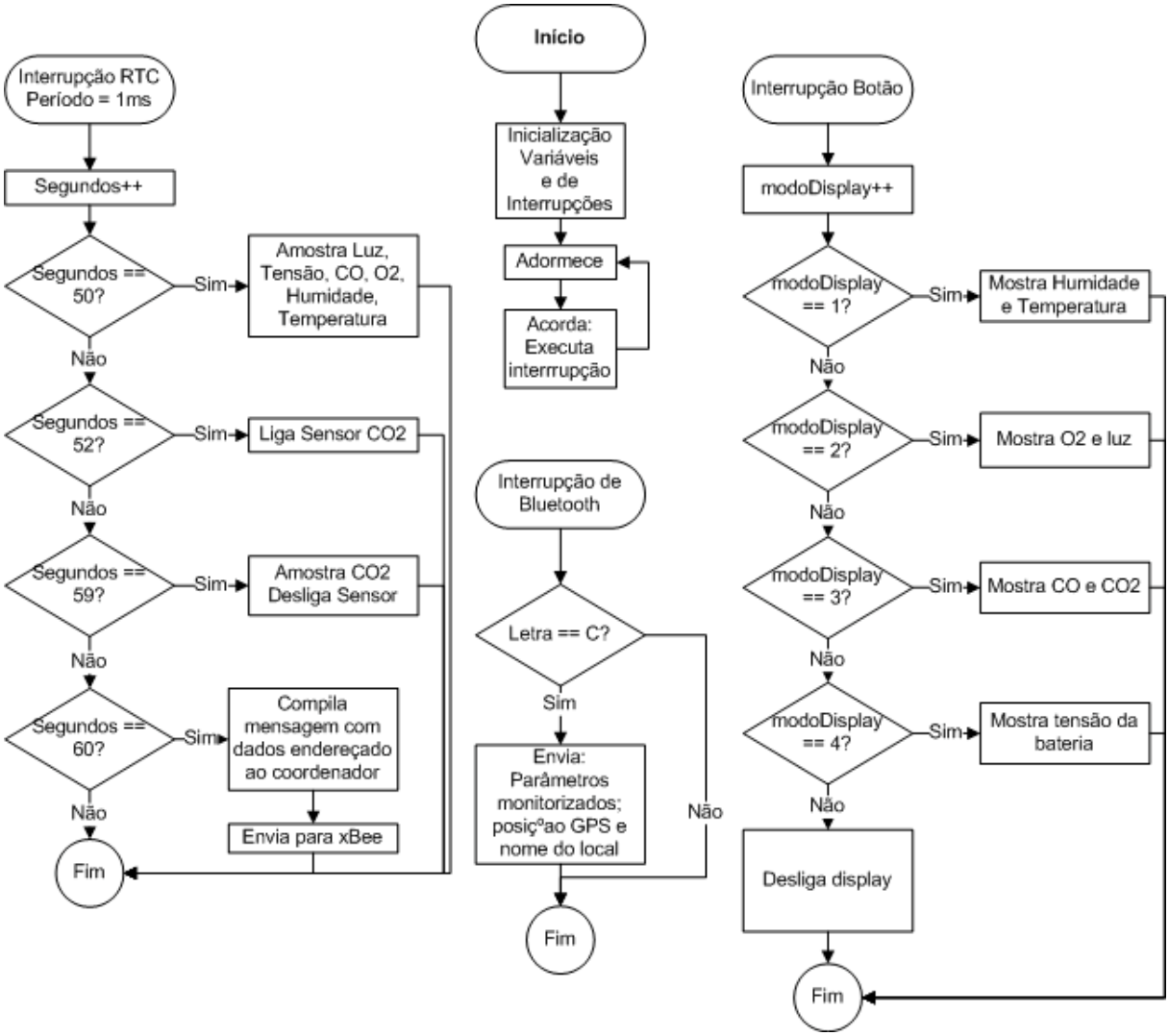


Figura 4.19 – Diagrama de fluxo de dados do controlo do módulo de monitorização ambiental.

No Anexo H apresenta-se o código fonte desenvolvido com os comentários respectivos que contribuem para uma melhor compreensão do programa.

4.8 Módulo de Monitorização de Atleta

A monitorização de atletas é feita através da utilização de um equipamento de monitorização dedicado para o efeito. Este equipamento é constituído por dois módulos: a banda torácica de monitorização do ritmo cardíaco e o módulo de cintura. Na figura 4.20 apresenta-se a localização dos componentes de monitorização no atleta.

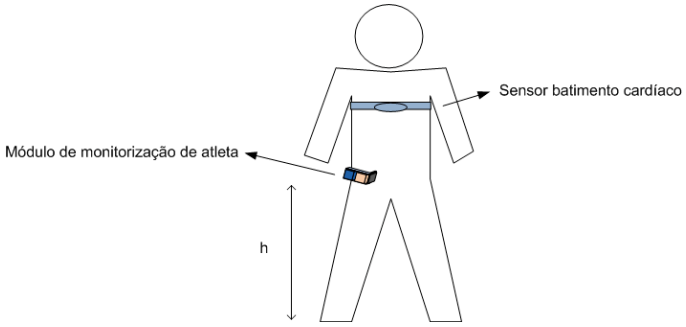


Figura 4.20 – Componentes de monitorização no atleta.

4.8.1 Descrição Funcional

O funcionamento do módulo de monitorização do atleta, compreende: medição do período entre batimentos cardíacos, registo das coordenadas GPS, medição de 100 amostras da aceleração vertical, e registo do nível de sinal das mensagens de *broadcast* das balizas de referência a cada segundo. Findo o qual, envia os dados para a *gateway* do sistema. Na figura 4.21 apresenta-se o diagrama funcional do módulo de monitorização do atleta.

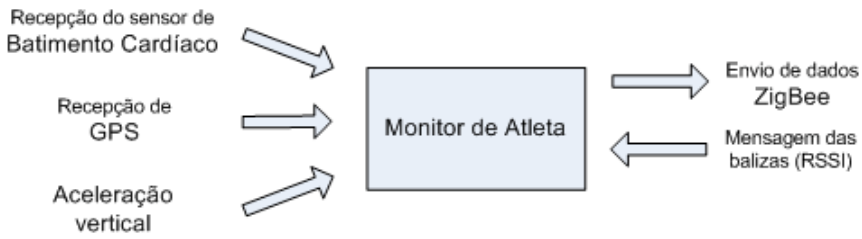


Figura 4.21 – Diagrama funcional do módulo de monitorização de atleta.

4.8.2 Componentes Utilizados

Realizou-se uma pesquisa de mercado da qual escolheu-se e adquiriu-se os sensores: módulos de GPS, sensor de batimento cardíaco e acelerómetros. De seguida descreve-se os componentes utilizados no módulo de monitorização de atletas e razões da sua escolha.

Os módulos de recepção de GPS no mercado apresentam uma precisão em torno dos cinco metros. Adquiriu-se dois módulos de recepção de GPS, o EM-406 da GlobalSat [110] e LS20031 da Locosys [38], com 5 metros e 2,6 metros de precisão respectivamente. Ambos apresentam um formato de *hardware* e de funcionamento padronizado muito similar. Ambos são módulos completos com antena incorporada, que oferecem uma interface UART, e utilizam o formato de mensagens NMEA, mas o modelo da LOCOSYS destaca-se em todos os parâmetros. Este apresenta actualizações de posicionamento a 5 Hz, consumo médio de 41 mA a 3.3 V, tempo de arranque médio de 36 segundos, e precisão de posição de 3 m e de 2.5 m aquando dados de correcção disponíveis (DGPS). Este módulo apresenta elevada

sensibilidade, mesmo em ambientes com elevada folhagem. Na figura 4.22 apresenta-se o módulo GPS [38].

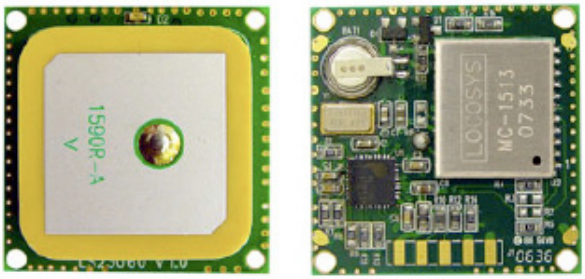


Figura 4.22 – Módulo GPS LS20031.

A mensagem NMEA contém, além dos dados de posicionamento, diversas informações sobre o sistema de posicionamento, como número de satélites utilizados, hora actual entre outros. Desta mensagem são extraídos para a aplicação, a Latitude, Longitude, e o indicador da validade da posição.

A escolha do sensor de batimento cardíaco, teve por base o trabalho apresentado no relatório intermédio do projecto de mestrado de Roberto Fernandes [111]. Foi escolhida a banda torácica da Polar [112] em conjunto com o receptor RMCM-01 [113]. Um receptor simples que detecta o impulso electromagnético de 5 KHz gerado pelas bandas torácicas de monitorização do ritmo cardíaco. O receptor gera um impulso de 1ms, por cada batimento cardíaco detectado. Na figura 4.23 apresenta-se a bandas da Polar e o receptor RMCM-01.

Figura 4.23 – Monitor de ritmo cardíaco da Polar.

O sinal do receptor é conectado a uma interrupção do microcontrolodador do módulo, que com recurso a um contador, realiza a cronometragem do tempo entre batimentos para calcular o ritmo cardíaco. A distância máxima entre a banda e o receptor é de 80cm, e para a melhor recepção estes deverão estar orientados como mostra a figura 4.24 [113]. O curto alcance, permite que as interferências entre atletas sejam quase nulas.

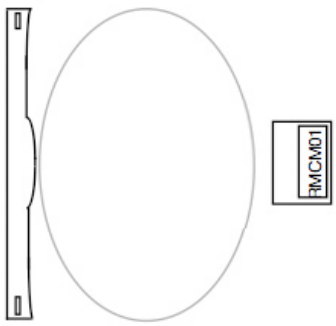


Figura 4.24 – Disposição adequada do receptor de batimento cardíaco.

Para o acelerómetro, adquiriu-se duas opções: o ADXL330 da Analog Devices [114] e o modelo LIS3LV02DQ do fabricante STMicroelectronics [115]. Ambos apresentam,

características comuns adequadas à monitorização de atletas, três eixos ortogonais de medida dando liberdade de colocação, pequenas dimensões, baixo consumo, sensibilidade na ordem do 1 mg, gama de medida de ±3,6 g e ±6 g respectivamente. Divergem essencialmente pelo ADXL330 apresentar uma saída analógica, implicando a necessidade de condicionamento de sinal, e por outro lado o LIS3LV02DQ apresentar uma saída digital, com o protocolo SPI. A saída digital já oferece uma saída filtrada e digitalizada facilitando o desenho do hardware e evitando complicações com ruído do andar de condicionamento de sinal, razões que ditaram a escolha deste acelerómetro para a monitorização de atletas. Na figura 4.25 apresenta-se o acelerómetro utilizado [99].

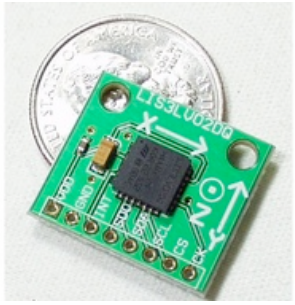


Figura 4.25 – Acelerómetro LIS3LV02DQ.

O restante hardware segue as opções tomadas no módulo de monitorização ambiental. Optou-se pelo microcontrolador ATmega324, novamente pela disponibilidade de duas portas UART, de forma a comunicar assincronamente com o módulo XBee e o GPS. Para a alimentação, uma vez que todos os componentes trabalham aos 3,3 v, utilizou-se também um *step-up* MAX1675 para a partir de um par de baterias alimentar o módulo.

4.8.3 Arquitectura

O módulo de monitorização do atleta é constituído por quatro partes:

- Controlo do nó (realizado por um microcontrolador);
- Sensores (Ritmo cardíaco, Acelerómetro, GPS);
- Periféricos de interface (*ZigBee*, LED e Botão);
- Alimentação (bateria e conversor DC-DC).

Na figura 4.26 apresenta-se a arquitectura do nó de monitorização de atletas desenvolvido.

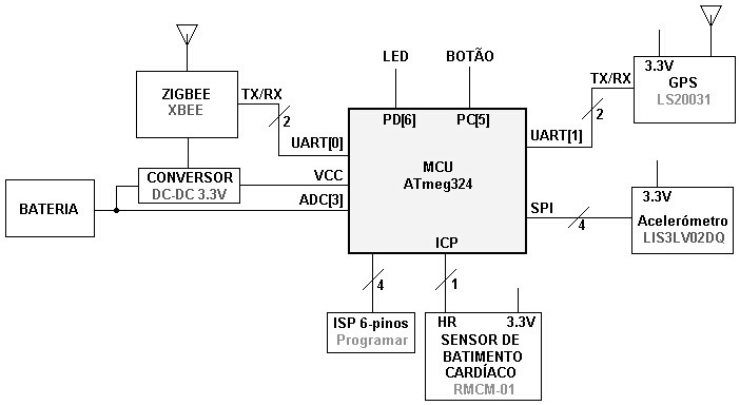


Figura 4.26 – Arquitectura do nó de monitorização de atletas.

Como já foi referido, o nó de monitorização de atletas é um dispositivo de cintura. Isto trás condicionamentos, nomeadamente, as dimensões e o peso do dispositivo deverão ser relativamente pequenos. Atendendo à necessidade de medir a aceleração vertical, como exposto no ponto 4.4, definiu-se o lado direito da cintura como um dos sítios mais idóneos para utilizar o dispositivo.

4.8.4 Protótipo Implementado

Desenvolveu-se um protótipo final, tendo em conta as folhas de características dos vários componentes e das seguintes considerações de construção: o XBee foi colocado por forma a afastar o campo electromagnético, gerado pela antena, dos outros componentes. A placa de circuito impresso é composta por dois planos, um de massa e outro ligado ao pólo positivo da alimentação, de forma a reduzir a propagação de ruído electromagnético pelo módulo [116]. O GPS foi posicionado na parte dianteira, de forma a proporcionar a melhor recepção de sinal. O acelerómetro foi colocado no lado interior do módulo com o eixo dos ZZ em paralelo com a placa principal. Por fim o sensor de batimento cardíaco foi colocado perpendicularmente ao circuito, de forma a que o módulo colocado na lateral da cintura, esteja orientado, em relação à banda torácica, da forma indicada pelo fabricante, como apresentado na figura 4.24. Na figura 4.27 apresenta-se o protótipo final do módulo de monitorização de atletas.

(a) (b)

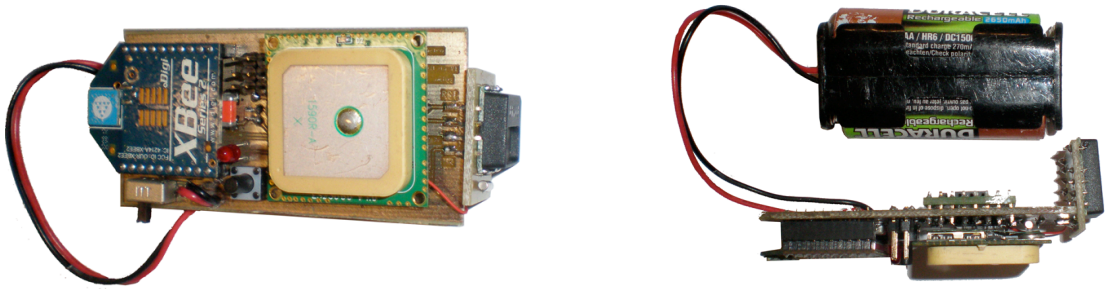


Figura 4.27 – Protótipo de monitorização de atleta: a) Frente; b) Topo.

4.8.5 Controlo Embebido

Análogo ao módulo de monitorização de atleta o programa pode ser subdividido em dois tipos de tarefas periódicas e assíncronas. As tarefas periódicas incluem a recolha da amostra da aceleração a cada 2 ms e o envio dos dados amostrados para o coordenador a cada segundo. As tarefas assíncronas são iniciadas por uma série de interrupções afectas aos componentes sensoriais embebidos no módulo. Estas são responsáveis por receber e processar informações do sensor de batimento cardíaco, do módulo GPS e da recepção e medição de mensagens de referência. Na figura 4.28 apresenta-se o diagrama de fluxo de dados que representa o funcionamento do programa desenvolvido. O diagrama encontra-se dividido em vários ramos que representam as diversas interrupções que são executadas assincronamente

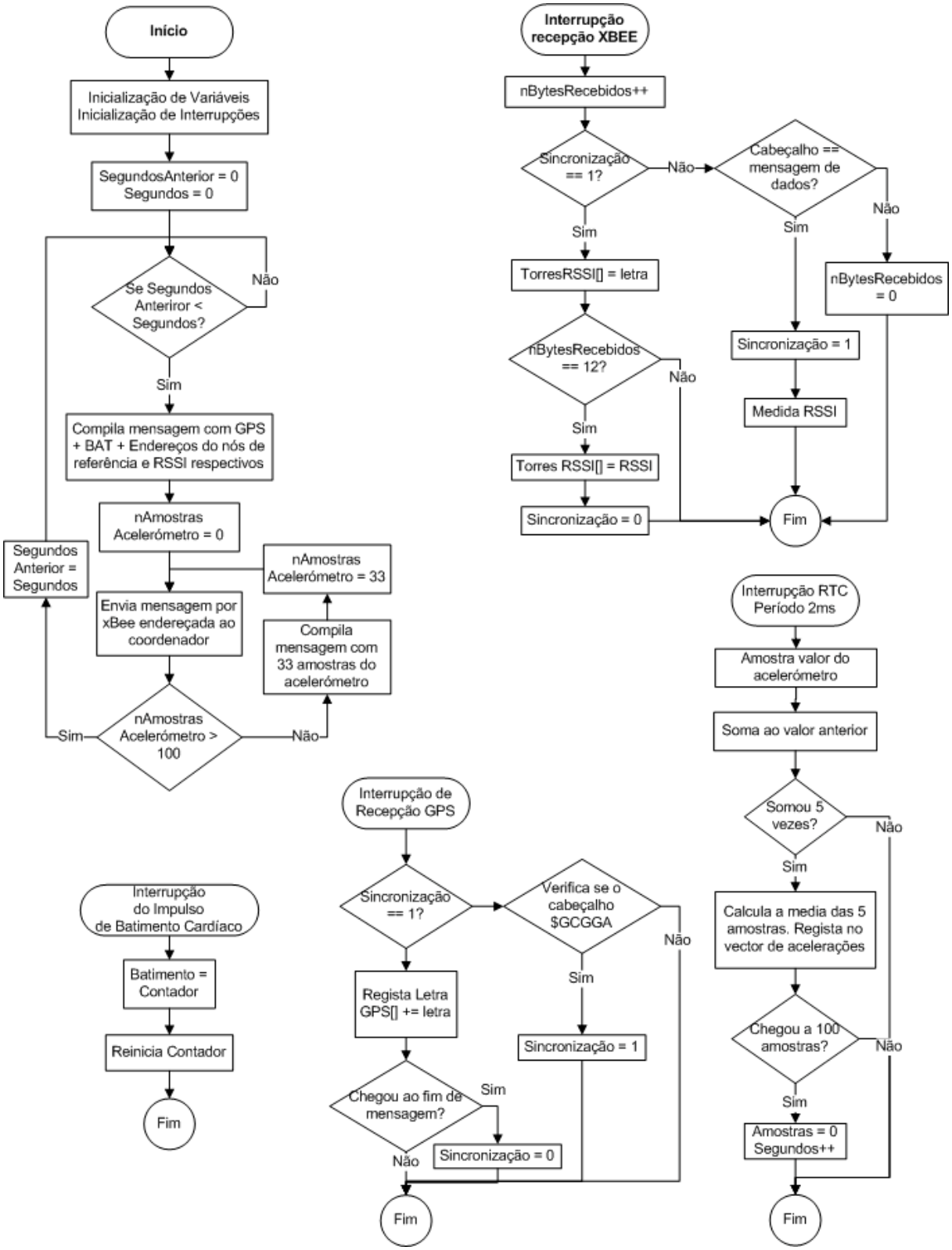


Figura 4.28 – Diagrama de fluxo de dados do controlo do módulo de monitorização de atletas.

No Anexo H apresenta-se o código fonte com os comentários respectivos que contribuem para uma melhor compreensão do programa.

4.9 Módulo de Nós de Referência

Os nós de referência têm duas tarefas importantes no sistema: garantir disponibilidade de rede, em qualquer ponto da área de treino, e fornecer um sinal de referência para o cálculo da localização do atleta, no caso de não haver recepção do sinal de GPS. Uma tarefa, secundária, é a de também registar o nível de sinal dos nós vizinhos, permitindo à aplicação poder inferir automaticamente a posição relativa dos nós de referência entre si.

4.9.1 Arquitectura

Na figura 4.29 apresenta-se o diagrama funcional dos nós de referência.

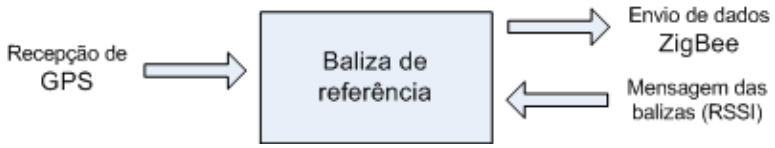


Figura 4.29 – Diagrama funcional dos nós de referência.

Este módulo, em relação ao *hardware*, é uma versão limitada do módulo de monitorização de atletas. Apresenta-se na figura 4.30 a arquitectura do nó de baliza de referência.

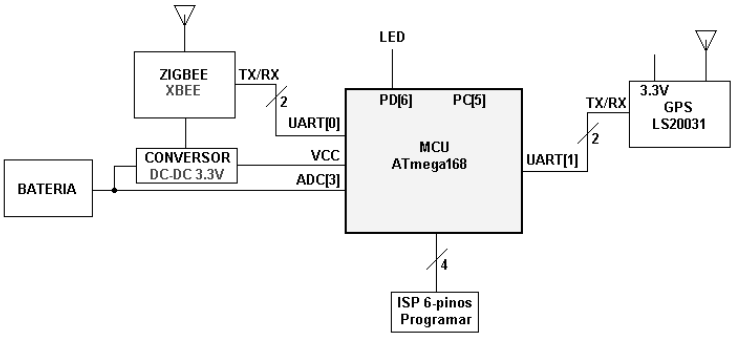


Figura 4.30 – Arquitectura das balizas de referência

4.9.2 Protótipo Implementado

O protótipo foi desenvolvido em conjunto com o aluno Tiago Braga, por ambos os projectos partilharem as mesmas necessidades e no sentido de partilhar a mesma base de circuito electrónico. Na figura 4.31 apresenta-se o protótipo dos nós de referência.

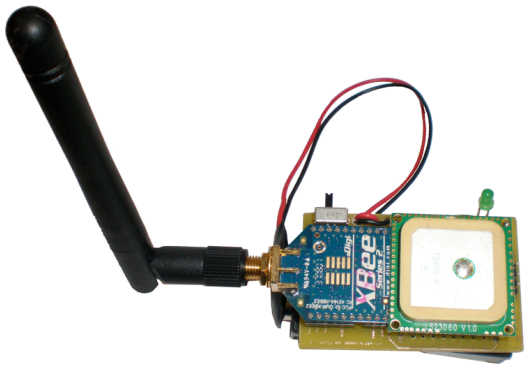


Figura 4.31 – Protótipo das balizas de referência

O controlo embebido é uma versão reduzida do módulo de monitorização de atleta realizando as mesmas tarefas de escuta da posição GPS, medição do RSSI das mensagens dos outros nós de referência e envio periódico dos dados. Ao código foi adicionada a função para enviar a cada 8 segundos a mensagem de referência para todos os nós vizinhos.

No Anexo H apresenta-se o código fonte com os comentários que permitem compreender o funcionamento do programa.

4.10 Armazenamento de Dados

Cada módulo da rede envia os seus dados, endereçados ao coordenador da rede. Este, por sua vez, entrega todas as mensagens por uma comunicação série ao computador. Para permitir uma maior flexibilidade de acesso aos dados e independência das aplicações desenvolvidas, definiu-se a utilização de uma base de dados para centralizar o armazenamento das várias mensagens dos módulos do sistema.

A aplicação que corre no computador central contém uma tarefa responsável por receber, interpretar e armazenar as diferentes mensagens na base de dados. Esta tarefa, distingue-se por pesquisar automaticamente a porta série na qual o coordenador, da rede XBee, está ligado ao computador, e ainda de criar a base de dados, caso esta ainda não exista. É requisito que esteja instalado e a correr um serviço de base de dados MySQL.

A base de dados foi estruturada de forma a guardar todos os dados disponibilizados pelos diversos equipamentos. Na figura 4.32 apresenta-se o diagrama de entidade relação (DER) da base de dados desenvolvida.

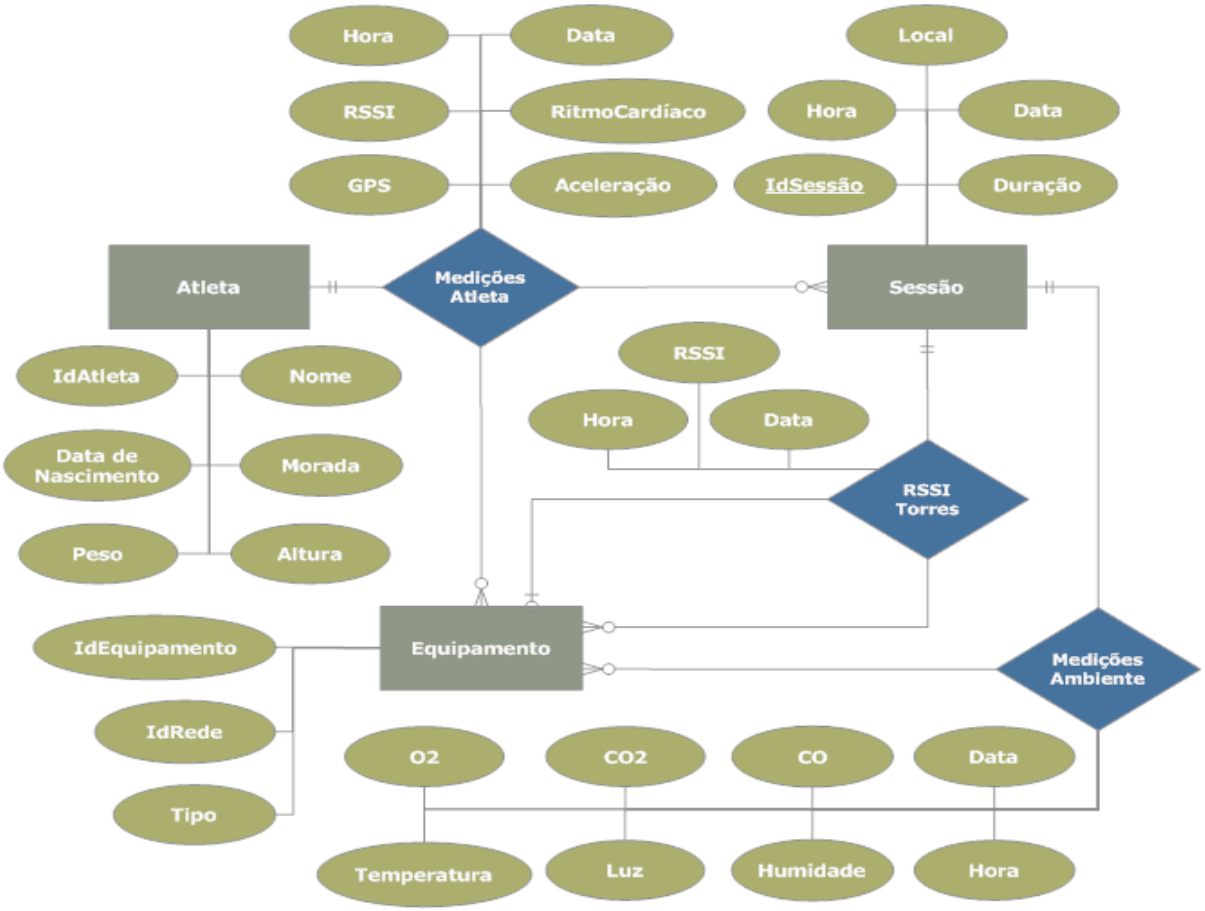


Figura 4.32 – DER da base de dados.

Num primeiro nível são guardadas, as informações relativas aos atletas (como o nome, data de nascimento, morada), a equipamentos (como a identificação única de rede, tipo de equipamento) e das sessões (como o dia, hora e local). Num segundo nível são registados, na base de dados, todos os valores gerados pelos diversos equipamentos. De entre esses valores estão as medições dos parâmetros de atletas obtidos via sensores, parâmetros ambientais e a posição GPS dos vários equipamentos e ainda dados para cálculo da posição relativa via trilateração.

O armazenamento de dados seguindo as relações apresentadas permite uma base de evolução para o futuro, por exemplo, a adição do registo do peso e altura dos atletas no início e fim de cada sessão. As várias relações permitem, por exemplo, monitorizar um atleta em diferentes sessões com diferentes equipamentos. Assim é possível comparar o desempenho desses equipamentos e inferir a repetibilidade de leituras entre eles. No anexo E apresenta-se o modelo relacional desenvolvido da base de dados utilizada pelo sistema.

4.11 *Interfaces* do Sistema

Um dos objectivos deste projecto era também o desenvolvimento de ferramentas ou aplicações que permitem a visualização dos parâmetros monitorizados em tempo real. Desenvolveu-se um conjunto de *interfaces*, no sentido de demonstrar os serviços que o sistema permite disponibilizar. Com as interfaces é possível aos utilizadores do sistema consultarem em tempo real os valores dos parâmetros monitorizados. Assim é possível ao treinador tomar decisões no momento do treino sobre a actividade a executar pelos diversos atletas ou um em particular.

Desenvolveu-se três aplicações: uma aplicação local que apresenta o desempenho do atleta com recurso a gráficos e representação sobre o mapa da área de treino; uma aplicação para a Internet que permite o acesso a dados remotamente; e por fim desenvolveu-se uma pequena aplicação para o telemóvel, que fornece aos seus utilizadores acesso directo aos dados dos módulos de monitorização ambiental.

4.11.1 Aplicação para Computador

A aplicação para o computador tem como objectivos gerais apresentar todos os dados monitorizados pelo sistema. Para o seu funcionamento identificou-se quatro processos base.

- Processo de recepção e armazenamento de dados: tarefa referida no ponto 4.10, que recebe as várias mensagens da rede; identifica os diferentes campos das mensagens consoante o valor no campo tipo de mensagem, como definido na tabela 4.2; e armazena-os nos respectivos campos da base de dados.
- Processo de posicionamento por GPS: posiciona a primeira torre no centro do gráfico; depois calcula as coordenadas X e Y relativas para os restantes nós.
- Processo de posicionamento por RSSI: por cada atleta pesquisa na base de dados os últimos três valores de RSSI de cada nó de referência. De seguida aplica o cálculo da trilateração.
- Processo de cálculo de velocidade: a partir dos dados de aceleração, aplica o processo descrito no ponto 4.4.

Os restantes processos incluem aplicação das fórmulas de conversão dos valores armazenados em informação útil, segundo as fórmulas descritas no desenvolvimento dos módulos, e na representação gráfica dos valores na janela da aplicação. Na figura 4.33 apresenta-se o fluxograma do funcionamento da aplicação e suas fronteiras.

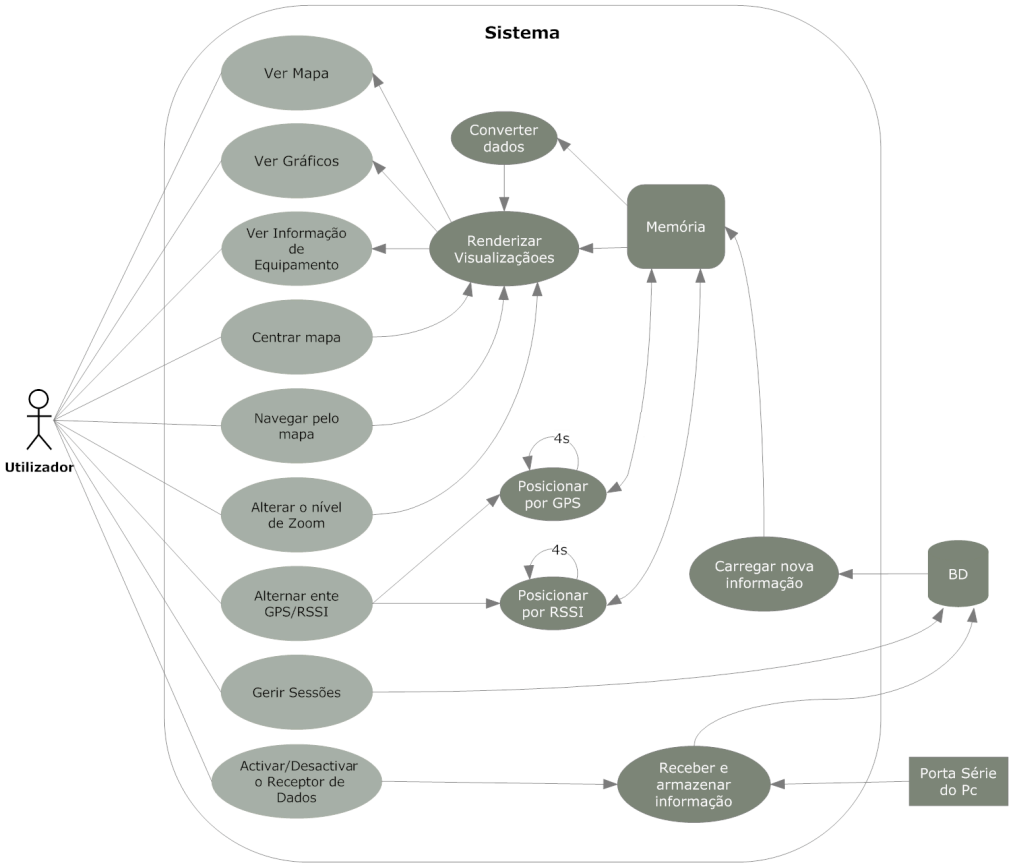


Figura 4.33 – Diagrama de fluxo de dados da aplicação local.

A aplicação desenvolvida em Java permite visualizar as medições em tempo real sobre o mapa do campo ou área de treino onde está a ser realizada a monitorização. A aplicação liga-se à base de dados e actualiza a interface sempre que a base de dados recebe novos dados. No Anexo F apresenta-se o código fonte das classes principais desenvolvidas para a aplicação.

Na figura 4.34 apresenta-se a janela da aplicação desenvolvida. Encontra-se dividida em três áreas: menu de ferramentas à esquerda, ao centro a área de visualização da posição dos atletas e nós de referência sobre o mapa, e na base da área do mapa uma área de visualização dos gráficos de evolução dos parâmetros dos atletas e ambientais. As ferramentas incluem, activação da tarefa de recepção de dados do coordenador referida no ponto 4.9, centralização do mapa no coordenador da rede, escolha do sistema de posicionamento entre GPS ou trilateração por RSSI, e a visualização de atletas activos. A área de visualização permite observar a posição actual do atleta sobre o mapa, e de consultar os dados actuais de cada elemento da rede ao clicar sobre os mesmos.

Figura 4.34 – Interface da aplicação local.

É possível navegar pelo mapa e ainda controlar o grau de ampliação. Na parte inferior da aplicação, na área de gráficos apresenta-se a evolução temporal dos valores monitorizados por atleta, como o ritmo cardíaco, velocidade e aceleração permitindo a comparação visual entre os vários atletas. Esta última área ainda permite consultar a evolução temporal dos parâmetros ambientais monitorizados.

4.11.2 Página de Internet

Para o acesso remoto, desenvolveu-se uma aplicação com funcionalidades básicas em PHP a partir do trabalho desenvolvido no ponto 3.2.1, que acede à base de dados MySQL do sistema e apresenta os dados da monitorização ambiental numa página *Web*. Esta página permite consultar os dados ambientais através de gráficos que representam a evolução temporal por várias horas ou dias.

A figura 4.35 mostra o interface da aplicação. Através de dois campos de texto é possível definir o período de visualização pretendido. O gráfico principal representa os dados para o período submetido. O parâmetro ambiental a visualizar é seleccionado no campo de lista apropriado.

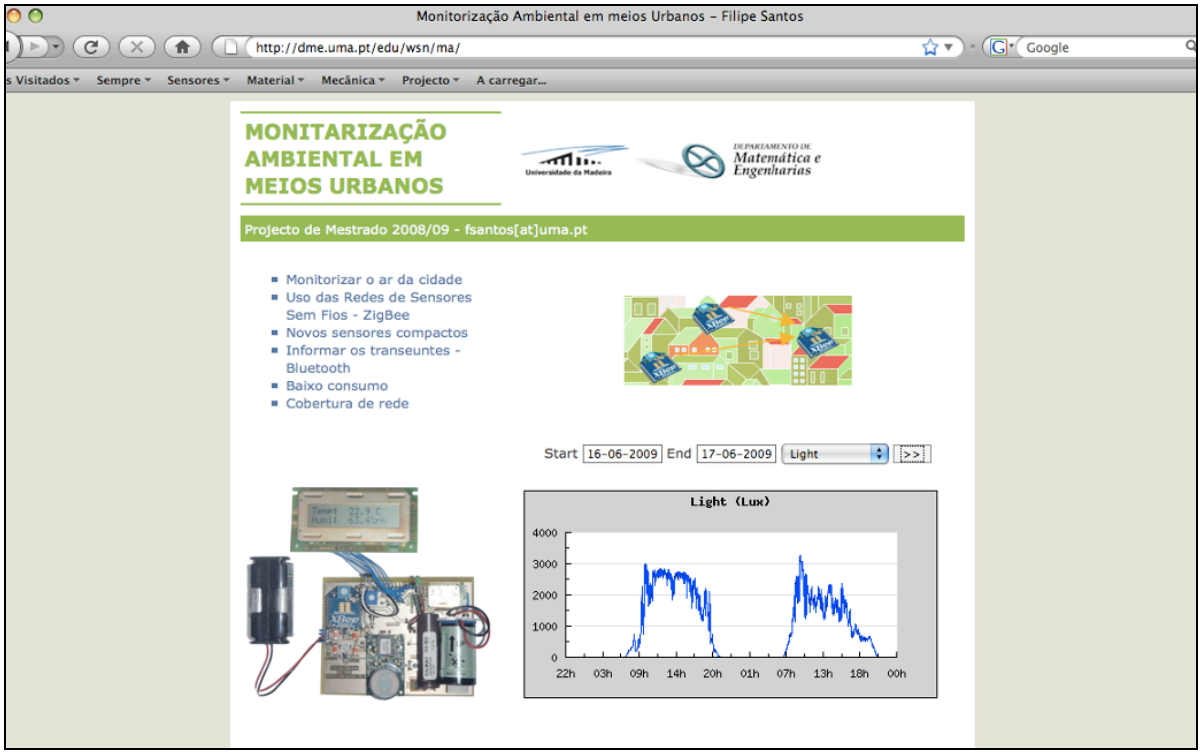


Figura 4.35 – Interface para a Internet.

Esta aplicação requer um servidor de páginas de Internet com motor PHP, e acesso à base de dados. Apresenta-se no Anexo I o código fonte desenvolvido para a aplicação.

4.11.3 Aplicação para Telemóvel

Para completar o sistema, foi desenvolvida uma pequena aplicação para apresentar os dados no telemóvel. Esta segue o digrama de fluxo de dados apresentado na figura 4.35. A aplicação começa por apresentar um mapa pré-carregado da área, e inicia a pesquisa de módulos de monitorização ambiental. O primeiro módulo encontrado é inquirido. E logo que os dados sejam recebidos, o mapa é centrado na posição do módulo e é apresentado os valores actuais dos parâmetros monitorizados e o nome da rua onde o módulo está localizado. Após dez segundos reinicia o processo de pesquisa de módulos.

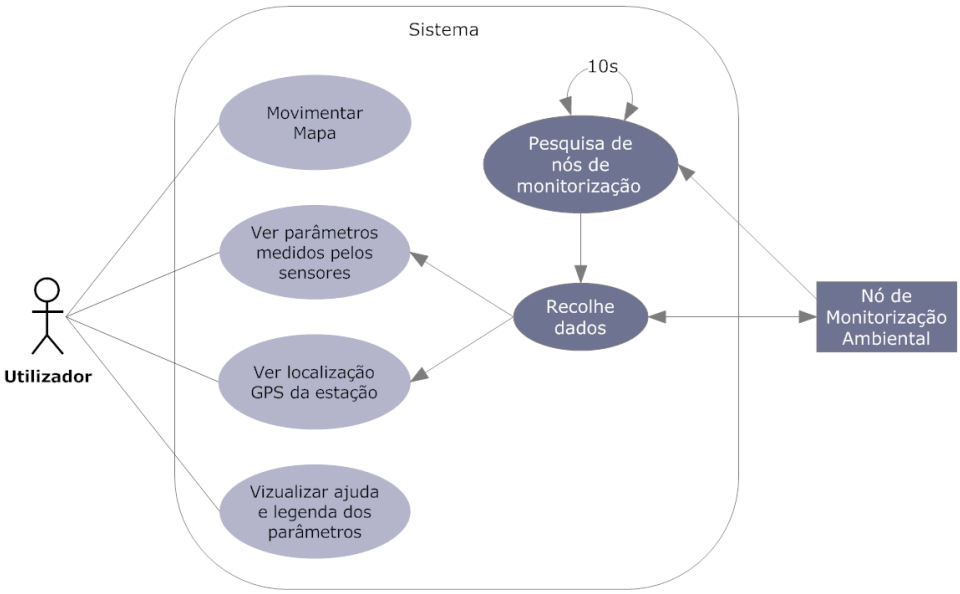


Figura 4.36 – Diagrama de fluxo de dados da aplicação para telemóvel.

O aplicativo foi desenvolvido na linguagem Java ME, que, embora não sendo a de maior desempenho para telemóveis, é uma das mais presentes em todos os modelos de telemóveis [117]. Na figura 4.35 apresenta-se a interface da aplicação para telemóvel. Numa parte superior apresenta-se os valores actuais do módulo de monitorização ambiental. O mapa mostra a localização do nó respectivo ao centro e em baixo apresenta a informação da localização que inclui o nome da rua.

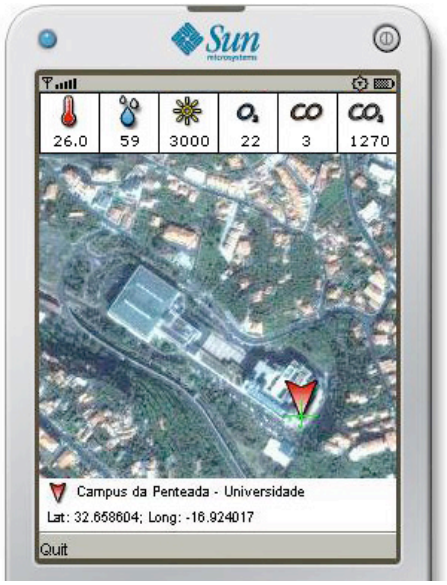


Figura 4.37 – *Interface* do telemóvel.

O funcionamento da aplicação destaca-se por ligar-se ao nó para obter os dados e desligar-se de seguida libertando, assim, o canal *Bluetooth* para outros telemóveis com a mesma aplicação. No Anexo G apresenta-se as classes desenvolvidas para a aplicação com os respectivos comentários para fácil compreensão.

5 TESTES E RESULTADOS

Neste capítulo expõe-se os resultados de funcionamento dos diversos elementos do sistema. Apresentam-se medidas de desempenho do sistema, expondo os pontos fortes e fracos do sistema, apontando sempre que possível os aspectos a melhorar num trabalho futuro.

5.1 Posição

A posição, como definido no ponto 4.4, pode ser obtida através do GPS. No caso de não haver recepção GPS, o sistema pode usar a trilateração para calcular a posição de cada atleta.

Realizaram-se vários testes a ambos os sistemas de localização para medir o seu desempenho. De seguida apresenta-se um conjunto de testes que permitiu comparar os sistemas com a posição real dos elementos do sistema de monitorização de atletas.

Utilizou-se uma área do parque de automóveis no exterior da Universidade para a realização dos testes. Definiu-se uma área rectangular de 20 x 23 metros e colocou-se quatro torres de referência em cada aresta do rectângulo. Uma das torres foi ligada ao computador, servindo de *gateway* para o sistema. Dois atletas com módulos de monitorização colocaram-se em duas posições distintas. Na figura 5.1 apresenta-se a disposição real do sistema [118].

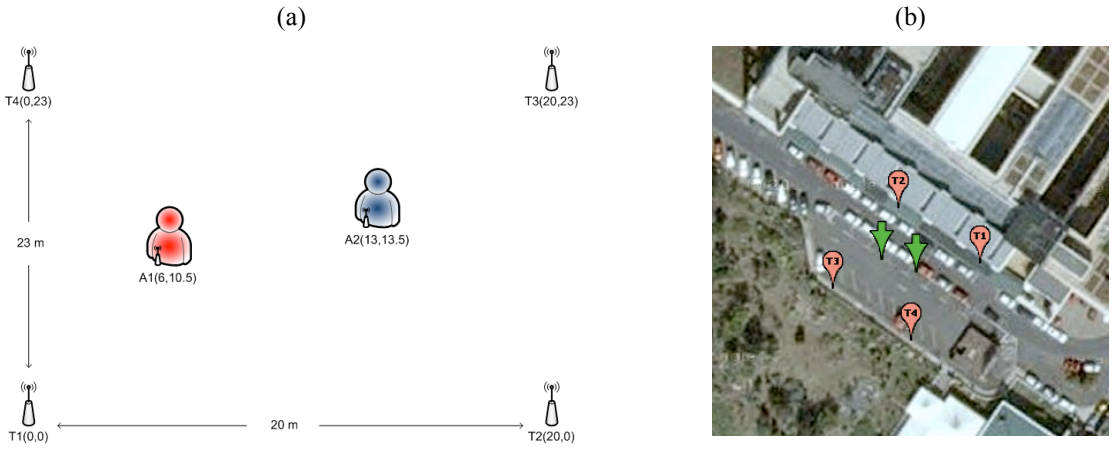


Figura 5.1 – Sistema para teste da posição: a) Posição real (metros); b) Posição sobre mapa.

O fabricante dos módulos GPS indica uma precisão de 2,5 metros. Dos testes verificou-se que a posição indicada pelo GPS não é estável, variando ao longo do tempo. Na tabela 5.1 apresenta-se uma sequência de posições globais indicadas pelo módulo GPS numa situação real estacionária.

TABELA 5.1 – SEQUÊNCIA DE LEITURAS DO MÓDULO GPS NUMA POSIÇÃO REAL FIXA.

Tempo (s)	Posição global no formato: gg mm.mmmm N/S ggg mm.mmmm E/W
1	32 39.5207N 016 55,4719W
2	32 39.5204N 016 55,4718W
3	32 39.5206N 016 55,4717W
4	32 39.5209N 016 55,4712W
5	32 39.5204N 016 55,4716W

Com recurso ao algoritmo apresentado em [119] que calcula a distância entre duas latitudes e longitudes, calculou-se a distância entre as posições mais distantes entre si.

Obteve-se uma variação de 1,7 metros para uma posição fixa do equipamento. Isto demonstra a necessidade de filtrar a informação entregue pelo GPS, ou de pelo menos compará-la com a informação do acelerómetro de forma a confirmar que existe movimento.

Outra observação pertinente feita do funcionamento do sistema, é que a periodicidade de monitorização de um 1 s, no módulo de atletas, parece visualmente que perde alguma informação, pois se um atleta estiver a correr a elevada velocidade, como exemplo: 8 m/s, em 5 segundos terá percorrido 40 metros, o que na aplicação do computador mostra apenas 5 saltos. Identifica-se assim uma melhoria a efectuar à aplicação local, que pode através da velocidade e direcção média descrever o percurso intermédio dos atletas entre as posições reportadas, reduzindo assim, a percepção de estar aos saltos na janela da aplicação. Outra possibilidade é fazer uso da frequência máxima de posição do GPS, de 5 Hz, para enviar 5 posições globais por segundo.

Utilizando a aplicação desenvolvida para o computador, avaliou-se a posição dos elementos do sistema via a informação GPS de cada um. Na figura 5.2 a) apresenta-se a distribuição obtida sobre o mapa. Na figura 5.2 b) apresenta-se as distâncias calculadas entre as várias posições. Observou-se que ambos os pontos do sistema aparecem transladados na mesma direcção em relação ao mapa da Google. Como utilizou-se dois módulos iguais para as medidas, enumeram-se duas causas possíveis: desvio da imagem do mapa Google ou erro de desvio de ambos os módulos GPS. As diferenças aos valores reais mantiveram-se abaixo do erro admissível de 2,5 metros.

(a) (b)

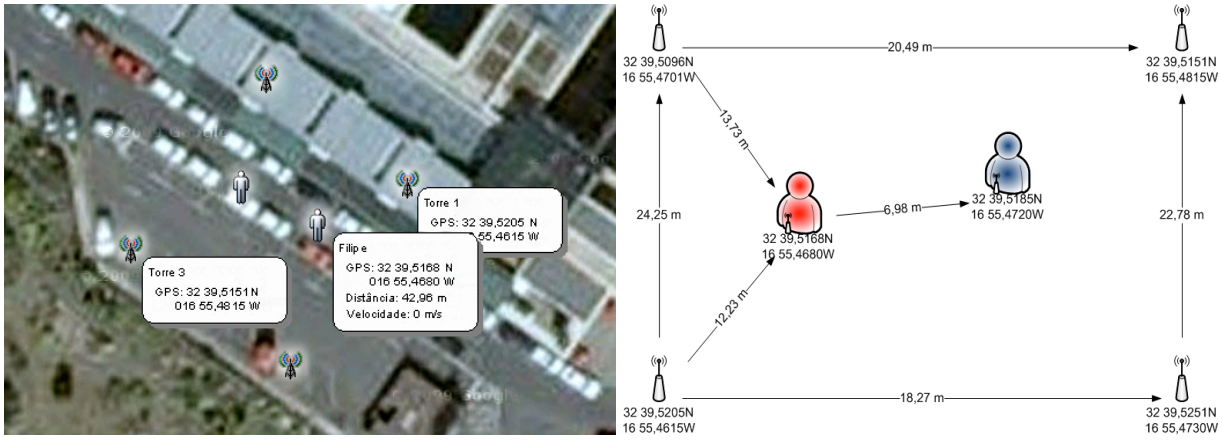


Figura 5.2 – Posicionamento por GPS: a) vista da aplicação; b) distâncias calculadas a partir das posições.

Para o mesmo sistema de testes, registou-se a reposta do sistema pela aplicação do posicionamento por trilateração dos níveis de sinal das várias ligações rádios. Durante os vários testes identificou-se uma série de dificuldades na medição da posição via nível de sinal recebido. Eram esperadas variações elevadas nos valores de distância baseados na medida de RSSI, como demonstrado pela variação do nível do sinal na figura 3.44. Mesmo o sistema estando dentro da parte inicial da curva onde a atenuação é acentuada com a distância as variações podem fazer inferir distâncias com erros até 50 metros. Há, assim, necessidade de realizar a média de várias medidas de RSSI para a mesma localização, para então poder-se calcular um valor de distância aproximado. Este requisito foi limitado desde logo pelo funcionamento do módulo rádio XBee, que não permite uma periodicidade de *broadcast* inferior aos 8 segundos. Estando o desenvolvimento do sistema numa fase avançada e dado o limites de tempo, identificou-se apenas um rádio no mercado que permite o processo do registo de várias medidas de nível de sinal com períodos na ordem dos 40 ms, o modelo

CC2431 da Texas Instruments [120]. Daí a apresentação de um sistema de testes em que os atletas estão numa posição fixa, no sentido de demonstrar a possibilidade de se inferir a posição relativa ao fim de algum tempo de medidas. Assim programou-se a aplicação para calcular a posição actual via trilateração da média dos valores de distância acumulados no último minuto. Para um período de medições entre os vários rádios de 8 segundos, obtém-se 7 medidas por minuto. Na figura 5.3 a) apresenta-se o posicionamento por trilateração ao fim de um minuto. Observou-se que as posições dos atletas representam a suas posições relativas no sistema, mas com um grau de erro elevado. Na figura 5.3 b) apresenta-se o cálculo da distância às posições reais.

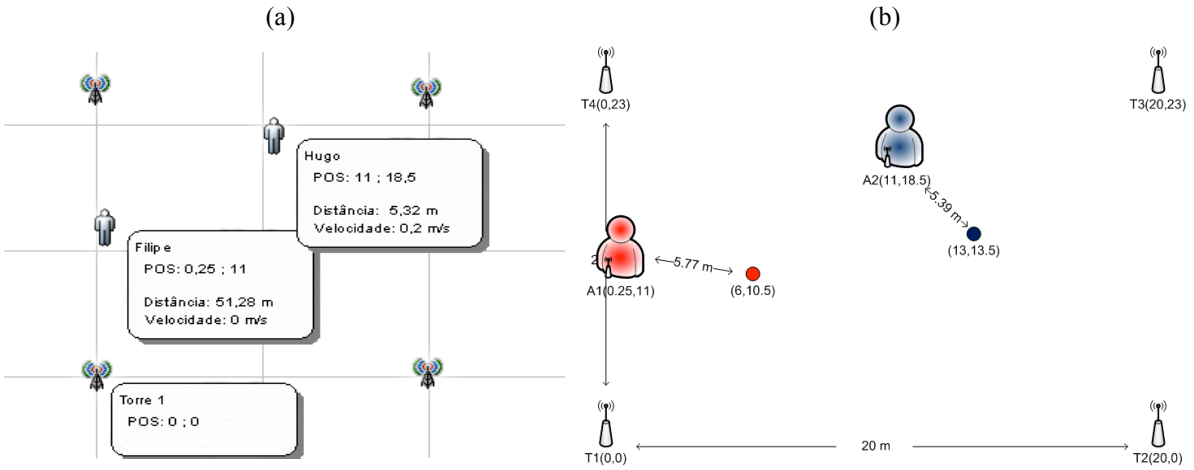


Figura 5.3 – Posicionamento por RSSI: a) vista da aplicação; b) distâncias aos pontos reais.

Os atletas aparecem cerca de 6 metros das posições reais, apontando para um erro de ±6 metros nesta situação de teste, ou seja, 30 % da lateral da área de teste. Para provar a potencialidade do método mediu-se o nível de sinal na posição dos atletas com recurso ao analisador de espectro e comparou-se os resultados com os do RSSI. Verificou-se que se a precisão da medida de RSSI fosse similar ao do analisador de espectro podia-se obter melhores resultados. Na figura 5.4 apresenta-se os níveis de sinal registados e respectivas distâncias calculadas.

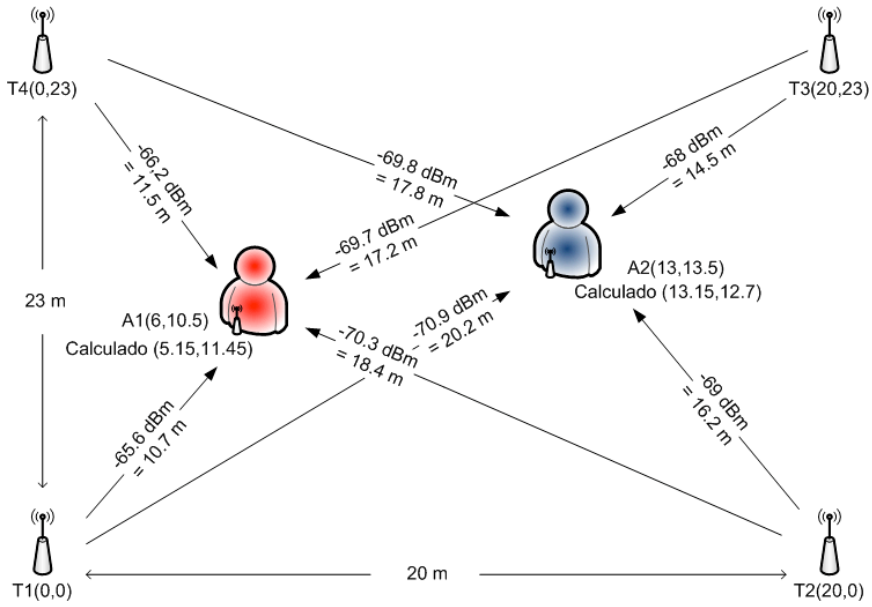


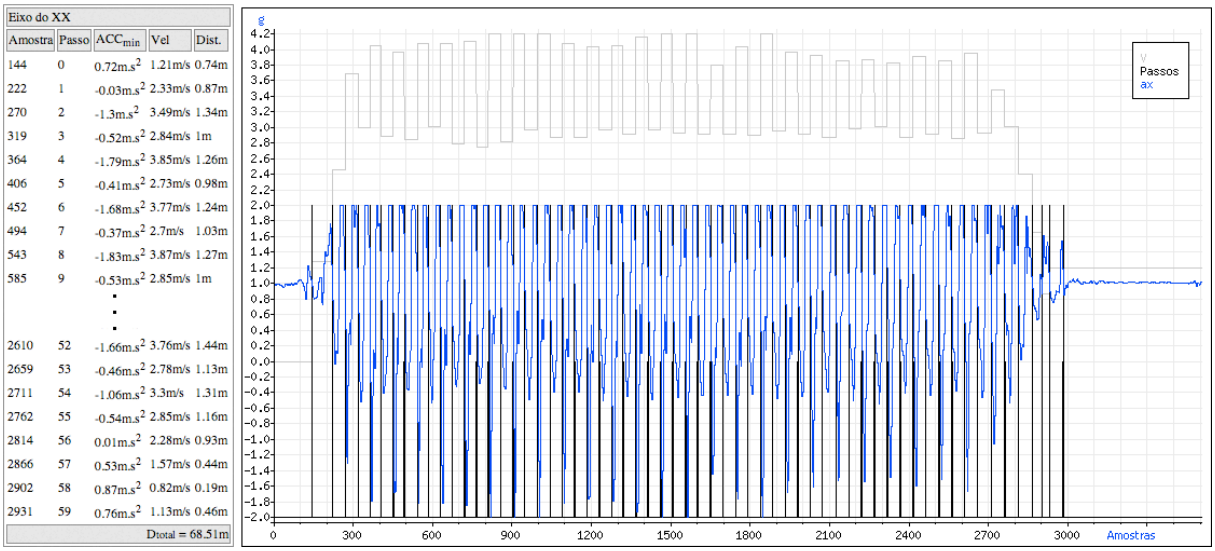
Figura 5.4 – Posicionamento via medidas de nível de sinal com analisador de espectros.

As medidas para a situação da figura 5.4 foram realizadas com a orientação da antena para a torre de referência, o que implica num sistema real a utilização de uma antena com um diagrama de radiação circular em ambos os planos. A diferença de posição entre a calculada por trilateração e a real divergiu de 1,28 metros para o atleta 1 e de 0,66 metros para o atleta 2, ou seja, 8,4 % e 3,3 % de erro em relação à largura da área de teste. Valores aceitáveis se considerarmos que um atleta ocupa um círculo com um diâmetro de 50 cm, ou seja, 2,5 % da largura da área de teste.

Por fim pode-se sublinhar que o aumento de nós de referência permite aumentar a precisão do sistema. Por outro lado as situações apresentadas, embora num ambiente real, não contam com as interferências de outros atletas. Interferências nos caminhos de sinal que vão de certeza afectar a precisão da medida. Nesta situação podem ser implementados algoritmos de localização relativa, que englobam o sinal dos vizinhos e dos atletas vizinhos, como apresentado por [11].

5.2 Velocidade

A velocidade pode ser obtida directamente a partir do módulo GPS, ou calculada a partir das distâncias e tempo entre as posições registadas do atleta. Como a posição GPS varia um pouco, em particular, na situação do atleta estar imóvel, como demonstrado pela tabela 5.1 a informação de posição do GPS não é estável. Para mais, como já foi referido anteriormente, em certas situações não temos acesso ao sinal GPS. Sendo assim, utilizou-se os dados da aceleração vertical do acelerómetro para cálculo da velocidade. Realizou-se medidas para diferentes distâncias e velocidades. Desenvolveu-se um algoritmo em PHP que permitiu avaliar graficamente os resultados da equação (4.1). O estudo [80] aponta que os melhores resultados são obtidos pela calibração. Para isso o atleta percorreu uma distância conhecida e nos cálculos ajustou-se o *h* para coincidir com a distância real. De seguida apresenta-se três resultados representativos dos valores obtidos. Na figura 5.5 apresenta-se o registo da aceleração vertical para uma distância de 75 metros a passo rápido. Os traço pretos representam a identificação de cada passo. Isto é definido quando a aceleração passa pelo valor de 1 g no sentido descendente. A linha azula corresponde à aceleração, a cinzento à velocidade instantânea. À esquerda apresenta os cálculos parciais das equações (4.1) e (4.2), e ainda o total da distancia percorrida.



Eixo do XX

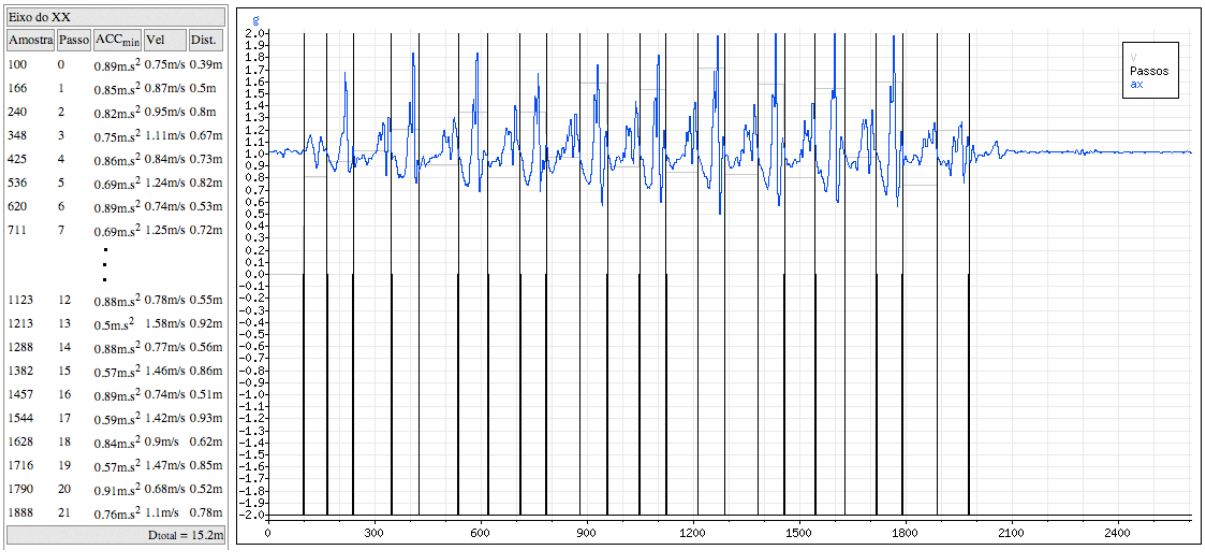
Amostra	Passo	ACC_{min}	Vel	Dist.
144	0	0.72m.s^2	1.21m/s	0.74m
222	1	-0.03m.s^2	2.33m/s	0.87m
270	2	-1.3m.s^2	3.49m/s	1.34m
319	3	-0.52m.s^2	2.84m/s	1m
364	4	-1.79m.s^2	3.85m/s	1.26m
406	5	-0.41m.s^2	2.73m/s	0.98m
452	6	-1.68m.s^2	3.77m/s	1.24m
494	7	-0.37m.s^2	2.7m/s	1.03m
543	8	-1.83m.s^2	3.87m/s	1.27m
585	9	-0.53m.s^2	2.85m/s	1m
2610	52	-1.66m.s^2	3.76m/s	1.44m
2659	53	-0.46m.s^2	2.78m/s	1.13m
2711	54	-1.06m.s^2	3.3m/s	1.31m
2762	55	-0.54m.s^2	2.85m/s	1.16m
2814	56	0.01m.s^2	2.28m/s	0.93m
2866	57	0.53m.s^2	1.57m/s	0.44m
2902	58	0.87m.s^2	0.82m/s	0.19m
2931	59	0.76m.s^2	1.13m/s	0.46m

Dtotal = 68.51m

Figura 5.5 – Resultados da aceleração vertical para uma distância de 75 metros.

Obteve-se um total de 68,51 m, cerca de 8,7 % abaixo do valor real. Repetiu-se a experiência para 15 metros. Na figura 5.6 apresenta-se os resultados para a mesma distância de 15 m, mas a velocidades de passo diferentes, como se pode observar dos resultados da velcidade. A figura 5.6a) representa um passo normal, e a figura 5.6b) representa um passo rápido.

(a)

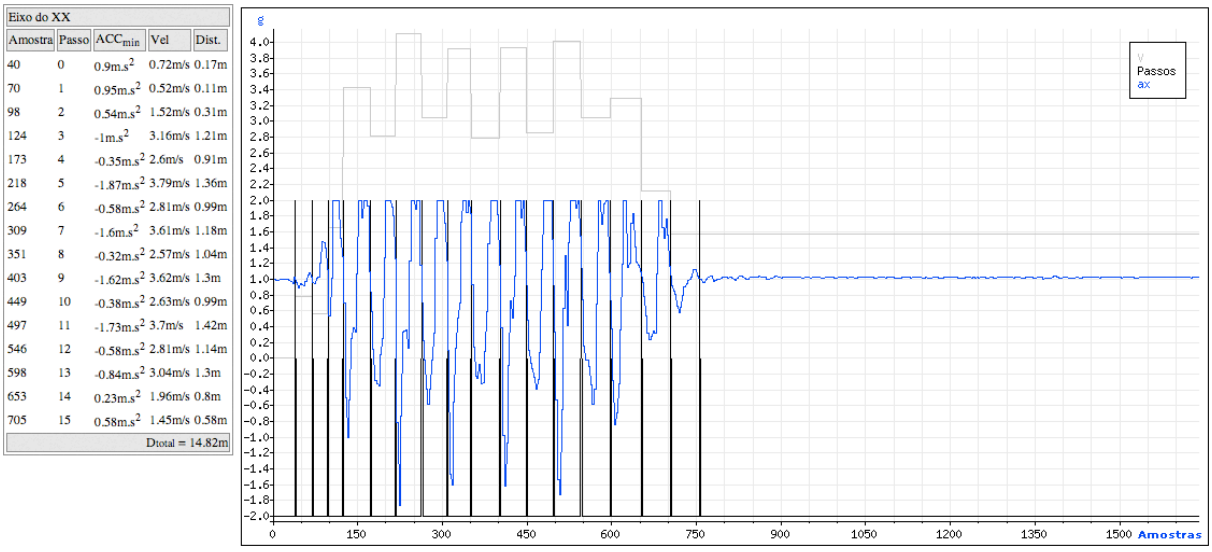


Eixo do XX

Amostra	Passo	ACC_{min}	Vel	Dist.
100	0	0.89m.s^2	0.75m/s	0.39m
166	1	0.85m.s^2	0.87m/s	0.5m
240	2	0.82m.s^2	0.95m/s	0.8m
348	3	0.75m.s^2	1.11m/s	0.67m
425	4	0.86m.s^2	0.84m/s	0.73m
536	5	0.69m.s^2	1.24m/s	0.82m
620	6	0.89m.s^2	0.74m/s	0.53m
711	7	0.69m.s^2	1.25m/s	0.72m
1123	12	0.88m.s^2	0.78m/s	0.55m
1213	13	0.5m.s^2	1.58m/s	0.92m
1288	14	0.88m.s^2	0.77m/s	0.56m
1382	15	0.57m.s^2	1.46m/s	0.86m
1457	16	0.89m.s^2	0.74m/s	0.51m
1544	17	0.59m.s^2	1.42m/s	0.93m
1628	18	0.84m.s^2	0.9m/s	0.62m
1716	19	0.57m.s^2	1.47m/s	0.85m
1790	20	0.91m.s^2	0.68m/s	0.52m
1888	21	0.76m.s^2	1.1m/s	0.78m

Dtotal = 15.2m

(b)



Eixo do XX

Amostra	Passo	ACC_{min}	Vel	Dist.
40	0	0.9m.s^2	0.72m/s	0.17m
70	1	0.95m.s^2	0.52m/s	0.11m
98	2	0.54m.s^2	1.52m/s	0.31m
124	3	-1m.s^2	3.16m/s	1.21m
173	4	-0.35m.s^2	2.6m/s	0.91m
218	5	-1.87m.s^2	3.79m/s	1.36m
264	6	-0.58m.s^2	2.81m/s	0.99m
309	7	-1.6m.s^2	3.61m/s	1.18m
351	8	-0.32m.s^2	2.57m/s	1.04m
403	9	-1.62m.s^2	3.62m/s	1.3m
449	10	-0.38m.s^2	2.63m/s	0.99m
497	11	-1.73m.s^2	3.7m/s	1.42m
546	12	-0.58m.s^2	2.81m/s	1.14m
598	13	-0.84m.s^2	3.04m/s	1.3m
653	14	0.23m.s^2	1.96m/s	0.8m
705	15	0.58m.s^2	1.45m/s	0.58m

Dtotal = 14.82m

Figura 5.6 – Resutlados da aceleração para uma distância de 15 metros, com a equação calibrada: a) passo normal; b) passo rápido.

Os resultados deram 15,2 m e 14,82 m correspondendo a +1,3 % e -1,2 % respectivamente. Estes valores embora animadores requerem maior número de testes para entender todas as variáveis físicas que influenciam o cálculo. Como exemplo apresenta-se na figura 5.7 os resultados de um teste com um segundo indivíduo, numa localização diferente para uma distância de 18 metros, já com a equação calibrada. A figura apresenta os outros dois eixos ortogonais, para obter-se uma percepção das variações registadas. Mais uma vez obteve-se bons resultados com uma diferença inferior aos 2 % em relação à distancia real. O valor do teste de 75 metros divergiu em relação aos dados do estudo [80] que aponta um erro máximo de ±15 % sem calibração e um erro médio inferior aos 3,4 %.

Eixo do YY				
9	0	0.38m.s^2	1.63m/s	0.83m
74	1	-0.19m.s^2	2.27m/s	1.03m
132	2	0.02m.s^2	2.06m/s	0.98m
193	3	0.02m.s^2	2.06m/s	0.93m
251	4	0.24m.s^2	1.81m/s	0.92m
316	5	0.2m.s^2	1.86m/s	0.8m
371	6	0.22m.s^2	1.83m/s	0.93m
436	7	0.13m.s^2	1.94m/s	0.82m
490	8	0.11m.s^2	1.96m/s	0.95m
552	9	0.24m.s^2	1.81m/s	0.82m
610	10	0.01m.s^2	2.07m/s	0.99m
671	11	-0.22m.s^2	2.3m/s	0.99m
726	12	-0.37m.s^2	2.43m/s	1.21m
790	13	0.21m.s^2	1.85m/s	0.75m
842	14	0.49m.s^2	1.48m/s	0.79m
910	15	0.11m.s^2	1.96m/s	0.93m
971	16	0.39m.s^2	1.62m/s	0.82m
1036	17	0.3m.s^2	1.74m/s	0.69m
1087	18	0.52m.s^2	1.43m/s	0.84m
1162	19	0.43m.s^2	1.57m/s	0.8m
1227	20	0.72m.s^2	1.09m/s	0.43m
		Dtotal = 18.24m		

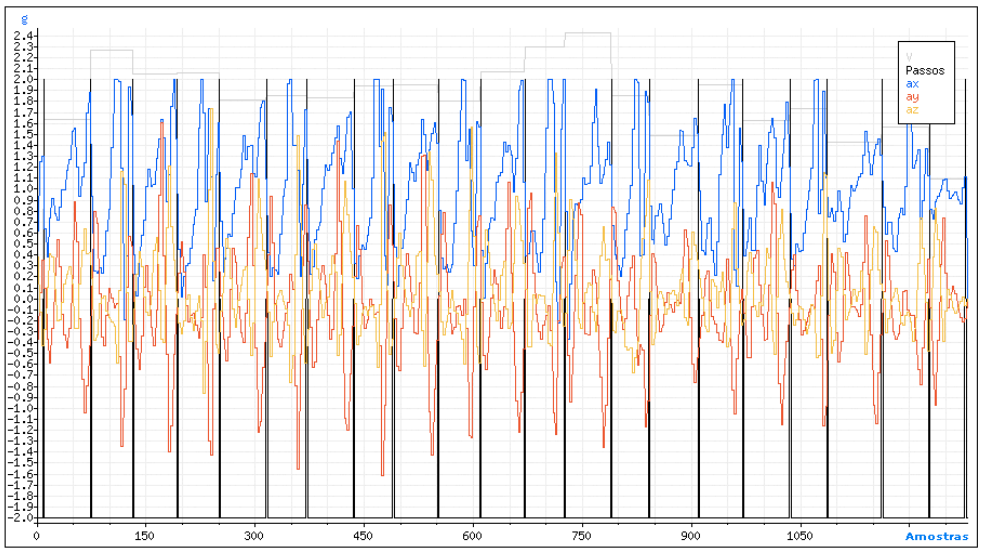


Figura 5.7 – Resultados das medidas da aceleração para uma distância de 18 metros, com a equação calibrada.

Durante os testes identificou-se um trabalho realizado pela Analog Devices [121], que também utiliza a medição da aceleração vertical, para inferir a distância e velocidade percorrida. O trabalho realizado optimiza a precisão da distância medida a partir de um acelerómetro. O trabalho apresenta erros máximos de 6 % para a situação de não estar calibrado. Identifica-se assim, um trabalho futuro de aplicar o algoritmo apresentado em [121] e comparar os resultados.

5.3 Ritmo Cardíaco

A monitorização do ritmo cardíaco, é realizada por uma solução completa da marca Polar. Esta solução utiliza uma banda torácica com dois eléctrodos. Esta configuração causa dificuldades para iniciar o registo do ritmo cardíaco, pois a detecção só se realiza se houver um electrólito entre os eléctrodos e o peito do atleta. Isto implica no início do treino molhar os eléctrodos, pois sem isto os impulsos do batimento cardíaco não são detectados correctamente. Isto obriga a introdução de funções adicionais no programa que calcula o ritmo cardíaco do atleta, para não passar informação discrepante para a interface. Estas funções incluem realização da média com o último período e verificação se a variação entre períodos não foi superior a um certo nível.

O fabricante afirma que o conjunto oferece a precisão de um electrocardiograma. A falta de equipamento não nos permitiu comparar os valores registados. No entanto, no sentido de avaliar a resposta da monitorização do ritmo cardíaco, efectuou-se o registo de uma actividade física de intensidade variável. Na figura 5.8 apresenta-se os resultados do ritmo cardíaco para a seguinte actividade: o atleta parte do repouso iniciando uma marcha, ao fim de 1 minuto e meio, passa para marcha rápida de meio minuto, aos dois minutos e meio repete e depois continua a marcha. É visível o aumento do ritmo cardíaco perante o esforço do atleta.

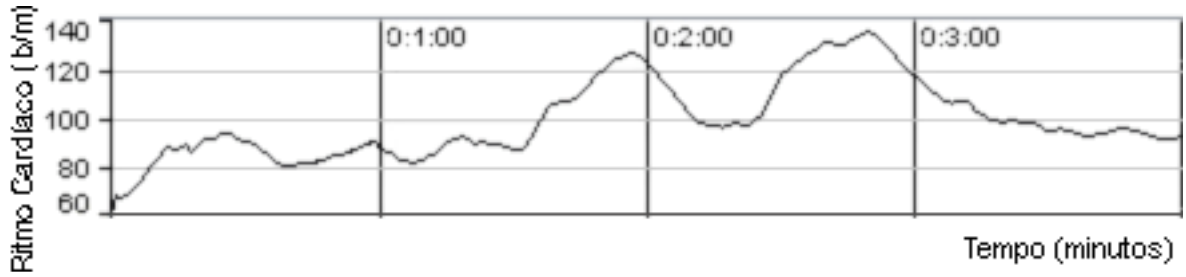


Figura 5.8 – Resultados das medidas do sensor de ritmo cardíaco, durante uma corrida (eixos).

Verifica-se ainda que, embora o atleta retorne a velocidades iguais às do início do registo, o valor médio do ritmo cardíaco foi subindo ao longo do teste. Isto é justificado pela falta de tempo para relaxamento do coração [24].

5.4 Monitorização Ambiental

Efectuou-se testes de longevidade, em que manteve-se o módulo de monitorização ambiental em funcionamento ao longo de dois meses, a fim de verificar a repetibilidade da resposta dos sensores. O módulo foi colocado junto da janela do laboratório de Telecomunicações e Redes. De seguida, apresentam-se gráficos representativos dos resultados obtidos.

Na figura 5.8 apresenta-se o registo do sensor de CO_2 durante dois dias de Julho. Como indicado pelo fabricante o sensor regista valores até o mínimo de 200 ppm, e detecta rapidamente o aumento de concentração do gás. O ruído presente nos valores mais baixos pode ser reduzido até 20 ppm aumentando, para isso, o factor de média no próprio sensor. Pode-se observar que a concentração de CO_2 aumentou durante o dia, associando-se à passagem de veículos. (Chegada de todos os veículo de manhã, maioria circula no almoço e vão saindo pela tarde fora)

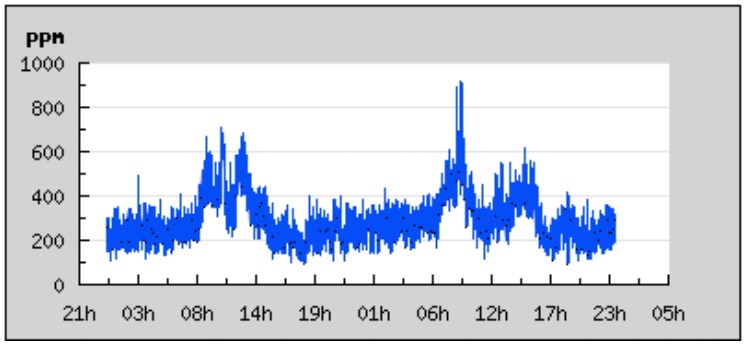


Figura 5.9 – Registo de CO_2 durante dois dias.

A saída do sensor de O_2 foi ajustada num ambiente de ar livre, no exterior da universidade, para um valor de 20,9%. O registo do sensor de O_2 para três dias pode ser observado na figura 5.9. Verifica-se que o valor mantém-se em torno de um valor estável, de 20,9%, apresentando uma pequena oscilação em torno do LSB resultante do processo de amostragem.

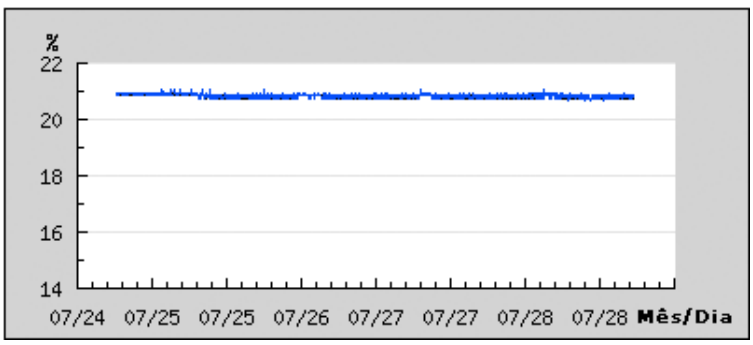


Figura 5.10 – Registo de O_2 durante três dias.

O sensor de monóxido de carbono, apresentou um desvio de 7,5 ppm após o circuito de amplificação e da conversão. A figura 5.10 a) apresenta o desvio registado no início da monitorização. O sinal apenas varia de um LSB do conversor ADC do microcontrolador. Não havendo a disponibilidade de meios de calibração, num ambiente de ar visivelmente limpo, ajustou-se o desvio a zero subtraindo-o no microncontrolador. Nas condições de teste, não se registou qualquer valor de concentração no sensor de CO, assim para demonstrar o seu funcionamento, expôs-se o módulo ao fumo proveniente de um papel acabado de queimar. Na figura 5.10 b) apresenta-se o resultado do teste, em que o sensor detectou de imediato o aumento de concentração de monóxido de carbono.

(a) (b)

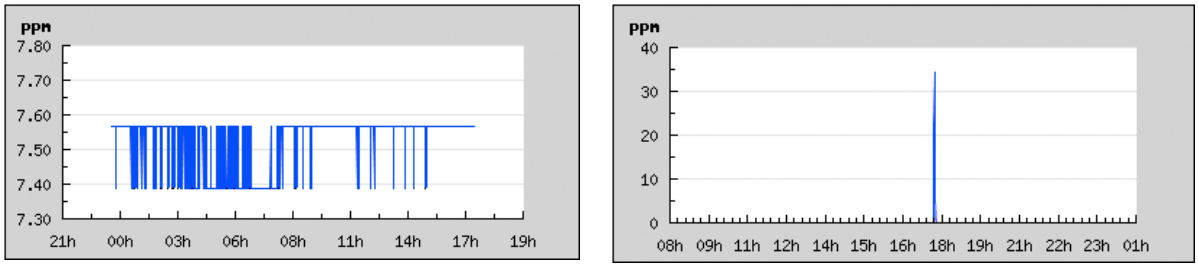


Figura 5.11 – Monitorização de CO: a) desvio inicial; b) resposta ao fumo de um cigarro.

Os dados monitorizados pelo sensor de temperatura e humidade, apresentados nas figuras 5.12, repetem os ciclos esperados ao longo de quatro dias. Consegue-se verificar que a humidade desce com o aumento da temperatura.

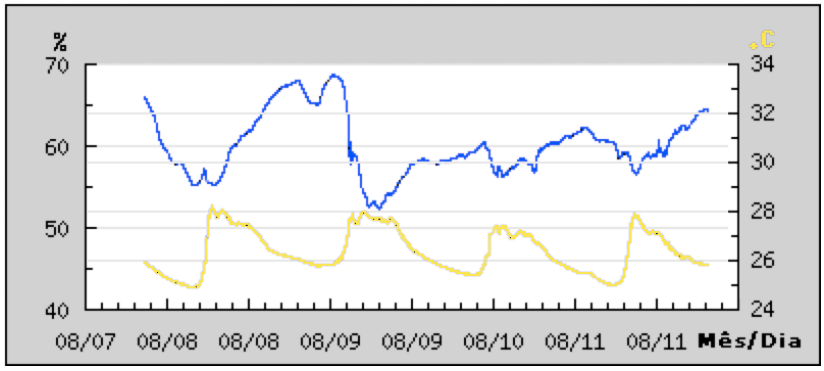


Figura 5.12 – Monitorização ambiental durante 4 dias da humidade e temperatura.

Na figura 5.13 apresentam-se os resultados para a intensidade luminosa. O sensor de luz, regista a variação da intensidade solar durante o dia, o valor apresentando é relativamente baixo, visto o módulo estar num local em que recebe luz directa do sol apenas durante uma hora da manhã como se pode verificar pelo pico durante a manhã.

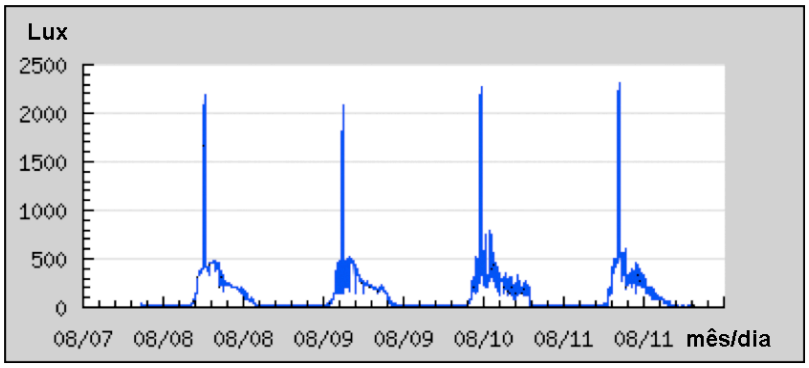


Figura 5.13 – Monitorização ambiental durante 4 dias da intensidade luminosa.

De forma geral os sensores responderam conforme esperado pela leitura das suas características. Os resultados reflectem a evolução dos parâmetros monitorizados. À excepção do sensor de oxigénio os sensores são comercializados já calibrados de fábrica, e para os outros fornece instruções de calibração simples em situações de ar limpo. Isto não garante a reposta correcta do sensor quando este está integrado com outros sensores numa pequena caixa onde a ventilação apresenta restrições. É importante frisar a necessidade de calibrar, com equipamento adequado, e avaliar o tempo de resposta dos sensores quando integrados no módulo.

5.5 Redes Sem Fios

Estudos demonstram que o *ZigBee* coexiste com as rede sem fios baseadas no padrão 802.11 [122]. Na literatura encontra-se estudos em que utilizam o XBee aplicado à monitorização de pessoas em movimento pela rede. Este num meio com interferências no canal rádio, como por exemplo o forno microondas, ainda apresenta um elevado sucesso de transferência de dados, superior a 96% [123].

No sentido de verificar o desempenho da rede na situação de mobilidade realizou-se o seguinte teste: distribuiu-se espacialmente três nós de referência ao longo dos corredores da Universidade da Madeira com uma distância de trinta metros entre cada torre. Depois movimentou-se o nó de monitorização de atleta pelo percurso entre o coordenador da rede e o último nó de referência, isto no intuito de obrigar as mensagens a serem encaminhadas pelos vários nós. Da tabela 4.1 observa-se que o módulo de monitorização de atletas é o nó da rede que envia maior quantidade de dados. Este módulo envia 247 bytes por segundo distribuídos por quatro mensagens, dos quais 200 bytes são informação da aceleração durante esse período. A velocidade máxima da rede é de 250 kbps. Se considerar-se que a cada mensagem é adicionada uma série de cabeçalhos das várias camadas do protocolo *ZigBee*, pode-se estimar que no máximo é transmitido o dobro dos dados por cada segundo, ou seja, cerca de 500 bytes por segundo. Assim, nesta situação só para um atleta é requerido uma taxa de transferência de dados de cerca de 5 kbps. Estima-se que o número máximo de atletas para o sistema implementado é de 50 atletas. No caso de pretender-se monitorizar maior número de atletas, é sempre possível embeber o algoritmo do cálculo da velocidade no microcontrolador e este devolve apenas o resultado, libertando 200 bytes por atleta por segundo. Ao custo de perder a informação da aceleração é possível aumentar o valor máximo de atletas para cerca de 250. Para confirmar este valor é necessário, num trabalho futuro, proceder à realização de testes com vários nós, de forma a confirmar o desempenho da rede com o aumento de nós de monitorização de atletas, particularmente porque com o aumento de mensagens de dados, aumenta o número de mensagens de controlo do protocolo.

Do teste realizado verificou-se a robustez da rede, pois nenhuma mensagem foi perdida. Por segurança, visto que os tipos de mensagens são numerados, adicionou-se uma verificação de sequência de dados na tarefa de recepção no computador. Se detectar-se a falta de uma mensagem, a tarefa completa o período da mensagem em falta com o valor anterior. Mais uma vez a rede XBee demonstrou ser uma rede robusta e adequada para a aplicação.

No sentido de aproveitar a tecnologia de rede *Bluetooth*, integrou-se um rádio *Bluetooth* ao módulo de monitorizarão ambiental para oferecer mais uma possibilidade de interface aos dados monitorizados. O sistema foi programado para o telemóvel procurar os nós disponíveis com *Bluetooth* na proximidade e aceder aos dados de monitorização automaticamente. Para fins de teste, construiu-se um segundo módulo com *Bluetooth* para simular um segundo nó de monitorização ambiental. Os nós foram colocados à distância de 100 metros no exterior do edifício como esquematizado pela figura 5.14. Com um telemóvel a correr a aplicação desenvolvida, realizou-se passagens na proximidade dos nós a velocidades diferentes e observou-se o funcionamento da aplicação.

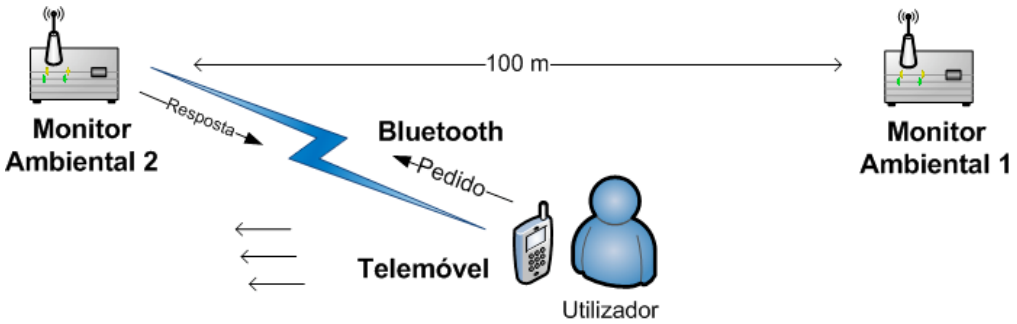


Figura 5.14 – Teste de reação do sistema *Bluetooth*.

Os resultados mostraram que, à velocidade de passo normal, a aplicação alterava a informação de um nó para a do outro em menos de trinta segundos. No caso de velocidades mais elevadas, situação de corrida, a maior parte das vezes a aplicação só captava um dos nós.

Realizou-se outro teste dentro do edifício da Universidade da Madeira, em que um nó foi colocado num laboratório e o outro no laboratório consecutivo, com cerca de 20 m entre nós. Circulando com o telemóvel entre os laboratórios, observou-se que devido a aplicação não realizar qualquer tratamento de *handover*, esta utiliza o primeiro nó que registou mesmo estando junto do segundo nó. A transição só se dá quando perde a recepção do primeiro nó encontrado.

Para finalizar realizou-se uma pequena experiência para testar o acesso concorrente ao módulo. Iniciou-se a aplicação em dois telemóveis diferentes ao mesmo tempo. Ambos conseguiram ligar-se e obter os dados, observando-se que um esperou mais alguns segundos. O próprio protocolo resolve a concorrência de acessos.

É interessante obter os valores dos parâmetros ambientais e os dados da posição dos dois nós sobre o mapa. Esta aplicação apresenta-se como um meio adequado para providenciar outros dados de interesse a outros utilizadores, como *joggers* e turistas.

5.6 Análise do Consumo Energético

Um dos principais objectivos do projecto é o baixo consumo, o que foi conseguido no primeiro protótipo em relação aos sistemas MicaZ e Tmote. Este protótipo apresentou um consumo médio de 2,5 mA a 2.6 V. Dada a limitação do *ZigBee* obrigar a todos os nós rádios estarem activos para o funcionamento da rede em malha, e ainda do funcionamento contínuo

de monitorização da aceleração do atleta, posição GPS e medição do valor de RSSI dos vários sinais de referência, optou-se por não aplicar qualquer técnica de poupança energética aos módulos de monitorização de atletas e nós de referência. Foi no módulo de monitorização ambiental em que se trabalhou no sentido de obter a maior eficiência.

Todos os elementos móveis do sistema, têm como fonte de energia um par de baterias AA. Foi utilizado o modelo da Energizer de 2500 mAh [108]. Os consumos dos vários módulos foram registados, através da inserção de uma resistência de 1Ω, em série, no circuito de alimentação, de modo a obter-se a corrente a partir da leitura da tensão. Com um osciloscópio observou-se a queda de tensão na resistência, que pela lei de Ohm corresponde à corrente consumida. Os consumos foram medidos para um par de baterias em que a carga apresentava uma queda de tensão de 2,6 V.

A figura 5.15 apresenta o consumo registado no nó de monitorização de atletas. Observou-se que o funcionamento dos componentes digitais reflecte um consumo inconstante. Esta oscilação é incrementada pelo funcionamento do conversor DC-DC à entrada do circuito. O valor médio de consumo é da ordem dos 149 mA a 2,6 V. Considerando a carga das baterias de 2500 mAh, estima-se uma duração de quinze horas de trabalho contínuo. Suficiente para vários dias de treino se considerarmos a duração de duas horas por cada treino.

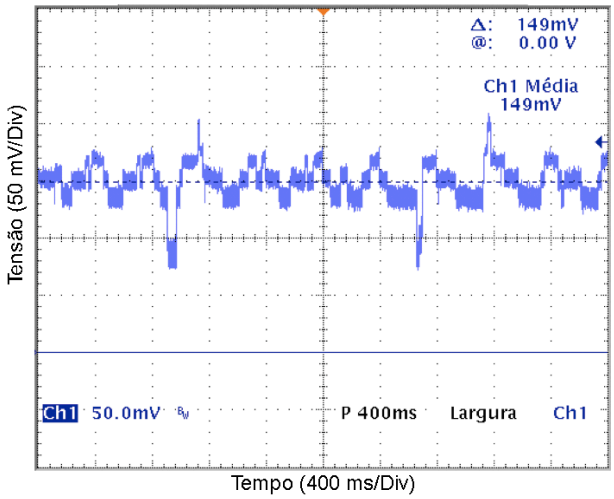


Figura 5.15 – Consumo registado do nó de monitorização de atletas.

O consumo do nó de referência resulta do consumo de três componentes: GPS, XBee e microcontrolador. Mais uma vez, estes componentes estão sempre activos, resultando num consumo de 41 mA, 40 mA e de 6 mA respectivamente. Um total de 87 mA a 3,3 V. Como a tensão da bateria é de 2,6 V e a eficiência do conversor é de cerca de 94%. O consumo antes do conversor sobe para 118 mA a 3,3 V. Estima-se uma duração de 19 horas de trabalho contínuo.

Na figura 5.16 a) apresenta-se o consumo do nó de monitorização ambiental durante um ciclo de monitorização. Este ciclo reflecte o funcionamento do programa de controlo do nó. Assim após um estado de adormecido o consumo aumenta com a activação do sensor de CO_2, gerando um pico de corrente até 270 mA. De seguida segue-se 8 amostras em períodos de meio segundo do sensor de CO_2, destacado pelos impulsos periódicos de consumo mais elevado na figura 5.16 a). Findo este período o sensor de CO_2 é desactivado e o programa passa a realizar as amostras do restantes sensores sequencialmente. No fim do ciclo activo aparece em destaque o consumo do rádio que emite a mensagem com um consumo médio de 40 mA, reflectindo o valor indicado pelo fabricante. A figura 5.16 b) destaca o funcionamento

periódico do conversor DC-DC. Neste caso apresenta-se o consumo no estado adormecido, em que apenas o *Bluetooth* está numa situação de espera de ligação. O módulo apresenta um consumo médio, no estado adormecido, de 3,2 mA.

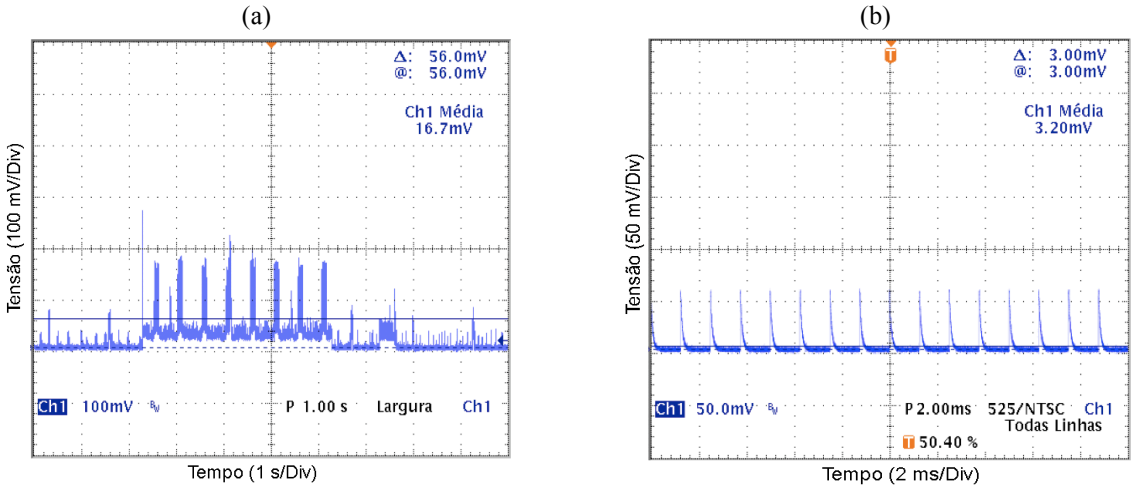


Figura 5.16 – Consumo registado do nó de monitorização ambiental: a) consumo activo; b) consumo adormecido.

O consumo do protótipo de monitorização ambiental final aumentou, imposto pelo consumo do sensor de CO_2 que é mantido ligado durante 4 segundos e pela necessidade de ter o módulo *Bluetooth* sempre ligado. Registou-se um consumo médio da ordem dos 13,5 mA. Para uma carga das baterias obtém-se a duração de sete dias, como se pode verificar pela evolução da descarga da bateria representada na figura 5.17.

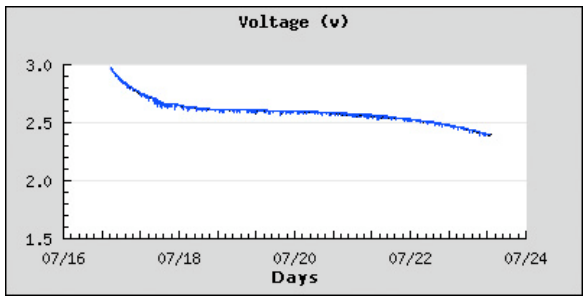


Figura 5.17 – Curva de descarga da tensão das baterias de alimentação.

Da análise do consumo, verificou-se que o sensor de CO_2 é o que mais contribui para o consumo final do nó sensor. Ele consome cerca de 65 mA durante 4 s enquanto o rádio consome 40 mA durante 0,3 s. Isto dá um consumo do sensor de CO_2 21 vezes superior ao do rádio enfatizando a inadequação da eficiência energética da tecnologia sensorial do CO_2. Verifica-se também a importância dos sensores e diversos componentes poderem passar para um estado inactivo de baixo consumo sempre que possível permitindo o controlo inteligente do sistema, aumentando a eficiência energética.

5.7 Características do Sistema

De seguida resume-se as características dos nós de monitorização do sistema.

Na tabela 5.2 apresenta-se o resumo da gama de parâmetros monitorizados pelo nó de atleta.

TABELA 5.2 – GAMA DE PARÂMETROS MONITORIZADOS NO MÓDULO DE ATLETA.

Parâmetro	Precisão	Gama		Unidades
		Mínimo	Máximo	
Posição GPS	2,5 m	-	-	metros
Posição RSSI	±30 %	2	120	metros
Ritmo cardíaco	1	0	255	Batimentos/minuto
Velocidade	±15 %	0	15	m/s
Aceleração	0,001	-6	6	g

O módulo de monitorização de atletas apresenta as características descritas na tabela 5.3, delineadas pelas características dos componentes integrados.

TABELA 5.3 – CARACTERÍSTICAS DO NÓ DE MONITORIZAÇÃO DE ATLETAS.

Parâmetro	Valores	
	Valor	Observação
Consumo de corrente	Médio = 150 mA	@ 2.6 V
Dimensões	144 x 89 x 45mm	Sem caixa
Peso	99 g	54 g são das baterias AA
Tensão de alimentação	1.3 to 3.3V	

Destaca-se a influência das baterias no peso final do nó de monitorização de atleta. Por ser importante não influenciar a actividade desenvolvida pelo atleta, se for necessário pode ser utilizado um par de baterias de formato AAA que reduzem o peso em cerca de 25 g. Neste caso a duração de funcionamento será reduzida para cerca de 6 horas.

Na tabela 5.4 apresenta-se o resumo da gama de parâmetros ambientais monitorizados pelo nó de monitorização ambiental.

TABELA 5.4 – GAMA DE PARÂMETROS AMBIENTAIS MONITORIZADOS.

Parâmetro	Resolução	Gama		Unidades
		Mínimo	Máximo	
Humidade	0,05	0	100	%RH
Temperatura	0,04	-40	123,8	ºC
Luz	15	0	10 000	Lux
CO	0,18	10	180	ppm
CO_2	20	200	650 000	ppm
O_2	1	0	100	%
Alimentação	0,005	0	3,3	V

Devido às características de cada componente o módulo de monitorização ambiental apresenta limitações de condições de funcionamento. O componente que tem a maior limitação é o sensor de oxigénio, limitando as condições de trabalho do nó à gama de temperatura entre os 5ºC e 40ºC, e à gama de humidade dos 10% aos 90% sem condensação. A tabela 5.5 apresenta as características funcionais do módulo.

TABELA 5.5 – CARACTERÍSTICAS DO NÓ DE MONITORIZAÇÃO AMBIENTAL.

Parâmetro	Valores	
	Valor	Observação
Consumo de corrente	Médio = 13,5mA Máximo < 270mA	@ 2.6 V

Parâmetro	Valores	
	Valor	Observação
Gama de humidade	10% a 90%	Limitado pelo sensor O2
Gama de temperatura	5ºC a 40ºC	Limitado pelo sensor de O2
Dimensões	144 x 89 x 45mm	Incluindo caixa
Tensão de alimentação	1.3 to 3.3V	

O sistema permite monitorizar o desempenho de atletas e o ambiente circundante numa prova de educação física durante sete dias, para um treino de duas horas diárias. Findo o qual será necessário recarregar as baterias. Como demonstrado no ponto 5.5, a rede pode monitorizar até um total de cinquenta atletas.

Na figura 5.18 apresenta-se o sistema desenvolvido.

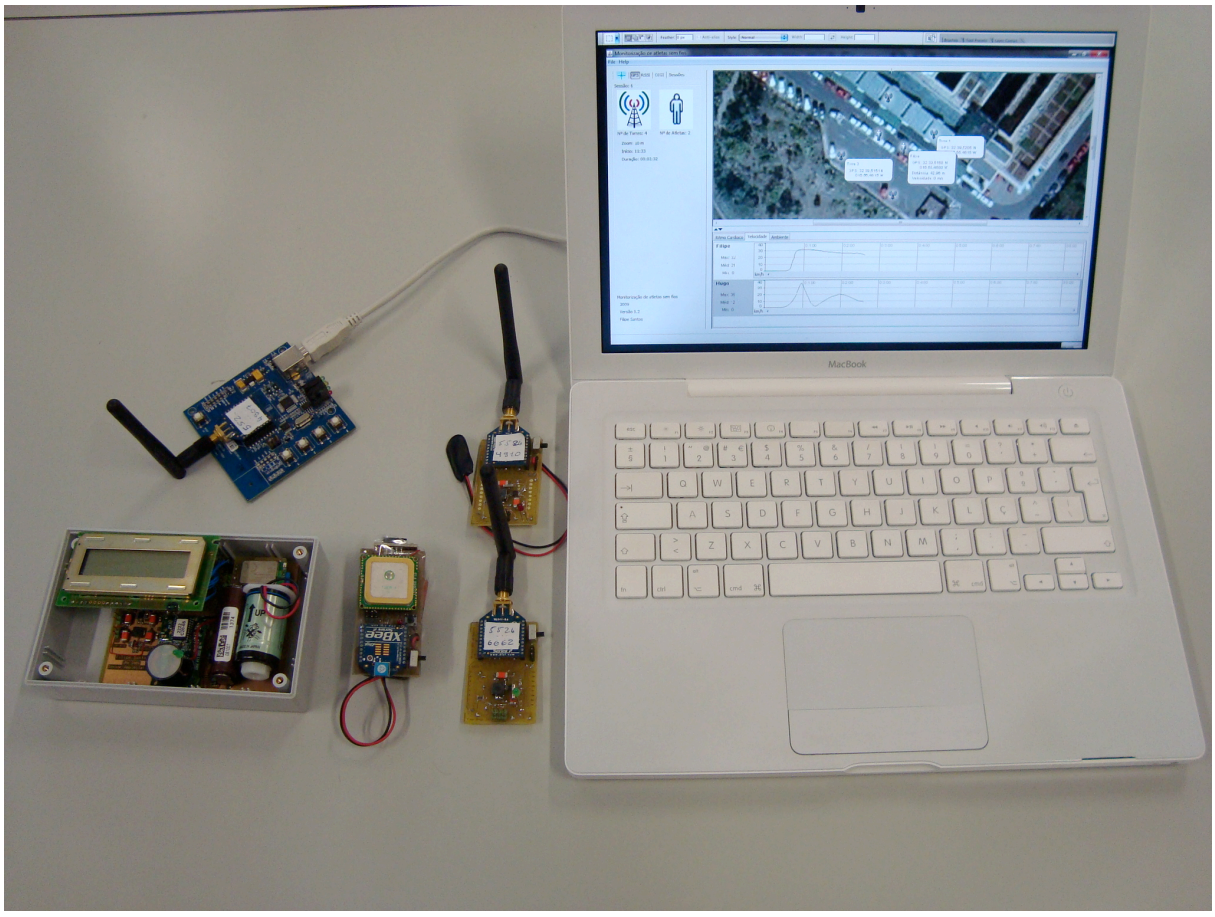


Figura 5.18 – Sistema desenvolvido.

6 CONCLUSÕES E TRABALHO FUTURO

Os objectivos desta tese foram de forma geral atingidos. Desenvolveu-se um protótipo do sistema, completamente funcional, fazendo uso de todos os elementos de *hardware* e *software* mencionados nesta dissertação.

O desenvolvimento deste trabalho permitiu-nos avaliar a importância de uma série de características que são intrínsecas ao desenvolvimento de aplicações baseadas em redes de sensores sem fios, nomeadamente a integração dos vários elementos do sistema, consumos energéticos, controlo do nó, entre outros. Também foram identificados serviços que são de grande importância neste tipo de redes como as possíveis interfaces e serviços de localização. As interfaces implementadas permitem a recolha de dados dos diferentes sensores e traduzi-los em informação útil. Os serviços de localização permitem manter o conhecimento da localização dos diferentes nós dada a sua possibilidade de portabilidade.

Demonstrou-se que com a tecnologia sensorial actual foi possível obter um equipamento de monitorização ambiental e de monitorização de atletas. Embora os fabricantes têm vindo a melhorar a tecnologia de sensores, ainda é uma limitação para a miniaturização dos dispositivos e obtenção duma melhor autonomia. A gestão da energia é um aspecto importante para as RSSF. Implica novos requisitos de *hardware* mas também de concepção do *software* de controlo dos nós do sistema. O nível de inteligência do programa de controlo é determinante para a eficiência energética dos módulos.

As redes baseadas em *ZigBee* permitem ter nós terminais de baixo consumo. Aplicações que requerem mobilidade dos diferentes nós exigem a implementação da topologia de rede mais adequada. A topologia em malha oferece melhor desempenho nestes casos, já que permite o encaminhamento dos dados, por diferentes caminhos mantendo maior robustez do sistema. A obrigatoriedade de ter todos os nós activos para a rede funcionar em malha não permite a aplicação de técnicas de poupança de energia, razão pela qual é um desafio desenvolver novos protocolos para os investigadores da área.

Com base nos resultados obtidos, comprovou-se que a tecnologia de rede *ZigBee* é adequada para a transmissão da informação obtida por cada equipamento da rede de monitorização de atletas.

A integração e compatibilização das várias tecnologias de rede, como a Internet, redes *ZigBee* e *Bluetooth* permitem veicular informação de um meio para outro, facilitando o acesso à informação, permitindo assim, a aproximação em espaço e tempo da informação para a população em geral, caminhando-se, assim, para o paradigma das redes ubíquas.

Por fim, este sistema ao oferecer uma maior disponibilidade dos valores dos parâmetros monitorizados, permite uma melhoria na avaliação da qualidade do ambiente de treino e do desempenho dos atletas. A disponibilidade desta informação permitirá aos treinadores e atletas a introdução de melhoras do plano de treino, com maior regularidade, maximizando o desempenho do atleta e salvaguardando a sua saúde e bem estar.

Obteve-se um sistema com um certo nível de integração de funcionalidades provenientes de diferentes tecnologias. Isto produz uma enorme lista de aspectos a ter em conta. Desde a aplicação correcta de cada tecnologia, normas a respeitar até à apresentação adequada dos resultados. Nesta dissertação identificou-se uma série de melhorias que podem ser aplicadas ao sistema desenvolvido. Destas sublinham-se duas que permitiriam passar para uma fase de testes mais completa: a calibração do sistema e o encapsulamento dos vários módulos. Os fabricantes já disponibilizam em geral os sensores calibrados de fábrica. Para integrar os sensores nos módulos do sistema utilizou-se circuitos de condicionamento de sinal, como amplificadores, conversores ADC, entre outros, que contribuem para um certo grau de incerteza do valor medido. Daí a necessidade de um dos primeiros trabalhos futuros ser a calibração dos vários sensores embebidos no sistema. O sistema numa situação real passará por uma série de condições físicas adversas como chuva, choques, humidade, entre outros justificando a segunda maior necessidade de obter-se o encapsulamento adequado para os diversos módulos do sistema.

Das várias observações efectuadas durante o desenvolvimento resume-se de seguida uma série de melhorias que podem permitir ao sistema aproximar-se de uma solução ideal:

- Aplicar um algoritmo de cálculo de velocidade com maior precisão;
- Utilizar outro rádio ZigBee que responda às seguintes características: menor consumo, possibilidade de implementar a aplicação no microcontrolador (reduzindo o tamanho e custo do sistema), e ainda a possibilidade de realizar medidas de RSSI com maior rapidez e precisão;
- Estudar as necessidades futuras da base de dados e prever o aumento significativo de dados caso o número de nós aumente;
- Estudar junto dos potenciais utilizadores o desenvolvimento da interface;
- Estudar e integrar os sensores de pressão barométrica, poeiras e de ruído no módulo de monitorização ambiental;
- Aplicar fontes de energia renováveis, em particular aos nós de monitorização ambiental;
- Preparar o controlo embebido dos vários nós e a aplicação gestora para alterar os parâmetros do sistema remotamente, como por exemplo: alterar o período de amostragem, actualizar as informações fornecidas aos utilizadores de telemóveis. Este último pode potenciar a utilização do sistema para informar turistas na cidade do Funchal.

Já houve contactos de entidades exteriores à Universidade da Madeira que demonstraram interesse na aplicação do sistema em casos reais. Espera-se em breve que em próximos trabalhos o sistema seja aplicado na monitorização de provas desportivas de orientação, e ainda que os módulos de monitorização ambiental sejam aplicados para avaliação das condições ambientais no meio urbano da cidade do Funchal.

7 REFERÊNCIAS

[1] D. Culler; D. Estrin; e M. Srivastava, "Overview of sensor networks". IEEE Computer, vol. 37(8), pp 41–49, Agosto 2004.

[2] I.F. Akyildiz; S. Weilian; Y. Sankarasubramaniam; e , E. Cayirci, “A survey on sensor networks”. Communications Magazine, IEEE. Agosto 2002.

[3] K. Römer; e F. Mattern. "The Design Space of Wireless Sensor Networks". IEEE Wireless Communications, vol. 11 (6), pp 54–61, doi:10.1109/MWC.2004.1368897, Dezembro 2004.

[4] Abd-El-Barr, M.I.; Youssef, M.A.M.; e Al-Otaibi, M.M. “Wireless Sensor Networks - Part I: Topology and Design Issues”. Electrical and Computer Engineering, Canadian Conference, 2005.

[5] F. Akyildiz; X. Wang; e W. Wang, "Wireless mesh networks: a survey". Computer Networks and ISDN Systems, vol. 47 no.4, pp 445-487, 15 Março 2005.

[6] P. Kinney, "ZigBee Technology: Wireless Control that Simply Works". Communications Design Conference, 2 Outubro 2003.

[7] "The Bluetooth Special Interest Group". [Online]. Disponível em: http://www.bluetooth.com/Bluetooth/Experience/Experience+Icons/. [Acedido: 02 de Novembro de 2009].

[8] I. Mathioudakis; N. M. White; N. R. Harris; e G. V. Merrett, "Wireless Sensor Networks: A Case Study for Energy Efficient Environmental Monitoring". Electronic Systems and Devices Group, School of ECS, University of Southampton, SO17 1BJ, UK.

[9] "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)", IEEE Computer Society, 1 de Outubro 2003.

[10] "ZigBee Specification". Acedido em Novembro de 2008, no *Web site* da ZigBee Aliance: http://www.zigbee.org

[11] J. Yick; B. Mukherjee; e D. Ghosal, "Wireless sensor network survey". Department of Computer Science, University of California, Davis, CA 95616, USA.

[12] "Controlo remoto e aquisição de dados via XBee/ZigBee (IEEE 802.15.4)". Disponível em: RCOM-MESHBEE: http://www.rogercom.com/ZigBee/ZigBee.htm. [Acedido: Agosto de 2008]

[13] "Comparison of 802.15.4 radio modules". [Online]. Disponível em: http://en.wikipedia.org/wiki/Comparison_of_802.15.4_radio_modules. [Acedido: Janeiro de 2009].

[14] "RC220x, datasheet (rev. 1.1). Radiocrafts AS", 2008. [Online]. Disponível em: http://vast.uccs.edu/files/projects/traumagps_files/docs/radiocraft/RC220x_Data_Sheet_1_0.pdf. [Acedido: Dezembro de 2008].

[15] "RC230x, datasheet (rev. 1.2). Radiocrafts AS", 2007. [Online]. Disponível em: http://radiocrafts.com/uploads/rc230x_data_sheet_1_2.pdf. [Acedido: Dezembro de 2008].

[16] "JN5139-xxx-MOyy-PB v1.2, datasheet. Jennic Ltd", 2008. [Online]. Disponível em: http://www.jennic.com/files/product_briefs/JN5139-xxx-Myy-PB_v1.2.pdf. [Acedido: Janeiro de 2008].

[17] "M2110, datasheet. Crossbow Technology Inc.", July, 2007. 7430-0549-01 Rev A. [Online]. Disponível em: http://www.xbow.com/Support/Support_pdf_files/OEM_Design_Reference_Manual.pdf. [Acedido: Dezembro de 2008].

[18] "MICAz Wireless Measurement System, datasheet. Crossbow Technology Inc.", 2005. [Online]. Disponível em: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf. [Acedido: Fevereiro de 2007].

[19] "M-251~01 ZigBit OEM Module, datasheet. Meshnetics". [Online]. Disponível em: http://www.meshnetics.com/netcat_files/Image/M-251~01-%28ZigBit%20OEM%20Module%20Product%20Datasheet%29.pdf. [Acedido: Janeiro de 2008].

[20] "M-251~03 ZigBit Amp OEM Module, datasheet. Meshnetics". [Online]. Disponível em: http://www.meshnetics.com/netcat_files/Image/M-251~03-

%28ZigBit%20Amp%20OEM%20Module%20Rev%202_2%20Product%20Datasheet%29.pdf. [Acedido: Janeiro de 2008].

[21] "M-251~06 ZigBit 900 OEM Module, datasheet. Meshnetics". [Online]. Disponível em: http://www.meshnetics.com/netcat_files/Image/M-251~06-%28ZigBit%20900%20OEM%20Module%20Product%20Datasheet%29.pdf. [Acedido: Janeiro de 2008].

[22] "XBee/XBee-PRO OEM RF Modules, datasheet. Digi International", 2008. 90000982_A. [Online]. Disponível em: http://www.digi.com/register/index.jsp?mu=/pdf/ds_xbeemultipointmodules.pdf [Acedido: Novembro de 2007].

[23] "XBee/XBee-PRO ZB RF Modules, datasheet. Digi International", 2009. 90000976_C. [Online]. Disponível em: http://www.digi.com/register/index.jsp?mu=/pdf/ds_xbeezbmodules.pdf. [Acedido: Novembro de 2007].

[24] J. D. Bronzino, "The Biomedical Engineering Hanbook", Second Edition, CRC Press and IEEE Press, 1995.

[25] A. J. Carlisle; e N. C. C. Sharp, "Exercise and outdoor ambient air pollution". British Journal of Sports Medicine, 01 Agosto de 2001.

[26] R. Maughan; e S. Shirreffs, "Preparing Athletes for Competition in the Heat: Developing an Effective Acclimatization Strategy". Gatorade Sports Science Exchange, vol. 10, 1997.

[27] D. Bailey; e B. Davis, "Physiological implications of altitude training for endurance performance at sea level: a review". British Journal of Sports Medicine, no. 31, pp 183-190, 1997.

[28] "GPSports". [Online]. Disponível em: http://gpsports.com/gpsports_*Web*site/. [Acedido: Agosto de 2009].

[29] "Hosand". [Online]. Disponível em: http://www.hosand.com/. [Acedido: Agosto de 2009].

[30] "Polar". [Online]. Disponível em: http://www.polar.fi/en/. [Acedido: Agosto de 2009].

[31] "Garmin". [Online]. Disponível em: http://www.garmin.com/garmin/cms/site/us. [Acedido: Agosto de 2009].

[32] "EKO Pro Series System. Crossbow. 6020-0135-01 Rev A". [Online]. Disponível em: http://www.xbow.com/Products/productdetails.aspx?sid=284. [Acedido: Fevereiro de 2007].

[33] "Wireless Temperature & Soil Moisture Sensors - HOBOnode". [Online]. Disponível em: http://www.onsetcomp.com/hobonode-wireless-sensor. [Acedido: Maio 2009].

[34] J.W. Kwon; Y.M. Park; S.J. Koo; H. Kim, "Design of Air Pollution Monitoring System Using ZigBee Networks for Ubiquitous-City," International Conference on Convergence Information Technology (ICCIT 2007), pp.1024-1031, 2007.

[35] G. Barrenetxea; F. Ingelrest; G. Schaefer; M. Vetterli; O. Couach; e M. Parlange, "SensorScope: Out-of-the-Box Environmental Monitoring". In Proceedings of the 7th International Conference on Information Processing in Sensor Networks, St. Louis, MO, USA. Acedido em Julho de 2009 no *Web site* da: SensorScope: http://sensorscope.epfl.ch/documents/agu_06_sensorscope.pdf

[36] R.N. Murty; G. Mainland; I. Rose; A.R. Chowdhury; A. Gosain; J. Bers; e M. Welsh, "CitySense: An Urban-Scale Wireless Sensor Network and Testbed". In Proceedings of the 8th IEEE Conference on Technologies for Homeland Security, Waltham, MA, USA. Acedido em Julho de 2009 no *Web site* da: Harvard University, School of Engineering and Applied Sciences: http://www.eecs.harvard.edu/~mdw/talks/citysense-commnet-nov07.pdf

[37] "Biomechanical Analysis", 12th IAAF World Championships in Athletics, Berlin 23/08/2009, Acedido em Julho de 2009, por o *Web site* da International Association of Athletics Federations: http://berlin.iaaf.org/mm/Document/Development/Research/05/31/54/20090817073528_httppostedfile_Analysis100mMenFinal_Bolt_13666.pdf

[38] "GPS LS20031, datasheet. Locosys". [Online]. Disponível em: http://www.sparkfun.com/datasheets/GPS/Modules/LS20030~3_datasheet_v1.0.pdf. [Acedido: Dezembro de 2008].

[39] "Prozone3". [Online]. Disponível em: http://www.prozonesports.com/product-prozone3.html. [Acedido: Janeiro de 2009].

[40] A. Savvides; H. Park; e M. B. Srivastava. "The N-Hop Multilateration Primitive for Node Localization Problems". MONET Special Issue on Sensor Networks and Applications, Junho de 2003.

[41] "Polar S3 Stride Sensor". [Online] Disponível em: http://www.polarusa.com/us-en/products/accessories/s3_Stride_Sensor_WIND. [Acedido: 04-07-09].

[42] "Forerunner 305". [Online] Disponível em: https://buy.garmin.com/shop/shop.do?pID=349&ra=true#specsTab. [Acedido: 05-07-09].

[43] "Omron HJ-112 Digital Pocket Pedometer". [Online] Disponível em: http://www.omronhealthcare.com/product/1131-229-pedometers-gosmart-pocket-pedometer-with-advanced-2d-smart-sensor-technology-hj-112. [Acedido: 04-07-09].

[44] A. P. Morise, "Heart Rate Recovery, Predictor of Risk Today and Target of Therapy Tomorrow?". American Heart Association, Inc., Circulation. 2004; no. 110, pp 2778-2780.

[45] B. H. Brown; R. H. Smallwood; D. C. Barber; P. V. Lawford; e D. R. Hose, "Medical Physics and Biomedical Engineering". Institute of Physics Publishing Bristol and Philadelphia, 1999.

[46] V. K. Jayasree; P. J. Shaija; V. P. N. Nampoori; C. P. Girijavallabhan; P. Radhakrishnan, "A Simple and Novel Integrated Opto-Electronic System for Blood Volume Pulse Sensing and Heart Rate Monitoring". International Journal of Optomechatronics, Volume 1, Issue 4, pp 392 – 403, Outubro de 2007.

[47] "ePulse". [Online]. Disponível em: http://www.impactsports.com/technology.php. [Acedido: Agosto de 2009].

[48] "Nissei". [Online]. Disponível em: http://www.nissei-kk.co.jp/html/hr40.html. [Acedido: Agosto de 2009].

[49] "Mio". Disponível em: http://www.miowatch.com/. [Acedido: Agosto de 2009].

[50] P. T. Abreu, "Teoria Digital da Cor". IADE, 2008.

[51] J. J. Cannell; B. W. Hollis; M. B. Sorenson; T. N. Taft; e J. J. B. Anderson, "Athletic Performance and Vitamin D". Medicine & Science in Sports & Exercise, Volume 41, Issue 5, pp 1102-1110, Maio 2009.

[52] "MOPITT - Measurements of Pollution in the Troposphere. NASA". [Online]. Disponível em: http://eos*Web*.larc.nasa.gov/PRODOCS/mopitt/table_mopitt.html. [Acedido: Fevereiro de 2009].

[53] J. A. Raub; e V. A. Benignus, "Carbon Monoxide and the Nervous System". Neuroscience and Biobehavioral Reviews, vol. 26(8), pp 925-940, 2002.

[54] T. Gosink, "What Do Carbon Monoxide Levels Mean?". Alaska Science Fórum, no. 588, 28 de Janeiro de 1983. Acedido em Fevereiro de 2009 por o *Web site* da: University of Alaska, Geophysical Institute: http://www.gi.alaska.edu/ScienceForum/ASF5/588.html

[55] "Europe – Air Quality", Acedido em Março de 2009, por o *Web site* da Comissão Europeia: http://ec.europa.eu/environment/air/quality/standards.htm

[56] "Questions and Answers on the "HELP – For a life without tobacco". Carbon Monoxide Campaign", MEMO/06/147 Bruxelas, 29 de Março de 2006. Acedido em Março de 2009 por o *Web site* da União Europeia: http://europa.eu/rapid/pressReleasesAction.do?reference=MEMO/06/147&format=HTML&aged=1&language=EN&guiLanguage=fr

[57] J. T. Houghton; Y. Ding; D.J. Griggs; M. Noguer; P. J. van der Linden; X. Dai; K. Maskell; e C.A. Johnson, "Climate Change 2001: The Scientific Basis". A Contribution of Working Group I to the Third Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, pp 881, 2001.

[58] "C20 Sensor Application Testing". Gas Sensing Solutions [E-mail]. Requerido em: http://www.gassensing.co.uk/. [Em: Janeiro de 2009].

[59] D. S. Robertson, "Health effects of increase in concentration of carbon dioxide in the atmosphere", Current Science, vol. 90, no. 12, 25 de Junho de 2006.

[60] "OSHA Permit-Required Confined Spaces Standard", (29 CFR 1910.146). [Online]. Disponível em: http://www.des.umd.edu/os/csp/oshacss.html. [Acedido: Março de 2009].

[61] W. D. McArdle; F. I. Katch; e V. L. Katch, "Essentials of exercise physiology".

[62] "Gas Sensing Solutions NDIR Carbon Dioxide Sensor C20, Background to Non-Dispersive InfraRed Gas Detection of Carbon Dioxide". Gas Sensing Solutions [E-mail]. Requerido em: http://www.gassensing.co.uk/. [Em: Janeiro de 2009].

[63] "Single Mode Optical Fibre Sensor Technology. Lecture Notes". University of Kent e Sira Ltd., Kent – Inglaterra: Three-day Workshop organized jointly by the Optics Groups of the University of Kent and Sira Ltd. 1985.

[64] W. J. Woodfin, "Portable Electrochemical Sensor Methods". NIOSH/DPSE.

[65] R. F. Taylor; e J. S. Schultz, "Handbook of chemical and biological sensors".

[66] P. Patnaik, "Handbook of Environmental Analysis: Chemical Pollutants in Air, Water, Soil, and Solid Wastes", CRC Press, pp. 65 – 67, 1997. ISBN 0873719891, 9780873719896.

[67] "Piezoelectric sensors". [Online] Disponível em: http://www.piezocryst.com/piezoelectric_sensors.php. [Acedido: 2 de Outubro de 2009]

[68] Sistemas de Última Generación para la Observación, Predicción y Vigilancia Activa de Espacios Naturales Forestales en la Macaronesia FORESMAC (INTERREG III B, 05/MAC/2.3/C16) Madeira.

[69] "Tmote Sky: Ultra Low Power IEEE 802.15.4 Compliant Wireless Sensor Module". 2006. [Online] Disponível em: http://www.moteiv.com/products/docs/tmote-skydatasheet.pdf. [Acedido: Junho de 2007].

[70] P. Levis; S. Madden; J. Polastre; R. Szewczyk; K. Whitehouse; A. Woo; D. Gay; J. Hill; M. Welsh; E. Brewer; e D. Culler. "TinyOS: An Operating System for Sensor Networks, from Ambient Intelligence", W. *Web*er et al, Ed.

[71] "Foresmac Demonstrator, Manual do programa". Edosoft Factory. 2008.

[72] "Matlab v6.1". [Software]. Licença: Universidade da Madeira.

[73] "AT90USB1287, datasheet. ATMEL". [Online] Disponível em: http://www.atmel.com/dyn/resources/prod_documents/doc7593.pdf. [Acedido: Dezembro de 2007].

[74] C. A. Balanis, "Antenna theory, Analysis and Design". Second edition, John Wiley & Sons, Inc. ISBN 0-471-59268-4.

[75] J. A. Azevedo, "Apontamentos da disciplina: Radiação e Propagação. Linhas de transmissão". Universidade da Madeira, 2008.

[76] J. A. Azevedo, "Apontamentos da disciplina: Antenas". Universidade da Madeira, 2008.

[77] J. B. Andersen; T. S. Rappaport; e S. Yoshida, "Propagation Measurements and Models for Wireless Communications Channels", IEEE Communications Magazine, vol. 33, pp. 42-49, 1995.

[78] K. Hobbs, "The real issue limiting wireless sensor networks 16 May 2007, data acquisition product manager". National Instruments, 16 de Maio de 2007.

[79] D. Strassberg, "Simple networks will free many sensors from wires". Contributing Technical Editor, EDN, 13 de Abril de 2006.

[80] "SCA3000 Accelerometer in speed, distance and energy measurement, Application Note 50". VTI technologies. [Online]. Disponível em: http://www.vti.fi/midcom-serveattachmentguid-9cdff49ade3611ddb72c3b87f201104c104c/an50_sca3000_accelerometer_in_velocity_distance_and_energya-s.pdf. [Acedido: Novembro de 2009].

[81] A. Goetsch, "The Evolution of GPS". Illumin, Issue iii, vol. 10. [Online]. Disponível em: http://illumin.usc.edu/article.print.php?articleID=137. [Acedido: Junho de 2009].

[82] "Multisim v2001". [Software]. Licença da Universidade da Madeira

[83] "Eagle v5.6.0 Light Edition for windows". [Software]. Licença *freeware for "non-profit"*. Disponível em: http://www.cadsoft.de/freeware.htm.

[84] "AVR Studio v4.14". [Software]. Licença *freeware*. Disponível em: http://www.atmel.com.

[85] "Win AVR v1.6.2". [Software]. Licença GNU Disponível em: http://winavr.sourceforge.net/.

[86] "FLIP – Flexible In-system Programmer v3.2.0. Disponível em: http://www.atmel.com.

[87] "AVR ISP500 – Tiny V2 datasheet". [Online]. Disponível em: http://www.olimex.com/dev/pdf/AVR/AVR-ISP500-TINY.pdf. [Acedido: Outubro de 2008].

[88] "X-CTU". [Software]. Licença *freeware*. Disponível em: http://www.digi.com.

[89] "Netbeans v6.7". Licença *freeware*. Disponível em: http://www.netbeans.com.

[90] "XAMPP v1.7.2 for Windows". Licença GNU. Disponível em: http://www.apachefriends.org.

[91] "PHP Designer 2007 v5.0.2". Licença *personal free version*. Disponível em: http://www.mpsoftware.dk/phpdesigner.php.

[92] "HD44780U LCD Library". [Software]. Licença GNU. Baseado em Volker Oth's LCD Library (http://members.xoom.com/volkeroth). Modificado por P. Fleury (http://jump.to/fleury). Configurado por M. Ermert. Adaptado para linuxfocus LCD display por G. Socher. Adaptado para ser usado com stdio.h por F. Gunawan utizando LCG_PUTC_STEAM.

[93] "JpGraph under QPL 1.0". [Software]. Licença *freeware.* Disponível em: http://www.aditus.nu/jpgraph.

[94] "mysql4.php v1.5". [Sodtware]. Licença GPL. Disponível em: http://www.phpbb.com.

[95] "SUN Microsystems JDBC driver for MySQL v5.1.7". [Software]. Licença GNU. Disponível em: http://dev.mysql.com/downloads/connector/j/5.1.html.

[96] "RXTXCOMM v2.1-7r2 Native interface to serial ports in Java". [Software]. Licença LGPL v2.1. Disponível em: http://users.frii.com/jarvi/rxtx/.

[97] "Bluecove version 2.1.0". [Software]. Licença GNU. Disponível em: http://bluecove.org.

[98] "TGS4160 datasheet. Figaro Engineering Inc". [Online]. Disponível em: http://www.figarosensor.com/products/4160pdf.pdf. [Acedido: Dezembro de 2008].

[99] "C20 user Instructions". Gas Sensing Solutions. Junho de 2008. [E-mail]. Requerido em: http://www.gassensing.co.uk/. [Em: Janeiro de 2009].

[100] "TGS5042, datasheet. Figaro Engineering Inc". [Online]. Disponível em: http://www.figarosensor.com/products/5042pdf.pdf. [Acedido: Dezembro de 2008].

[101] "KE-25/KE-50, datasheet. Figaro Engineering Inc". [Online]. Disponível em: http://www.figarosensor.com/products/O2.pdf. [Acedido: Dezembro de 2008].

[102] "SHT1x, datasheet. Sensirion". [Online]. Disponível em: http://www.parallax.com/dl/docs/prod/datast/shtx.pdf. [Acedido: Julho de 2008].

[103] "S1087/S1033 Si Photodiode, datasheet. Hamamatsu ". [Online]. Disponível em: http://sales.hamamatsu.com/assets/pdf/parts_S/S1087_etc.pdf. [Acedido: Junho de 2008].

[104] A. R. Z. Nascimento; J. C. J. de Almeida; E. C. Ferreira; O. V. A Filho; e A. L .P. Mattei. "Circuitos Amplificadores de Transimpedância Integrados a Fotodíodos". Universidade Estadual, de Campinas (UNICAMP), Departamento de Electrónica e Microelectrónica Campinas, Instituto de Estudos Avançados (IEAv), CTA São José dos Campos, São Paulo, Brasil.

[105] "BISM Bluetooth, datasheet. Ezurio". [Online]. Disponível em: http://www.ezurio.com/files/00482.pdf. [Acedido: Dezembro de 2008].

[106] "WT12, datasheet. Bluegiga". [Online]. Disponível em: http://www.bluegiga.com/files/bluegiga/Pub%20files/WT12_Datasheet.pdf. [Acedido: Dezembro de 2008].

[107] "ATmega324, datasheet. ATMEL". [Online]. Disponível em: http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf. [Acedido: Fevereiro de 2009].

[108] "ENERGIZER NO. NH15, AA NiMH Rechargeable Battery Datasheet". Energizer. [Online]. Em: http://www.energizer.com. [Acedido: Março de 2009].

[109] "MAX1675, datasheet. Maxim". Disponível em: http://datasheets.maxim-ic.com/en/ds/MAX1674-MAX1676.pdf. [Acedido: Dezembro de 2008].

[110] GPS EM-406, datasheet. GlobalSat. Disponível em: http://www.sparkfun.com/datasheets/GPS/EM-406%20Product_Guide1.pdf. [Acedido: Maio de 2009].

[111] Roberto Fernandes, "Monitorização de provas de educação física, Relatório intermédio de projecto de mestrado". Universidade da Madeira. Janeiro de 2008.

[112] "T31 Transmitter, product details. Polar". [Online]. Disponível em: http://www.polarusa.com/us-en/products/accessories/T31_transmitter. [Acedido: Junho de 2008].

[113] "RMCM-01 Heart Rate Receiver Component, specifications. Polar". [Online]. Disponível em: http://www.sparkfun.com/datasheets/Wireless/General/RMCM01.pdf. [Acedido: Maio de 2008].

[114] "ADXL330, datasheet. Analog Devices". [Online]. Disponível em: http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf. [Acedido: Julho de 2008].

[115] "LIS3LV02DQ, datasheet, STMicroelectronics". [Online]. Disponível em: http://www.st.com/stonline/books/pdf/docs/11115.pdf. [Acedido: Agosto de 2008].

[116] Jerald G. Graeme , "Photodiode amplifiers : op amp solutions". - New York [u.a.] : McGraw Hill, 1996.

[117] S. Mason; e E. Korolev, "Native and Java ME Development on Symbian OS". Symbian Developer Network Version: 1.0, Março de 2008.

[118] "Foto por satélite da Universidade da Madeira". [Online]. Disponível em: http://maps.google.com. [Acedido: Julho de 2009].

[119] "GPS Latitude and Longitude Distance Calculator". [Online]. Disponível em: http://www.csgnetwork.com/gpsdistcalc.html. [Acedido: Maio de 2009].

[120] "CC2431, datasheet. Texas Instruments". [Online]. Disponível em: http://focus.ti.com/lit/ds/symlink/cc2431.pdf. [Acedido: Junho de 2009].

[121] J. Scarlett, "Enhancing the Performance of Pedometers Using a Single Accelerometer", Aplication Note 900, Analog Devices.

[122] M. Petrova; J. Riihijarvi; P. Mahonen; e S. Labella, "Performance study of IEEE 802.15.4 using measurements and simulations". IEEE Wireless Communications and Networking Conference, pp. 487-492, Abril 2006.

[123] C. Chen; e C. Pomalaza-Ráez, "Monitoring Human Movements at Home Using Wearable Wireless Sensors". Indiana University - Purdue University, Fort Wayne, Indiana, USA. Acedido em Setembro de 2009 no *Web site* da: IEEE 802 LAN/MAN Standards Committee: http://www.ieee802.org/15/pub/03/15-03-0305-00-0040-zigbee-tutorial.ppt.

ANEXOS

ANEXO A - Estrutura API Específica do XBee

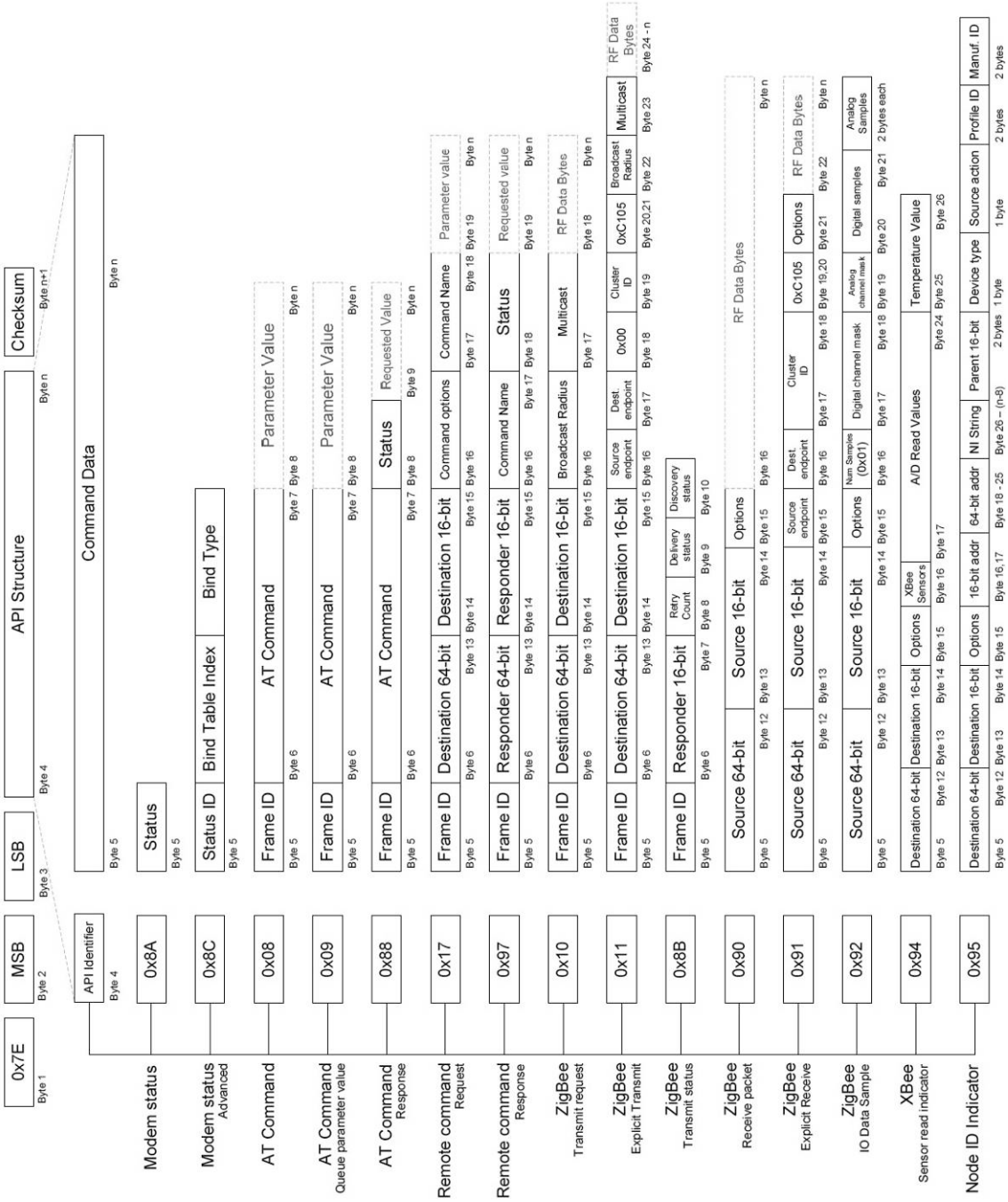


Figura A.1 - Resumo da estrutura API do XBee .

Referência: "Využití bezdrátové komunikace v malých medicínských (bateriově napájených) elektronických zařízeních", Vít Prajzler. 2008.

ANEXO B – Características e notas de utilização do XBEE

Constituição do *Kit*

Para montar uma rede 802.15.4/*ZigBee* com XBee basta ter 2 módulos.

No Kit de desenvolvimento comercializado pela DIGI são fornecidos:

- Vários módulos RF XBee (com diferentes características);
- Placas de desenvolvimento;
- E o software X-CTU (gratuito).

As placas de desenvolvimento permitem não só interligar o XBee ao computador como também actualizar o seu *firmware* através do programa X-CTU fornecido em conjunto.

Especificações

O que se destaca entre as duas versões de hardware é a potência de emissão e a sensibilidade de recepção. Na tabela 3.1 apresenta-se as características detalhadas das várias versões [26].

Tabela B.1 – Especificações dos módulos XBee

Parâmetros	***XBee (Pro)***	***XBee série 2 (Pro)***
Firmware	802.15.4 / DigiMESH 2.4	Znet 2.5 / ZB
Fabricante rádio	Freescale	Ember 250
Alcance RF Interior / urbano	até 30 m	até 40 m
Alcance RF Exterior	até 100 m	até 120 m
Potência de transmissão	0 dBm (+18 dBm)	+3 dBm (+10 dBm)
Taxa de transmissão	250,000 bps	
Sensibilidade de recepção	-92 dBm (-100 dBm)	-96 dBm (-102 dBm)
Tensão de alimentação	2.8 – 3.4 V	2.8 - 3.3 V
Corrente de transmissão	45 mA @ 3.3 V (250 mA)	40 mA (170 mA)
Corrente de recepção	50 mA @ 3.3 V (55 mA)	40 mA (45 mA)
Corrente de adormecido	< 10 µA	< 1 µA
Frequência	ISM 2.4 GHz	
Dimensões	0.960” x 1.087”	
Temperatura de funcionamento	-40 to 85º C	
Opções de antenas	*Chip*, *Integrated Whip*, U.fl, RPSMA	
Topologias de rede suportadas	*P2P, P2M, PP* e *MESH*	*P2P, P2M,* e *MESH*
Número de canais	16 *Direct Sequence Channels*	
Opções de filtro	PAN ID, Canal e Remetente/Destino	
ZigBee	-	*2006+ / PRO*
FCC Part 15.247	OUR-XBee	OUR-XBee2
Industry Canada (IC)	4214A-XBee	4214A-XBee2
Europe (CE)	ETSI	ETSI

Firmware AT e API

O fabricante já fornece o *firmware* para instalar nos módulos XBee, os diferentes *firmware's* permitem controlar o funcionamento do módulo, em diferentes variáveis. São fornecidos dois *firmware's*: ZIGBEE e ZNET 2.5, este último *stack* semelhante em muito ao

standard *ZigBee* de 2006, mas com funcionalidades adicionadas. Estes *firmware's* são subdivididos em *Coordinator* e *Router/End Device* apresentando diferentes variáveis de controlo conforma a funcionalidade. (Na figura 11.10 é mostrado o ecrã do programa de instalação de *firmware* com os parâmetros do *firmware* ZNET 2.5).

É através da UART que podemos enviar/receber dados e executar comandos.

Em cada versão de *firmware* existem dois modos de funcionamento modo transparente AT e modo API:

- **Modo tranparente AT** – modo de funcionamento "transparente", que permite o envio e recepção de dados como se de uma extensão de comunicação série se tratasse. Dentro deste modo destaca-se os seguintes comportamentos:
    - *Idle mode* – Quando não está nem a receber ou enviar dados
    - *Transmit mode* – Quando estão disponíveis dados no buffer de recepção série
    - *Receive mode* – Quando é recebido dados válidos pela antena
    - *Sleep mode* – Entra em modo de repouso de baixo consumo (apenas para End Devices);
    - *Command mode AT* – em que comunicamos com o próprio XBee através de comandos;
- ***API mode*** – *Aplication Programming Interface*, neste modo os dados são enviados em forma *frames* bem estruturadas com comandos, endereçamento e informação de estado.

Endereçamento

Cada nó é identificado por dois endereços, um número único de 64 bit gravado pelo fabricante e outro de 16 bit que pode ser alterado pelo utilizador.

Em todas as versões de *firmware* o endereçamento é definido pelas seguintes variáveis:

- SL, SH – número de série fixo que identifica o nó
- MY – endereço de 16 bit configurado localmente
- DH, DL – endereço de destino de 64 bit
    - *Broadcast* se for configurado para #FFFF
    - Envio apenas para a base se for configurado com o respectivo endereço ou com #0000
- PAN ID – Número de identificação de 8 bit do grupo a que pertence
    - *Broadcast* se for configurado para #FF

Transmissão e recepção de dados

No modo AT, podemos comunicar entre dois ou mais dispositivos via UART através de dois ou mais módulos XBee, respectivamente, muito facilmente. È necessário apenas ligar correctamente as UART's dos dispositivos (por exemplo microcontroladores) às dos módulos XBee.

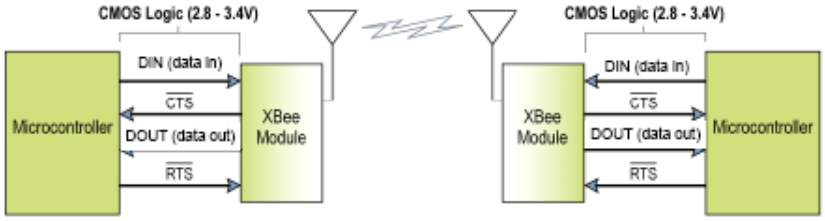


Figura B.1 – Interface UART do XBee

É necessário utilizar a versão AT de um dos *firmware's*, definir os destinatários de cada módulo e utilizar a mesma configuração da UART entre os dispositivos e os módulos XBee (bit de paridade, Stop bit, etc.).

Modo de comandos AT

Modo em que falamos directamente com o módulo através da UART.

É necessário instalar a versão AT de um dos *firmware*'s disponíveis.

Para entrar no modo comandos AT, a partir do Idle mode enviar os a sequência de 3 caracteres "+++", dando início ao modo de comandos AT que termina ao fim de 10 segundos de inactividade. De seguida executar os comandos AT pretendidos. Exemplos:

- AT -> OK
- ATMY -> o endereço próprio
- ATDH, ATDL -> endereço de destino
- ATID -> PAN ID
- ATDB -> RSSI da última mensagem recebida
- ATCN -> fim do modo de comandos

A sintaxe para enviar comandos AT é apresentada na figura B.2.

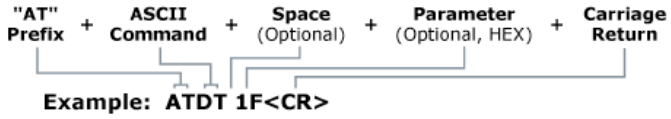


Figura B.2 – Sintaxe para o envio de comandos AT

Nota: Para ler o valor do parâmetro no registo do XBee basta omitir o valor no comando.

È possível executar comandos por linha ou múltiplos comandos como exemplificado na figura B.3.

Method 1 (One line per command)

Send AT Command	System Response
+++	OK <CR> (Enter into Command Mode)
ATDL <Enter>	{current value} <CR> (Read Destination Address Low)
ATDL1A0D <Enter>	OK <CR> (Modify Destination Address Low)
ATWR <Enter>	OK <CR> (Write to non-volatile memory)
ATCN <Enter>	OK <CR> (Exit Command Mode)

Method 2 (Multiple commands on one line)

Send AT Command	System Response
+++	OK <CR> (Enter into Command Mode)
ATDL <Enter>	{current value} <CR> (Read Destination Address Low)
ATDL1A0D,WR,CN <Enter>	OK, OK, OK <CR> (Command execution is triggered upon each instance of the comma)

Figura B.3 – Sequencia de execução de comandos AT

A resposta devolvida pelo XBee é composta por valores hexadecimais que correspondem a caracteres da tabela ASCII.

Modo API

Ao contrário dos modos anteriores para comunicarmos pelo modo API é necessário conhecer a estrutura do modo API. Pois os dados são enviados empacotados em mensagens que contêm comandos e informação de estado. Neste modo as mensagens na UART terão de ter o formato apresentado na figura B.4.

Figura B.4 – Estrutura da frame de dados UART

Quaisquer dados recebidos antes do delimitador de início é descartado silenciosamente. Se a frame não for recebida correctamente ou a verificação de checksum falhar, o módulo responderá com uma frame de estado indicado a natureza do erro.

Na figura B.5 é mostrado o conteúdo completo da frame para a mudança de um parâmetro no próprio módulo XBee.

Figura B.5 – Exemplo: Frame API para o valor do parâmetro NJ do módulo XBee

Legenda:

* Lenght – o comprimento da mensagem é dado pela soma do número de Bytes dos campos entre o API Identifier, inclusive, e o Checksum exclusive, neste caso o comprimento é dado por: Lenght[Bytes] = API Identifier + Frame ID + AT Command + Parameter Value;

** Frame ID – é usualmente usado para numerar sequencialmente as frames (neste exemplo o valor 0x4D foi escolhido arbitrariamente).

O campo API Identifier – identifica o tipo de dados que é transportado na mensagem, definindo também qual a estrutura a usar. No Anexo A é apresentado as estruturas API específicas do XBee.

Para as frames de transmissão e recepção de dados, as estruturas são as apresentadas nas figuras B.6 e B.7 respectivamente.

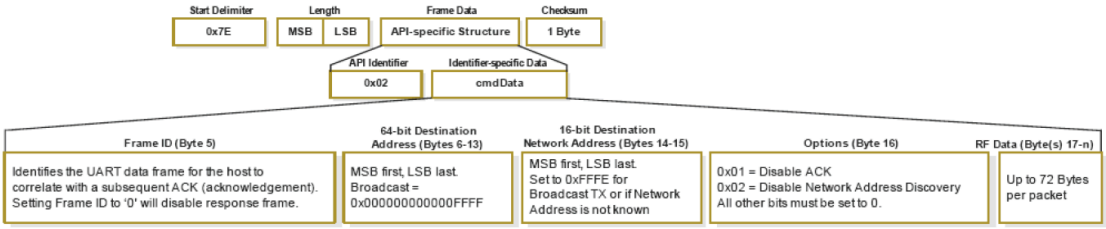


Figura B.6 – Frame de um pacote de transmissão (TX)

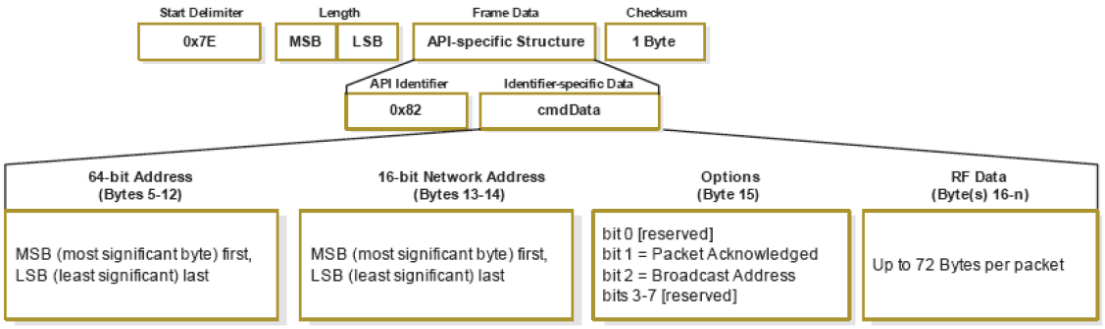


Figura B.7 – Frame de um pacote de recepção (RX)

Notas de implementação

Para a instalação de *firmware* num módulo XBee é necessário utilizar hardware específico ou uma das placas de desenvolvimento fornecidas pela DIGI.

- Conectar o módulo XBee à placa de interface (figura B.8);
- Ligar a placa de interface ao computador;

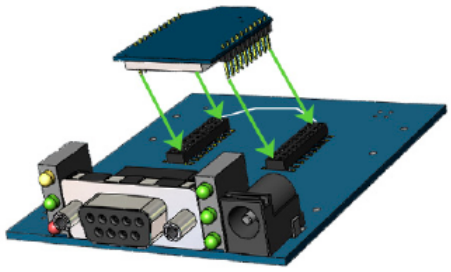


Figura B.8 – Placa de desenvolvimento para XBee com interface RS-232

- Executar o software X-CTU fornecido pela DIGI;
- Escolher no primeiro ecrã a porta COM a que a placa de interface está ligada (no botão Test / Query, testar se o módulo está bem inserido);
- Seleccionar a janela "*Modem Configuration*" (figura B.9);

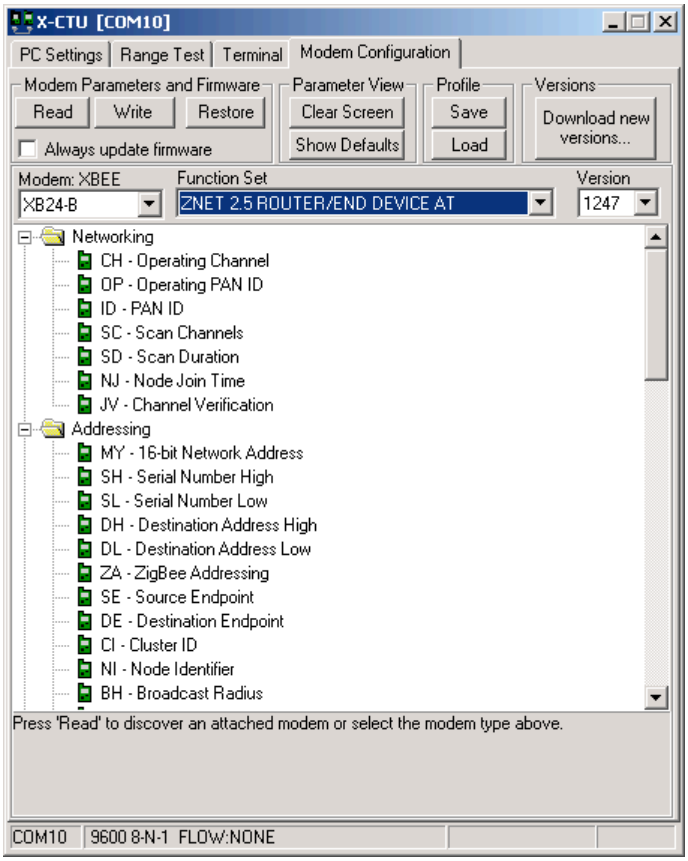


Figura B.9 – Ecrã do software de instalação de *firmware*

- Clicar em "*Read*" para ler a *firmware* actual do módulo XBee;
- Escolher a *firmware* pretendida em "*Function Set*", e definir os parâmetros nas variáveis respectivas conforme funcionamento pretendido do nó;
- Gravar clicando em "*Write*".

NOTA: Se houver dificuldades em reconhecer o nó ou em gravar a nova *firmware*, verificar três pontos:

- Conexão física entre o nó e a placa de desenvolvimento;
- Verificar o modo de funcionamento do módulo XBee com o modo escolhido na primeira janela do software X-CTU (Enable API se o módulo estiver em modo API);
- Forçar a instalação do novo *firmware* escolhendo a opção "*Always update firmware*" (verificar se o fimware escolhido é compatível com a versão do módulo XBee).

Utilizar uma alimentação entre 3.0V e 3.4V – preferencialmente 3.3V estabilizados, ter atenção em usar condensadores de *decoupling*, pois usualmente os rádios não funcionam sem eles.

O módulo XBee apresenta um espaçamento entre pinos de 2mm, e é preferível adaptar para a medida standard de 2.4mm aquando trabalhando no protótipo. São comercializadas placas adaptadoras.

Para interligar o XBee a um microcontrolador via UART basta ligar os pinos da alimentação e ligar o pino TX e RX do XBee aos pinos RX e TX do microcontrolador.

Notas:

Escolher, sempre que possível, componentes para a aplicação que funcionem entre os 3V a 3.3V para evitar mais electrónica de conversão DC-DC.

Para receber os dados no computador da rede XBee sem as placas de desenvolvimento é necessário o interface eléctrico dos níveis de tensão TTL aos dos níveis da porta RS-232 do computador. Ou utilizar um conversor série TTL para USB.

ANEXO C – Características de sensores

Curva de compensação em temperatura do sensor de CO TGS5042.

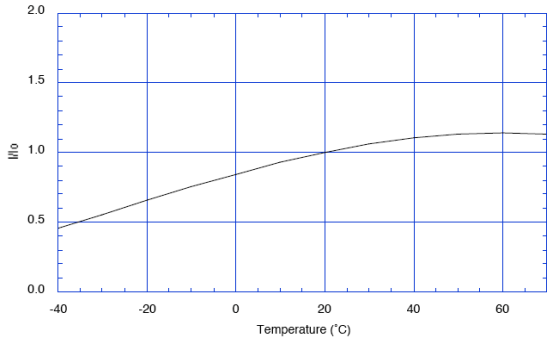


Figura C.1 – TGS5042, compensação de temperatura [35].

Recta de conversão do sensor de oxigénio KE-25.

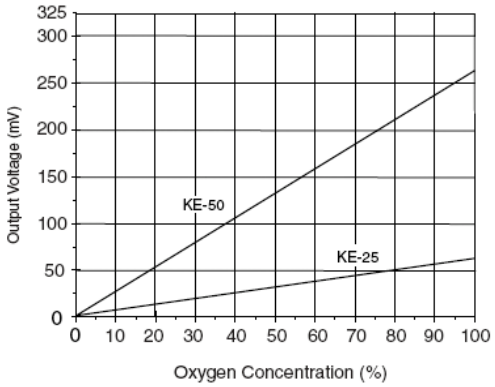


Figura C.2 – KE-25, recta de conversão [35].

Gama de medida e recta de conversão dos fotodíodos S1087 da Hamamatsu.

(a) (b)

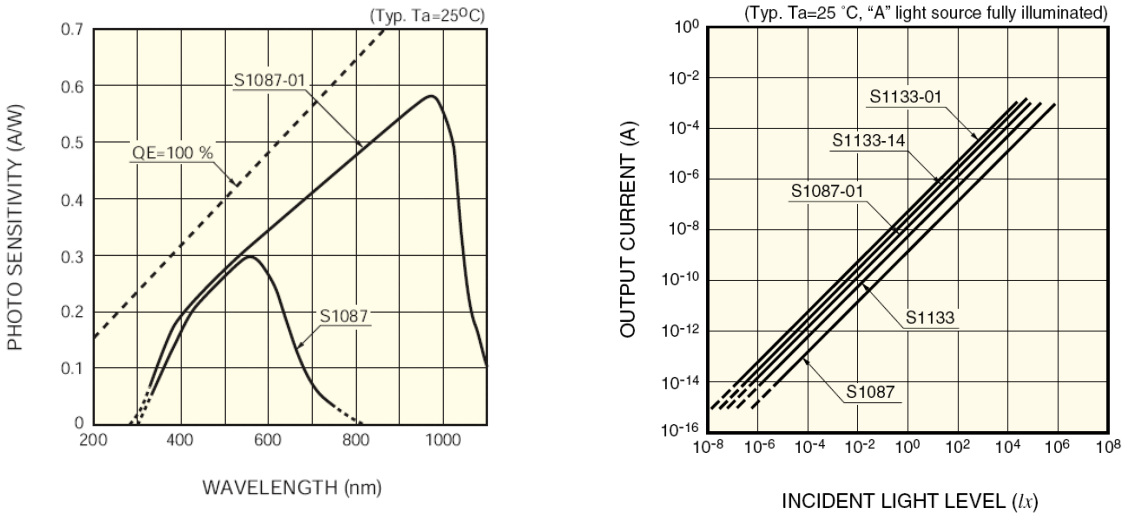


Figura C.3 – Características dos fotodíodos S1087: a) Gama de medida; b) Recta de conversão [32].

ANEXO D – Esquemáticos eléctricos e desenhos dos circuitos impressos dos módulos do sistema

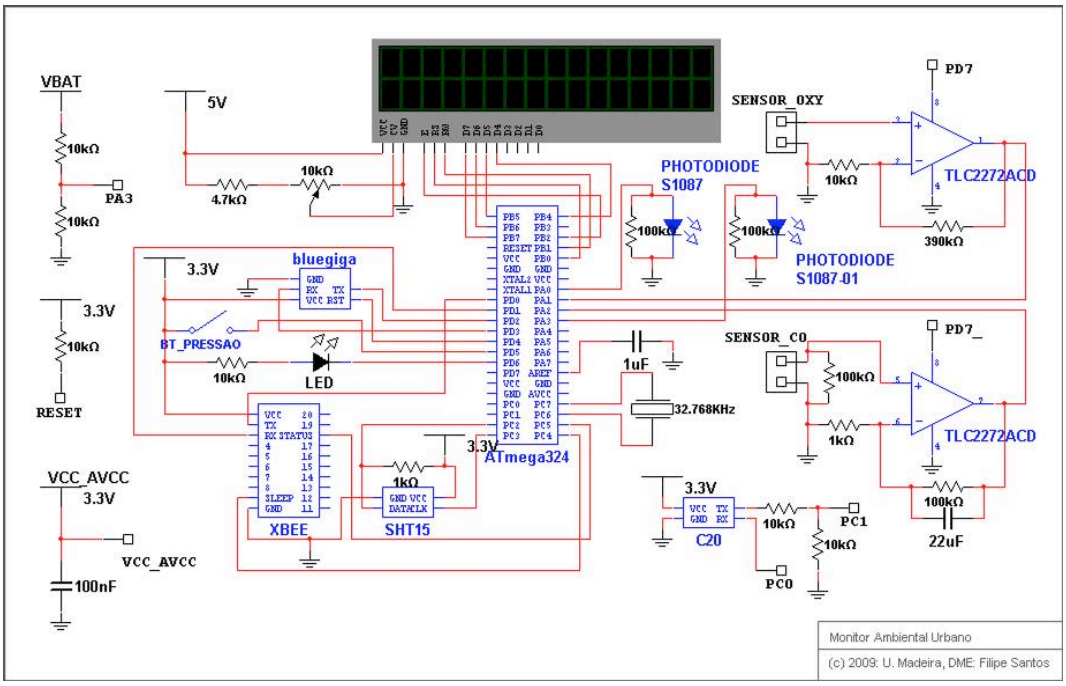


Figura D.1 – Esquemático eléctrico do módulo de monitorização ambiental.

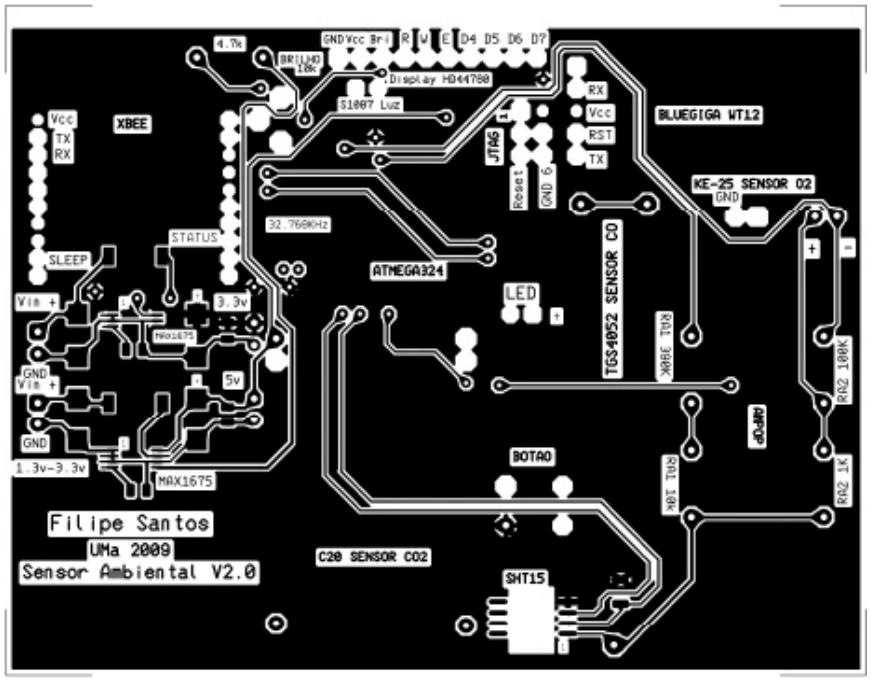


Figura D.2 – Circuito impresso do módulo de monitorização ambiental (topo).

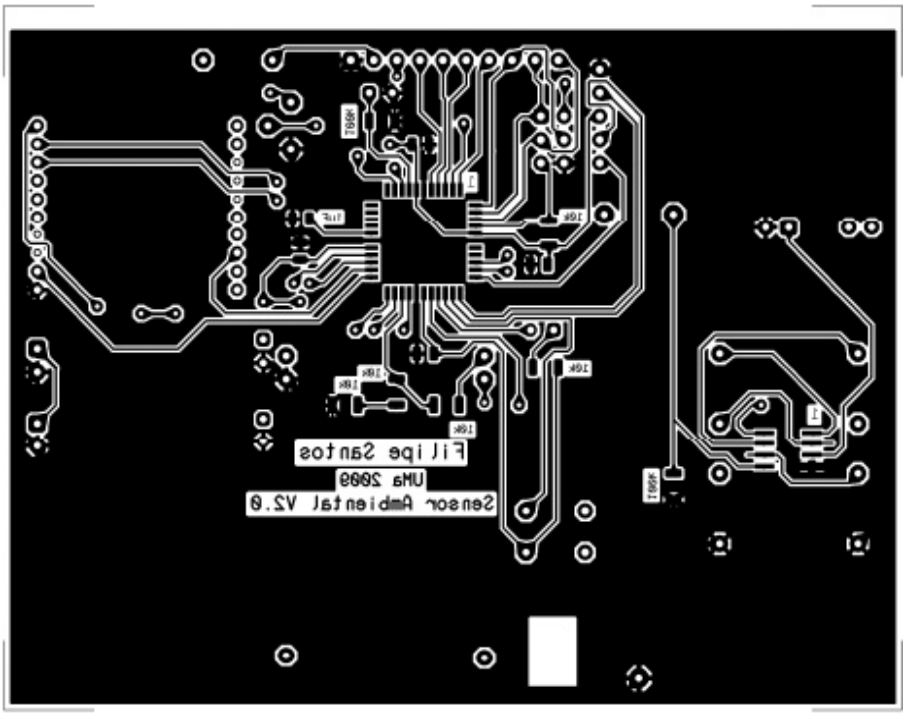


Figura D.3 – Circuito impresso do módulo de monitorização ambiental (fundo).

ANEXO E – Base de dados do sistema

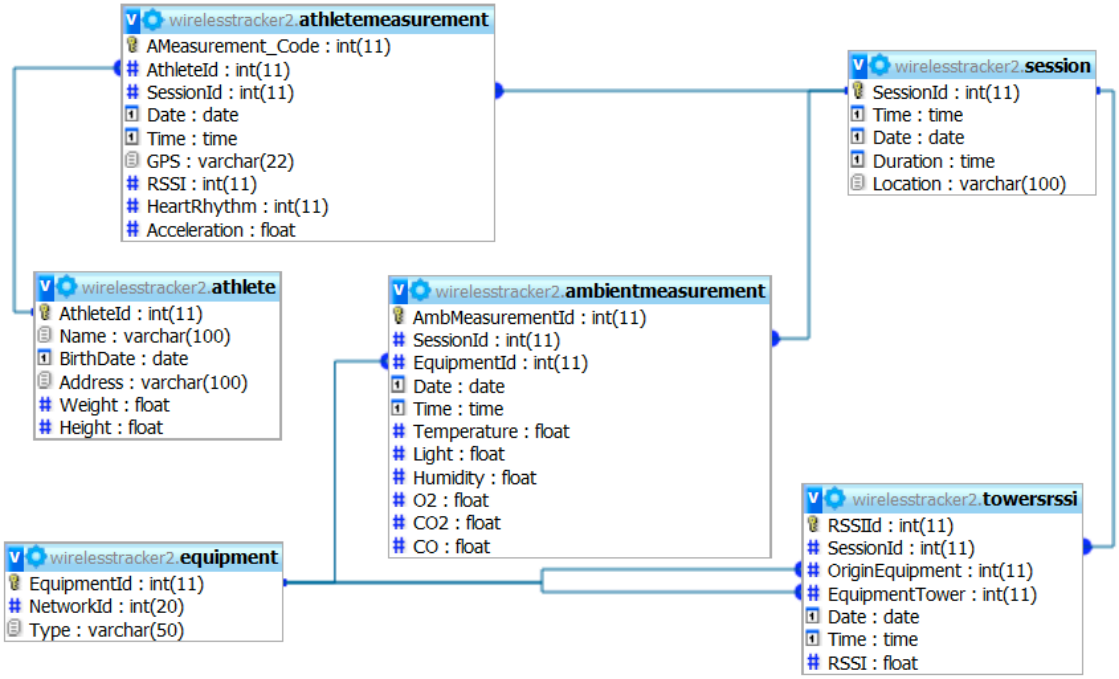


Figura E.1 – Modelo relacional da base de dados.

ANEXO F – Classes principais da aplicação Java

Índice

CircleUtils.class

```
import java.awt.Point;
import java.util.Vector;

class CircleUtils {
    private final static float INC = 0.30f;

    public synchronized static Point circleIntersecionSolver(Point[] center, double[]
radius, Point currentPos) throws Exception {
        Vector<Point> points = new Vector<Point>();
        double[][] results;
        // check input
        if (center.length != radius.length) {
            throw new Exception("The number of centers doesn't match the number of
radius.");
        } else if (center.length == 1 || radius.length == 1 || center == null || radius
== null) {
            throw new Exception("Not enough information to calculate intersection
points.");
        } else {
            for (int i = 0; i < center.length - 1; i++) {
                for (int j = i + 1; j < center.length; j++) {
                    try {
                        results = compute(center[i], (int) (radius[i]), center[j], (int)
(radius[j]));
                        points.add(new Point((int) results[0][0], (int) results[0][1]));
                        points.add(new Point((int) results[1][0], (int) results[1][1]));
                    } catch (Exception e) {
                    }
                }
            }
        }
        if (points.size() == 1) {
            return points.firstElement();
        } else if (points.size() == 2) {
            if (points.firstElement().x > points.lastElement().x &&
points.firstElement().y < points.lastElement().y)
                return points.lastElement();
            else if (points.firstElement().x < points.lastElement().x &&
points.firstElement().y > points.lastElement().y)
                return points.firstElement();
            else if (points.firstElement().x >= points.lastElement().x &&
points.firstElement().y >= points.lastElement().y)
                return points.firstElement();
            else return points.lastElement();
        } else if (points.size() == 0) {
            return null;
        } else {
            int xx = 0, yy = 0;
            for (Point p : points) {
                xx += p.x;
                yy += p.y;
            }
            return new Point(xx / points.size(), yy / points.size());
        }
    }

    private synchronized static double[][] compute(Point c1, int r1, Point c2, int r2)
throws Exception {
        // Check that there is intersection betwen the tow circles
            // Centers of both circles
        int x1 = c1.x, y1 = c1.y, x2 = c2.x, y2 = c2.y;
            // Calculates distance betwen enters
        double d = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
        if (d > (r1 + r2)) {
            r1 *= (1 + INC);
            r2 *= (1 + INC);
            if (d > (r1 + r2)) {
                throw new Exception("No intersection, too far apart.");
            }
        } else if (d < Math.abs(r1 - r2)) {
```

```
            if (r1 < r2){
                r1 *= (1+INC);
            } else if(r1 > r2){
                r2 *= (1+INC);
            }
            if (d < Math.abs(r1 - r2)) {
                throw new Exception("No intersection, one circle within another.");
            }
        }
        // If they intersect Calculates the intesection point(s)
        double a = (r1 * r1 - r2 * r2 + d * d) / (2 * d),
                // h is a common leg for two right triangles.
                h = Math.sqrt(r1 * r1 - a * a),
                // locate midpoint between intersections along line of centers
                P0x = x1 + a * (x2 - x1) / d,
                P0y = y1 + a * (y2 - y1) / d,
                // extend to intersection 1 from midpoint
                P1x = P0x + h * (y2 - y1) / d,
                P1y = P0y - h * (x2 - x1) / d,
                // extend to intersection 2 from midpoint
                P2x = P0x - h * (y2 - y1) / d,
                P2y = P0y + h * (x2 - x1) / d;
        double[][] results = new double[2][2];
        results[0][0] = P1x;
        results[0][1] = P1y;
        results[1][0] = P2x;
        results[1][1] = P2y;
        return results;
}
```

PositioningThread.class

```
import data.*;
import java.awt.Point;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JPanel;

public class PositioningThread extends Thread {
    private GeneralMem mem;
    private Vector<String> towersPositioned;
    private JPanel track;

    public PositioningThread(GeneralMem m, JPanel t) {
        mem = m;
        towersPositioned = new Vector<String>();
        track = t;
        this.start();
    }

    public void calcGPSPosition(Transmiter t) {
        try{
            // Gets the last GPS measurement form the transmiter
            GPSDataType gps = t.gps.getLastLocation();
            // Checks if it has moved
            if (t.hasMoved(gps)){
                // If it did then calculates the current distance (x,y) to the PC
                GPSDataType pc =
mem.getFixed().get(""+mem.getGeneral().get("coordinatorId")).gps.getLastLocation();
                double x = GPSUtils.calculateLatDist(pc.getLat(), gps.getLat());
                double y = GPSUtils.calculateLonDist(pc.getLon(), gps.getLon());
                t.setPos((int)x, (int)y, gps);
                track.repaint();
            }
        } catch (Exception e){
        }
    }

    public void calcRSSIPosition(Transmiter t, Vector<DistanceDataType> dist) throws
Exception{
        if (!dist.isEmpty()) {
```

```
                // Assembles the centers and the radius for the calculus
                Point[] centers = new Point[dist.size()];
                double[] radius = new double[dist.size()];
                for (int i = 0; i < dist.size(); i++) {
                    centers[i] =
mem.getFixed().get(dist.get(i).getTowerId()).getPosition();
                    radius[i] = dist.get(i).getDistance();
                }
                // Calculates the position of this tower
                t.setPos(CircleUtils.circleIntersecionSolver(centers, radius,
t.getPosition()));
                track.repaint();
        }
    }

    public void calcTowerRSSIPosition(Fixed f) {
        // If is the second tower
        if (towersPositioned.size() == 1 || towersPositioned.get(1).equals(f.getId())) {
if(towersPositioned.size() > 1)
System.out.println("tp(1): "+towersPositioned.get(1) + " == "+ f.getId());
            // Gets the towers and distances from the last 60s
            Vector<DistanceDataType> dist = f.distances.getDistances(towersPositioned,
mem.getGeneral().get("coordinatorId")+"", 60);
            // Calculates the position of this tower
            if (!dist.isEmpty()) {
                try {
f.setPos(mem.getFixed().get(dist.firstElement().getTowerId()).getX() + (int)
dist.firstElement().getDistance(), 0);
                    track.repaint();
                    if (!towersPositioned.contains(f.getId()))
towersPositioned.add(f.getId());
                } catch (Exception ex) {
Logger.getLogger(PositioningThread.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        } // If is over three towers
        else if (towersPositioned.size() > 1){
            Vector<String> ids = new Vector<String>();
            if (towersPositioned.contains(f.getId())){
                for (int i = 0; i < towersPositioned.indexOf(f.getId()); i++)
                    ids.add(towersPositioned.get(i));
            } else ids = towersPositioned;
            // Gets the towers and distances from the last 60s
            Vector<DistanceDataType> dist = f.distances.getDistances(towersPositioned,
ids, 60);
                try{
                    calcRSSIPosition(f, dist);
                    if (!towersPositioned.contains(f.getId()))
towersPositioned.add(f.getId());
                } catch (Exception ex) {
                }
        }
    }

    public void calcAthleteRSSIPosition(Mobile m) {
         // If there are over two towers
        if (towersPositioned.size() > 2){
            // Gets the towers and distances from the last 60s
            Vector<DistanceDataType> dist = m.distances.getDistances(towersPositioned,
3, 60);
                try {
                    calcRSSIPosition(m, dist);
                } catch (Exception ex) {
                }
        }
    }

    public void clearTowersPositioned(){
        towersPositioned.setSize(1);
    }

    @Override
    public void run() {
```

```
        short time = 9;
        // Positiones the first tower
        while(true){
            try {
                PositioningThread.sleep(1000);
                // The first tower to be positioned is the coordinator (near the PC)
                Fixed f = mem.getFixed().get(""+mem.getGeneral().get("coordinatorId"));
                if (mem.getGeneral().get("gps").equals(true))
                    f.setPos(0, 0, f.gps.getLastLocation());
                else if (mem.getGeneral().get("rssi").equals(true))
                    f.setPos(0, 0);
                towersPositioned.add(""+f.getId());
                break;
            } catch (Exception ex) {
            }
        }

        while (true) {
            try {
                PositioningThread.sleep(1000);
                if (++time == 10) {
                    // If 10s time has elapsed
                    time = 1;
                    // Calculates the positions of the towers
                    if (mem.getGeneral().get("gps").equals(true)){
                        // If is to use GPS coordenates to calculate position
                        for (Fixed f : mem.getFixed().getAll()) {
                            if
(!f.getId().equals(mem.getGeneral().get("coordinatorId"))){
                                try{
                                    if (f.hasMoved(f.gps.getLastLocation()))
calcGPSPosition(f);
                                } catch (Exception e){}
                            } else
                                try{
                                    f.setPos(0, 0, f.gps.getLastLocation());
                                } catch (Exception e){}
                        }
                    } else if (mem.getGeneral().get("rssi").equals(true)){
                        // If is to use RSSI coordenates to calculate position
                        for (Fixed f : mem.getFixed().getAll()) {
                            if (!towersPositioned.isEmpty() &&
!f.getId().equals(mem.getGeneral().get("coordinatorId"))){
                                calcTowerRSSIPosition(f);
                            } else if
(f.getId().equals(mem.getGeneral().get("coordinatorId"))) f.clearGPSPositioned();
                        }
                    }
                }
                // Calculates the positions of the athletes
                if (mem.getGeneral().get("gps").equals(true)){
                    for (Mobile m : mem.getMobile().getAll()) {
                        try{
                            if (m.hasMoved(m.gps.getLastLocation())) calcGPSPosition(m);
                        } catch (Exception e){}
                    }
                }
                else if (mem.getGeneral().get("rssi").equals(true)){
                    for (Mobile m : mem.getMobile().getAll()) {
                        calcAthleteRSSIPosition(m);
                    }
                }
            } catch (InterruptedException ex) {}
        }
    }
}
```

GPSUtils.class

```
public class GPSUtils {
    private final static float EARTH_RADIUS = 6378136.245f;
```

```
    public static double degToRad(double d){
        // Converts an angle from degrees to radians
        return d * Math.PI / 180;
    }

    public static double calculateLatDist(float lat1, float lat2){
        // Calculates the distance between tow latitudes
        return degToRad(lat1-lat2)*EARTH_RADIUS;
    }

    public static double calculateLonDist(float lon1, float lon2){
        // Calculates the distance between tow longitudes
        return degToRad(lon1-lon2)*EARTH_RADIUS;
    }

    public static double calculateDistance(float lat1, float lon1, float lat2, float
lon2){
        // Calculates the distance between tow GPS coordinates
        double latDist = calculateLatDist(lat1, lat2), lonDist = calculateLonDist(lon1,
lon2);
            // Uses the Pythagoras Theorem to calculate the distance
        double d = Math.pow(latDist, 2) + Math.pow(lonDist, 2);
        return Math.sqrt(d);
    }

    public static float[] toDegrees(String[] gps){
        // Converts a GPS position from dddmm.mmmm to degrees
        float[] location = new float[2];
        String aux;
        // Latitude
        location[0] = (int)(Float.parseFloat(gps[0])/100);
        aux = gps[0].substring(gps[0].length()-2);
        aux += "." + gps[1];
        location[0] += Float.parseFloat(aux)/60;
        if (gps[2].equals("S")) location[0] *= -1;
        // Longitude
        location[1] = (int)(Float.parseFloat(gps[3])/100);
        aux = gps[3].substring(gps[3].length()-2);
        aux += "." + gps[4];
        location[1] += Float.parseFloat(aux)/60;
        if (gps[5].equals("W")) location[1] *= -1;
        return location;
    }
```

ComReader.class

```
import data.MySQL;
import java.io.*;
import java.util.*;
import gnu.io.*;

public class ComReader implements Runnable, SerialPortEventListener {

    private final static String DB = "wirelesstracker";
    InputStream inputStream;
    OutputStream outputStream;
    SerialPort serialPort;
    Thread readThread;
    Message msg;
    WelcomeMsg wmsg;
    int msg_len;
    int msg_type;
    MySQL conect;
    int broadcast;
    private int sessionId;
    private Vector<AccDataType> accs = new Vector<AccDataType>();

    public ComReader(SerialPort sp, int sessionId) {
        this.sessionId = sessionId;
```

```
        try {
            serialPort = sp;
            inputStream = sp.getInputStream();
            outputStream = sp.getOutputStream();
            serialPort.addEventListener(this);
        } catch (TooManyListenersException e) {
            System.out.println(e + " - " + e.getCause() + " - " +
e.getStackTrace());
        } catch (IOException e) {
            System.out.println(e + " - " + e.getCause() + " - " +
e.getStackTrace());
        }
        readThread = new Thread(this);
        msg = new Message();
        wmsg = new WelcomeMsg();
        conect = new MySQL();
        if (conect.Connect() == true) {
            System.out.println("Ligado ao servidor de BD.");
        } else {
            System.out.println("Erro ao ligar a base de dados.");
        }
        if (conect.EnterDatabase() == false) {
            System.out.println("ERRO: Nao foi possivel ligar-se a Base de
Dados");
            return;
        }

        msg_len = 0;
        msg_type = 0;
        broadcast = 0;
        readThread.start();
    }

    public void Stop() {
        System.out.println("ComReader Stoped!!!");
        serialPort.close();
    }

    public void run() {
        while(true){
            try {
                Thread.sleep(100);
                if (broadcast++ == 80)
                    throw new Exception ("Broadcast Exception");
            } catch (InterruptedException e) {
            } catch (Exception e) {
                try{
                    if (e.getMessage().equals("Broadcast Exception")){
                        broadcast = 0;
                        int[] b = {0x7E, 00, 0x10, 0x10, 00, 00, 00, 00, 00,
00, 00, 0xFF, 0xFF, 0xFF, 0xFE, 0x01, 00, 0xB1, 0x01, 0x41};
                        for (int i = 0; i < b.length; i++) {
                            outputStream.write(b[i]);
                        }
                        outputStream.flush();
                    }
                } catch (IOException ioe) {
                    System.out.println("IOex: " + ioe + " - " +
ioe.getCause());
                }
            }
        }
    }

    public void serialEvent(SerialPortEvent event) {
```

```
        switch (event.getEventType()) {
            case SerialPortEvent.BI:
            case SerialPortEvent.OE:
            case SerialPortEvent.PE:
            case SerialPortEvent.CD:
            case SerialPortEvent.CTS:
            case SerialPortEvent.DSR:
            case SerialPortEvent.RI:
            case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                    break;
            case SerialPortEvent.DATA_AVAILABLE:
                try {
                    while (inputStream.available() > 0) {
                        int data = inputStream.read();
                        if (data != 0x7E || msg_len > 0
                        { // Begining of a new message
                            if (msg_len == 0) {
                                msg_len += data;
                            } else if (msg_type == 0) {
                                msg_type = data;
                                if (msg_type == 0x95) {
                                    // Connected Successfuly
                                    wmsg.Process(0);
                                    wmsg.Process(msg_len);
                                    wmsg.Process(msg_type);
                                } else if (msg_type == 0x90) {
                                    // Data Message
                                    msg.Process(0);
                                    msg.Process(msg_len);
                                    msg.Process(msg_type);
                                }
                            } else if (msg_type == 0x95) {
                                // Welcome Message
                                wmsg.Process(data);
                                if (wmsg.IsComplete() == true) {
                                    wmsg = new WelcomeMsg();
                                    msg_len = 0;
                                    msg_type = 0;
                                }
                            } else if (msg_type == 0x90) {
                                // Data Mesage
                                msg.Process(data);
                                if (msg.IsComplete() == true) {
                                    if (msg.GetData()[0] == 0xA0){
                                        msg = new AthleteData(msg);
                                    }
                                    else if (msg.GetData()[0] > 0xA0 &&
msg.GetData()[0] < 0xA4){
                                        msg = new AccelerometerData(msg);
                                    }
                                    else if (msg.GetData()[0] == 0xB0){
                                        msg = new TowerData(msg);
                                    }
                                    else if (msg.GetData()[0] == 0xC0){
                                        msg = new AmbientData(msg);
                                    }

                                    System.out.println(msg.toString());
                                    try {
                                            // Adds the iformation to DataBase
                                            msg.AddToDatabase(conect,
sessionId, accs);
                                    } catch (Exception ex) {}
                                    msg = new Message();
                                    msg_len = 0;
```

```
                                    msg_type = 0;
                                }
                            }
                        }
                    }
                } catch (IOException e) {
                    System.out.println("IOex: " + e + " - " + e.getCause());
                } catch (Exception e) {
                    System.out.println("Ex: " + e + " - " + e.getCause());
                }
                break;
        }
    }
}
```

Message.class

```
import data.MySQL;
import java.util.Vector;

public class Message {

    int LENGTH; // 2 bytes
    int CODE; // 1 byte
    long SIXTY_FOUR_BITADDRESS; // 8 bytes
    int SIXTEEN_BITADDRESS; // 2 bytes
    int OPTION; // 1 byte
    int[] DATA; // LENGTH bytes
    int CHECK_SUM; // 1 byte
    int indice;

    public Message() {
        indice = 1;
        LENGTH = 0;
    }

    public Message(Message msg){
        this.LENGTH = msg.LENGTH;
        this.CODE = msg.CODE;
        this.SIXTY_FOUR_BITADDRESS = msg.SIXTY_FOUR_BITADDRESS;
        this.SIXTEEN_BITADDRESS = msg.SIXTEEN_BITADDRESS;
        this.OPTION = msg.OPTION;
        this.DATA = msg.DATA;
        this.CHECK_SUM = msg.CHECK_SUM;
        this.indice = msg.indice;
    }

    public void Process(int info) {
        switch (indice) {
            case 1:
                LENGTH = info << 8;
                break;
            case 2:
                LENGTH += info;
                break;
            case 3:
                CODE = info;
                break;
            case 4:
                SIXTY_FOUR_BITADDRESS = (long) info << 56;
                break;
            case 5:
                SIXTY_FOUR_BITADDRESS += (long) info << 48;
                break;
```

```
            case 6:
                SIXTY_FOUR_BITADDRESS += (long) info << 40;
                break;
            case 7:
                SIXTY_FOUR_BITADDRESS += (long) info << 32;
                break;
            case 8:
                SIXTY_FOUR_BITADDRESS += info << 24;
                break;
            case 9:
                SIXTY_FOUR_BITADDRESS += info << 16;
                break;
            case 10:
                SIXTY_FOUR_BITADDRESS += info << 8;
                break;
            case 11:
                SIXTY_FOUR_BITADDRESS += info;
                break;
            case 12:
                SIXTEEN_BITADDRESS = info << 8;
                break;
            case 13:
                SIXTEEN_BITADDRESS += info;
                break;
            case 14:
                OPTION = info;
                break;
            case 15:
                if (LENGTH >= 13) // 13 is the 1st data element
{
                    DATA = new int[LENGTH - 12];
                }
            default:
                if (indice == (LENGTH + 3)) {
                    CHECK_SUM = info;
                } else {
                    if ((DATA != null) && ((indice - 15) < DATA.length)) {
                        DATA[(indice - 15)] = info;
                    }
                }
        }
        indice += 1;
    }

    public int GetDataLength() {
        return DATA.length;
    }

    public int GetCode() {
        return CODE;
    }

    public long Get64BitAddress() {
        return SIXTY_FOUR_BITADDRESS;
    }

    public int Get16BitAddress() {
        return SIXTEEN_BITADDRESS;
    }

    public int GetOptions() {
        return OPTION;
    }

    public boolean CheckSum() {
```

```
        // Incomplete
        return false;
    }

    public int[] GetData() {
        return DATA;
    }

    public boolean IsComplete() {
        return (indice > (LENGTH + 3)); // 2 bytes for Length + 1 for checksum
    }
    public void AddToDatabase(MySQL db, int sessionId, Vector<AccDataType>
acc) throws Exception{}

    protected String athleteExists(MySQL db, int sessionId, long address64)
throws Exception{
        // Checks if the athlete already exists for this session and if it
does not, creates it
        String sql = "SELECT athleteId FROM athlete" +
             " WHERE 64address = '" + address64 + "'" +
             " AND sessionId = '" + sessionId + "'";
        String[] ident, vals;
        Vector<Vector<String>> result = db.query(sql);
        if (result == null){
            // Adds the athlete to the database
            ident = new String[3];
            ident[0] = "64address";
            ident[1] = "sessionId";
            ident[2] = "dateTime";
            vals = new String[3];
            vals[0] = ""+address64;
            vals[1] = ""+sessionId;
            vals[2] = ""+DateUtils.GetCurrentDateTime();
            if (db.InsertTableValue("athlete", ident, vals)){
                result = db.query(sql);
            }
            else throw new Exception("Failed to insert the athlete " +
address64 + " into the DB");
        }
        try{
            return result.firstElement().firstElement();
        }
        catch (Exception e){
            throw e;
        }
    }

    protected String towerExists(MySQL db, int sessionId, long address64)
throws Exception{
        // Checks if the tower already exists for this session and if it does
not, crates it
        String sql = "SELECT towerId FROM tower" +
             " WHERE 64address = '" + address64 + "'" +
             " AND sessionId = '" + sessionId + "'";
        Vector<Vector<String>> result = db.query(sql);
        if (result == null){
            // Adds the tower to the database
            String[] ident, vals;
            ident = new String[3];
            ident[0] = "64address";
            ident[1] = "sessionId";
            ident[2] = "dateTime";
            vals = new String[3];
            vals[0] = ""+address64;
            vals[1] = ""+sessionId;
```

```
            vals[2] = ""+DateUtils.GetCurrentDateTime();
            if (db.InsertTableValue("tower", ident, vals)){
                result = db.query(sql);
            }
        }
        try{
            return result.firstElement().firstElement();
        }
        catch (Exception e){
            throw e;
        }
    }
}
```

AthleteData.class

```
import data.MySQL;
import java.text.DecimalFormat;
import java.util.Vector;


public class AthleteData extends commonData {

    public AthleteData(Message msg) {
        super(msg);
    }

    public int getRhythm() {
        return this.DATA[11];
    }

    @Override
    public void AddToDatabase(MySQL db, int sessionId, Vector<AccDataType>
accs) throws Exception {
        // Chcks if the athlete already exists and if it does not, creates it
        try {
            String athleteId = this.athleteExists(db, sessionId,
this.Get64BitAddress());
            Vector<String> ident = new Vector<String>();
            Vector<String> vals = new Vector<String>();
            String dateTime = DateUtils.GetCurrentDateTime();
            String[] ids, vls;
            // Iserts the athlets data
            ident.add("athleteId");
            vals.add(""+athleteId);
            if (((this.getLatL() != 0 || this.getLatR() != 0) &&
                 (this.getNS() == 'N' || this.getNS() == 'S') &&
                ((this.getLonL() != 0 || this.getLonR() != 0) &&
                 (this.getEW() != 'E' || this.getEW() != 'W')))){
                ident.add("latL");
                vals.add("" + this.getLatL());
                ident.add("latR");
                vals.add("" + this.getLatR());
                ident.add("NS");
                vals.add("" + this.getNS());
                ident.add("lonL");
                vals.add("" + this.getLonL());
                ident.add("lonR");
                vals.add("" + this.getLonR());
                ident.add("EW");
                vals.add("" + this.getEW());
            }
            ident.add("rhythm");
            vals.add(""+this.getRhythm());
```

```
            ident.add("dateTime");
            vals.add(dateTime);
            ident.toArray(ids = new String[ident.size()]);
            vals.toArray(vls = new String[vals.size()]);
            if (!db.InsertTableValue("athletesdata", ids, vls)) {
                throw new Exception("Failed to insert Athlete's data into the
database");
            }
            // Inserts the RSSI measurements
            Vector<Long> adds = new Vector<Long>();
            Vector<Integer> rssis = new Vector<Integer>();
            for (int i = 4; i > 0; i--)
                // Only adds measurements form different towers
                if (this.getRSSIAddresses(i*9+5) != 0000000000000000000 &&
this.getRSSIs(i*9+13) != 0 && !adds.contains(this.getRSSIAddresses(i*9+5))){
                    adds.add(this.getRSSIAddresses(i*9+5));
                    rssis.add(this.getRSSIs(i*9+13));
                }
            ident = new Vector<String>();
            ident.add("towerId");
            ident.add("athleteId");
            ident.add("rssi");
            ident.add("dateTime");
            for (int i = 0; i < adds.size(); i++){
                    vals = new Vector<String>();
                    vals.add(this.towerExists(db, sessionId, adds.get(i)));
                    vals.add(athleteId);
                    vals.add(""+rssis.get(i));
                    vals.add(dateTime);
                    ident.toArray(ids = new String[ident.size()]);
                    vals.toArray(vls = new String[vals.size()]);
                    if (!db.InsertTableValue("athletesrssi", ids, vls)) {
                        throw new Exception("Failed to insert Athlete's RSSI
measurement into the database");
                    }
            }
        } catch (Exception e) {
            throw new Exception("Exception: Relative to an interaction with
the database - ", e);
        }
    }

    @Override
    public String toString() {
        DecimalFormat df = new DecimalFormat("0.00");
        String result = "Length: " + this.LENGTH;
        result += " Code: " + this.CODE;
        result += " 64BitAddress: " + this.SIXTY_FOUR_BITADDRESS;
        result += " 16BitAddress: " + this.SIXTEEN_BITADDRESS;
        result += " Option: " + this.OPTION;
        result += " Latitude: " + this.getLatL() + "." + this.getLatR() + " "
+ this.getNS();
        result += " Longitude: " + this.getLonL() + "." + this.getLonR() + " "
+ this.getEW();
        result += " Rhythm: " + df.format(this.getRhythm());
        for (int i = 1; i < 5; i++){
            result += " Tower(" + i + "): " + this.getRSSIAddresses(i*9+5);
            result += " RSSI(" + i + "): " + this.getRSSIs(i*9+13);
        }
        result += " CheckSum: " + this.CHECK_SUM;
        return result;
    }
}
```

DBThread.class

```
import data.*;
import java.awt.Color;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Vector;
import javax.swing.JOptionPane;

public class DBThread extends Thread{
    private MySQL sql; // The object which makes the interface with the DB
    private volatile GeneralMem mem; // The applications memory
    private WirelessTrackerView view;
    private boolean lastMobileBColor = true;
    private float h = -0.0557f, s = 0.5f, b = 0.5f;
    private int lastRows[] = new int[7]; // 0-athlete, 1-tower, 2-
accelerometer, 3-athletesdata, 4-ambient, 5-towersrssi, 6-athletesrssi

    public DBThread(GeneralMem m, WirelessTrackerView v){
        super();
        mem = m;
        sql = new MySQL();
        sql.Connect();
        view = v;
        for (int i = 0; i < 6; i++)
            lastRows[i] = 0;
        this.start();
    }

    private void processGPS(Transmiter t, String latL, String latR, String NS,
String lonL, String lonR, String EW, String d){
        // If the GPS coordenate isn't null reads it from the database into
the memory
        if (latL != null && latR != null && NS != null && lonL != null && lonR
!= null && EW != null){
            String[] gps = {latL, latR, NS, lonL, lonR, EW};
            t.gps.addLocation(gps, d);
        }
    }

     private Color generateColor(){
         // Generates a different color for each athlete
         if (h == 0.9469f) h = 1f;
         else if (h == 1f){
             h = -0.0557f;
             if (s == 0.5f){
                 s = 1;
                 b = 0.5f;
             }
             else if (s == 1){
                 s = 0.7f;
                 b = 0.7f;
             }
             else if (s == 0.7f){
                 s = 0.5f;
                 b = 0.5f;
             }
         }
         else h += 0.0557f;
         return Color.getHSBColor(h, s, b);
     }

    @Override
    @SuppressWarnings("static-access")
```

```
    public void run(){
        int time;
        try {
            DBThread.sleep(5000);
        } catch (InterruptedException ex) {
            Logger.getLogger(DBThread.class.getName()).log(Level.SEVERE, null,
ex);
        }
        view.centerView();
        System.out.println("DBThread-Running");

        if (sql.EnterDatabase()){
            System.out.println("Connection to the database successfull");
            Vector<Vector<Vector<String>>> r;
            Vector<Vector<String>> t;
            Vector<String> l;
            Mobile m;
            Fixed f;

            while(true){
                // Only fetches rows that hasn't allread read
                r = sql.getNewRows(lastRows);
                // Processes the results tables fom SQL queries
                for (int tab = 0; tab < r.size(); tab++){
                    t = r.get(tab);
                    if (!t.isEmpty()){
                        switch (tab){
                            case 0: // Athlete
                                for(int i = 0; i < t.size(); i++){
                                    l = t.get(i);
                                    // If there's no entity of this mobile
transmiter in memory
                                    if(!mem.getMobile().contains(l.get(0))){
                                        // Creates it with the id and name on
the results tables
                                        m = new Mobile(l.get(0), l.get(1),
generateColor(), lastMobileBColor = !lastMobileBColor);
//                                        m.addPos((int)2.7^i, (int)(i^2+i));
                                        mem.getMobile().add(l.get(0), m);
                                        // Adds the boards ftrom "m" to the
tabs
                                        view.addRB(m.getRhyBoard());
                                        view.addVB(m.getVelBoard());
                                    }
                                }
                                break;
                            case 1: // Tower
                                for(int i = 0; i < t.size(); i++){
                                    l = t.get(i);
                                    // If there's no entity of this fixed
transmiter in memory
                                    if(!mem.getFixed().contains(l.get(0))){
                                        // Creates it with the id and name on
the results tables
                                        if
(l.get(1).equals(mem.getGeneral().get("coordinator64"))){
                                            f = new Fixed(l.get(0), l.get(1),
"PC", Color.black);

mem.getGeneral().set("coordinatorId", l.get(0));
                                        } else f = new Fixed(l.get(0),
l.get(1), Color.black);
                                        mem.getFixed().add(l.get(0), f);
                                    } else f = mem.getFixed().get(l.get(0));
                                    // And processes the GPS data
```

```
                                    processGPS(f, l.get(4), l.get(5),
l.get(6), l.get(7), l.get(8), l.get(9), l.get(10));
                                }
                                break;
                            case 2: // Accelerometer

                                break;
                            case 3: // AthletesData
                                // And processes the new data
                                for(int i = 0; i < t.size(); i++){
                                    l = t.get(i);
                                    m = mem.getMobile().get(l.get(1));
                                    m.addRhy(l.get(9));
                                    processGPS(m, l.get(3), l.get(4),
l.get(5), l.get(6), l.get(7), l.get(8), l.get(2));
                                }
                                break;
                            case 4: // Ambient
                                for(int i = 0; i < t.size(); i++){
                                    l = t.get(i);
                                    // Processes the ambient data and the
voltage
                                    f = mem.getFixed().get(l.get(1));
                                    int tp = Integer.parseInt(l.get(2)),
                                            hm = Integer.parseInt(l.get(3)),
                                            li = Integer.parseInt(l.get(4));
                                    double temp = tp * 0.01 - 40,
                                    hum = (temp - 25) * (0.01 + 0.00008 * tp)
+ (-4 + 0.0405 * hm - 0.0000028 * hm * hm),
                                    lux = (li * 1.1 / 1024) * 6250;
                                    f.addTemp(temp);
                                    f.addHum(hum);
                                    f.addLig(lux);
                                }
                                break;
                            case 5: // TowersRSSI
                                for(int i = 0; i < t.size(); i++){
                                    l = t.get(i);
                                    // If origin tower is not the coordinator
                                    if
(!l.get(1).equals(mem.getGeneral().get("coordinatorId")))

mem.getFixed().get(l.get(1)).distances.addDistance(l.get(3), l.get(2),
l.get(4));
                                    // If origin(fliped) tower is not the
coordinator
                                    if
(!l.get(2).equals(mem.getGeneral().get("coordinatorId")))

mem.getFixed().get(l.get(2)).distances.addDistance(l.get(3), l.get(1),
l.get(4));
System.out.println("--- TOWER RSSI ADDED");
                                }
                                break;
                            case 6: // AthletesRSSI
System.out.println(t);
                                for(int i = 0; i < t.size(); i++){
                                    l = t.get(i);

mem.getMobile().get(l.get(2)).distances.addDistance(l.get(3), l.get(1),
l.get(4));
System.out.println("--- ATHLETE RSSI ADDED");
                                }
                                break;
                        }
```

```
                    }
                }
                time = 0;
                do{
                        view.updateView();
                    try {
                        DBThread.sleep(50);
                    }
                    catch (InterruptedException ex) {

Logger.getLogger(DBThread.class.getName()).log(Level.SEVERE, null, ex);
                    }
                    catch (Throwable ex) {

Logger.getLogger(DBThread.class.getName()).log(Level.SEVERE, null, ex);
                    }
                    time++;
                }while(time<20);
                // Memory Cleaners
                r.removeAllElements();
                System.gc();
                sql = new MySQL();
                sql.Connect();
                sql.EnterDatabase();
            }
        }
        else
            JOptionPane.showMessageDialog(view.getFrame(), "Connection to the
database failed!", "ERROR", JOptionPane.ERROR_MESSAGE);
    }
```

DistanceDataType.class

```
import digiusb.DateUtils;
import java.util.Date;

public class DistanceDataType {

    private int rssi;
    private String towerId;
    private Date date;

    public DistanceDataType(String r, String tId, String d) {
        rssi = Integer.parseInt(r);
        towerId = tId;
        date = DateUtils.parseDate(d);
    }

    public double getDistance() {
        return this.toM(rssi);
    }

    public double getDistanceKm() {
        return this.toKm(rssi);
    }

    public Date getTimeStamp() {
        return date;
    }

    public String getTowerId() {
        return towerId;
    }
```

```
    public double toM(int rssi) {
        // Converts the RSSI measurement to meters
        int dBm = 256 - ((rssi * 12 + 5928) / 41);
        dBm = -dBm + 7; // Towers emission power is -7 dBm
        double cst = -40.4; // Parameter due to the frequency 2.4 GHz
        return Math.round(Math.pow(10, (cst - dBm) / 20) * 100) / 100;
    }

    public double toKm(int rssi) {
        return this.toM(rssi) / 1000;
    }

    @Override
    public String toString() {
        return " Tower: " + towerId + ", RSSI: " + rssi + " = " +
this.getDistance() + "m, Date: " + date;
    }
}
```

ANEXO G – Classes da aplicação Java ME para telemóvel

Índice

Map.class

```
import java.io.IOException;
import javax.microedition.lcdui.*;

/**
 * @author Filipe Santos
 */
public class Map extends Canvas implements Runnable {

    /* Definição de variáveis de estado do mapa */
    private int selectedZoom = 0, defaultZoom = 16, markerX = -500, markerY =
-500, mapX = 0, mapY = 0, cursorX = getWidth()/2, cursorY = getHeight()/2;
    private int iconWidth = (int)(getWidth()/6), code;
    private String imagePath = null, coordGPS = null, keyPressed = "None";

    private float pixelLatVal, pixelLongVal, maxLatitude, minLatitude,
maxLongitude, minLongitude, valLat, valLong;

    /* Valores de referência da longitude e latitude da imagem utilizada */
    private static float x16MinLatitude = (float)(32.657925);
    private static float x16MinLongitude = (float)(-16.929953);
    private static float x16MaxLatitude = (float)(32.662550);
    private static float x16MaxLongitude = (float)(-16.919031);

    /* Variáveis de estado */
    private boolean showHelp = true, showIconHelp = false;

    /* Imagens utilizadas no layout do mapa */
    private Image mapImage = null, smallMarker = null, bigMarker = null;
    private Image thermometer = null, co = null, co2 = null, o2 = null, bright
= null, humi = null;

    private BluetoothConnect btConn;

    private Font screenFont = Font.getFont(Font.FACE_PROPORTIONAL,
Font.STYLE_PLAIN, Font.SIZE_SMALL);

    /* Variáveis que irão conter os valores recebidos do bluetooth */
    private float btLat = 0, btLong = 0, btHumi = 0, btTemp = 0, btO2 = 0,
btCO = 0, btCO2 = 0, btLuz = 0, btBat = 0;
    private String btRua = null;
```

```
    private CommandListener cmdList;
    private Display disp;

    /* Constructor da classe Map */
    public Map(BluetoothConnect conn, CommandListener cmd, Display disp) {
        this.disp = disp;
        cmdList = cmd;
        btConn = conn;

        try {
            // Prepara este canvas para escutar a eventos de comandos
            // Adiciona comando exit

if(!System.getProperty("microedition.platform").startsWith("Nokia"))
                addCommand(new Command("Quit", Command.SCREEN, 0));
        } catch(Exception e) {}

        if(selectedZoom == 0) {
            selectedZoom = defaultZoom;
            createImages(selectedZoom);
        }

        try {
            smallMarker = Image.createImage("/marker_small.png");
            bigMarker = Image.createImage("/marker_big.png");
            thermometer = Image.createImage("/thermometer.png");
            co = Image.createImage("/co.png");
            co2 = Image.createImage("/co2.png");
            o2 = Image.createImage("/o2.png");
            bright = Image.createImage("/brightness.png");
            humi = Image.createImage("/humidity.png");
        } catch (IOException ex) {}

        setFullScreenMode(true);

        calculaPosicoesGPS();

        /* Inicia uma tarefa do mapa, para ser possível deslocá-lo em tempo
real */
        new Thread(this).start();

        this.repaint();
    }

    /* Função que calcula as posições GPS utilizando as posições do cursor */
    public void calculaPosicoesGPS() {
        pixelLatVal = (maxLatitude - minLatitude)/mapImage.getHeight();
        pixelLongVal = (maxLongitude - minLongitude)/mapImage.getWidth();

        valLat = (float) ((maxLatitude - ((-mapY + cursorY))*pixelLatVal) -
0.000079);
        valLong = (float) ((minLongitude + ((-mapX + cursorX)*pixelLongVal)) +
0.000047);

        coordGPS = "Lat: "+valLat+"; Long: "+valLong;
    }

    /* Função que calcula a posição do marcador na imagem */
    public void calculaCoordenadasImagem(String gpsLat, String gpsLong) {
        float tempLat, tempLong;

        tempLat = Float.parseFloat(gpsLat);
        tempLong = Float.parseFloat(gpsLong);
```

```
        markerY = (int) (((tempLat + 0.000079 - maxLatitude) / -pixelLatVal) +
mapY - 30);
        markerX = (int) (((tempLong - 0.000047 - minLongitude) / pixelLongVal)
+ mapX - 15);
    }

    /* Função invocada por repaint() */
    public void paint(Graphics g) {
        g.setFont(screenFont);

        if(mapImage != null) {
            g.drawImage(mapImage, mapX, mapY, Graphics.SOLID); // Desenha
imagem do mapa
        }

        g.drawImage(bigMarker, markerX, markerY, Graphics.SOLID); // Desenha o
marcador principal

        /* Desenha o cursor */
        g.setColor(0, 255, 0);
        g.drawLine(cursorX - 10, cursorY, cursorX + 10, cursorY);
        g.drawLine(cursorX, cursorY - 10, cursorX, cursorY + 10);

        /* Desenha as informações do mapa (coordenadas, nome de rua, etc) -
Parte inferior do ecrã */
        g.setColor(255, 255, 255);
        g.fillRect(0, getHeight() - 35, getWidth(), 35);
        g.setColor(0, 0, 0);
        g.drawImage(smallMarker, 5, getHeight() - 33, Graphics.SOLID);
        g.drawRect(0, getHeight() - 35, getWidth(), getHeight() - 17);
        g.drawRect(0, getHeight() - 17, getWidth(), 17);
        if(btRua != null)
            g.drawString(btRua, 25, getHeight() - 33 +
smallMarker.getHeight()/2 - screenFont.getHeight()/2, Graphics.SOLID);
        g.drawString(coordGPS, (getWidth()/2) -
(screenFont.stringWidth(coordGPS)/2), getHeight() - 15, Graphics.SOLID);

        /* Desenha informações provenientes do bluetooth */
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, getWidth(), 40);

        g.setColor(0, 0, 0);
        g.drawImage(thermometer, (iconWidth - thermometer.getWidth())/2, 0,
Graphics.SOLID);
        g.drawImage(humi, iconWidth + (iconWidth - thermometer.getWidth())/2,
0, Graphics.SOLID);
        g.drawImage(bright, 2*iconWidth + (iconWidth -
thermometer.getWidth())/2, 0, Graphics.SOLID);
        g.drawImage(o2, 3*iconWidth + (iconWidth - thermometer.getWidth())/2,
0, Graphics.SOLID);
        g.drawImage(co, 4*iconWidth + (iconWidth - thermometer.getWidth())/2,
0, Graphics.SOLID);
        g.drawImage(co2, 5*iconWidth + (iconWidth - thermometer.getWidth())/2,
0, Graphics.SOLID);

        g.drawRect(0, 0, iconWidth, 40);
        g.drawRect(iconWidth, 0, 2*iconWidth, 40);
        g.drawRect(2*iconWidth, 0, 3*iconWidth, 40);
        g.drawRect(3*iconWidth, 0, 4*iconWidth, 40);
        g.drawRect(4*iconWidth, 0, 5*iconWidth, 40);
        g.drawRect(5*iconWidth, 0, 6*iconWidth, 40);
```

```
        g.drawString(""+(float)(((int)((btTemp+0.05)*10.0))/10.0), iconWidth /
2 - (screenFont.stringWidth("" + (float) (((int)((btTemp+0.05)*10.0))/10.0)) /
2), 40 - screenFont.getHeight(), Graphics.SOLID);
        g.drawString(""+(int)btHumi, iconWidth + iconWidth / 2 -
(screenFont.stringWidth("" + (int)btHumi) / 2), 40 - screenFont.getHeight(),
Graphics.SOLID);
        g.drawString(""+(int)btLuz, 2*iconWidth + iconWidth/2 -
screenFont.stringWidth(""+(int)btLuz)/2, 40 - screenFont.getHeight(),
Graphics.SOLID);
        g.drawString(""+(float)(((int)((btO2+0.05)*10.0))/10.0), 3*iconWidth +
iconWidth/2 -
screenFont.stringWidth(""+(float)(((int)((btO2+0.05)*10.0))/10.0))/2, 40 -
screenFont.getHeight(), Graphics.SOLID);
        g.drawString(""+(int)btCO, 4*iconWidth + iconWidth/2 -
screenFont.stringWidth(""+(int)btCO)/2, 40 - screenFont.getHeight(),
Graphics.SOLID);
        g.drawString(""+(int)btCO2, 5*iconWidth + iconWidth/2 -
screenFont.stringWidth(""+(int)btCO2)/2, 40 - screenFont.getHeight(),
Graphics.SOLID);

        g.setColor(255, 0, 0);
        g.drawString(btConn.getStatus(), 5, 60, Graphics.SOLID);
        if(!btConn.isDeviceFound())
            g.drawString("Node not found!", 5, 45, Graphics.SOLID);

        if(showHelp) {
            /* Mostra os comandos do programa */
            g.setColor(255, 255, 255);
            g.fillRect(10, 10, getWidth() - 20, getHeight() - 20);

            g.setColor(0, 0, 0);
            g.drawRect(10, 10, getWidth() - 20, getHeight() - 20);

            g.drawString("-= Help Menu =-", 15, 15, Graphics.SOLID);
            g.drawString("Up/Down - Move Up/Down", 15, 50, Graphics.SOLID);
            g.drawString("Left/Right - Move Left/Right", 15, 65,
Graphics.SOLID);
            g.drawString("Select - Center map", 15, 80, Graphics.SOLID);
            g.drawString("5 - Help On/Off", 15, 95, Graphics.SOLID);
            g.drawString("0 - Subtitles On/Off", 15, 110, Graphics.SOLID);
            g.drawString("Left/Right Button - EXIT", 15, 125, Graphics.SOLID);
        }
        else if(showIconHelp) {
            /* Mostra a legenda dos dados recebidos por bluetooth */
            g.setColor(255, 255, 255);
            g.fillRect(10, 10, getWidth() - 20, getHeight() - 20);

            g.setColor(0, 0, 0);
            g.drawRect(10, 10, getWidth() - 20, getHeight() - 20);

            g.drawImage(thermometer, 15, 15, Graphics.SOLID);
            g.drawImage(humi, 15, 40, Graphics.SOLID);
            g.drawImage(bright, 15, 65, Graphics.SOLID);
            g.drawImage(o2, 15, 90, Graphics.SOLID);
            g.drawImage(co, 15, 115, Graphics.SOLID);
            g.drawImage(co2, 15, 140, Graphics.SOLID);

            g.setColor(0, 0, 0);
            g.drawRect(15, 15, 25, 25);
            g.drawRect(15, 40, 25, 25);
            g.drawRect(15, 65, 25, 25);
            g.drawRect(15, 90, 25, 25);
            g.drawRect(15, 115, 25, 25);
```

```
            g.drawRect(15, 140, 25, 25);

            g.drawString("Temperature (Â°C)", 45, 15 +
thermometer.getHeight()/2 - screenFont.getHeight()/2, Graphics.SOLID);
            g.drawString("Humidity (%)", 45, 40 + thermometer.getHeight()/2 -
screenFont.getHeight()/2, Graphics.SOLID);
            g.drawString("Brightness (lux)", 45, 65 +
thermometer.getHeight()/2 - screenFont.getHeight()/2, Graphics.SOLID);
            g.drawString("Oxygen (%)", 45, 90 + thermometer.getHeight()/2 -
screenFont.getHeight()/2, Graphics.SOLID);
            g.drawString("Carbon monoxide (ppm)", 45, 115 +
thermometer.getHeight()/2 - screenFont.getHeight()/2, Graphics.SOLID);
            g.drawString("Carbon dioxide (ppm)", 45, 140 +
thermometer.getHeight()/2 - screenFont.getHeight()/2, Graphics.SOLID);
        }
    }

    /* Carrega a imagem do mapa */
    private void createImages(int zoom) {

        switch(zoom) {
            case 16:
                imagePath = "/FxMaps16/pic_map_16.jpg";

                maxLatitude = x16MaxLatitude;
                minLatitude = x16MinLatitude;
                maxLongitude = x16MaxLongitude;
                minLongitude = x16MinLongitude;
                break;
        }

        try {
            mapImage = Image.createImage(imagePath);
        } catch (IOException ex) {}
    }

    /* Chamado quando as teclas são pressionadas */
    protected void keyPressed(int keyCode) {
        keyPressed = getKeyName(keyCode);
        code = keyCode;

        if(getGameAction(keyCode) == Canvas.UP) {
            if(cursorY - 1 >= 40) {
                cursorY--;
            }
            else {
                if(mapY + 1 <= 40) {
                    mapY++;
                }
            }
        }
        else if(getGameAction(keyCode) == Canvas.DOWN) {
            if(cursorY + 1 <= getHeight() - 36 && cursorY + 1 <=
mapImage.getHeight()) {
                cursorY++;
            }
            else {
                if(mapY + mapImage.getHeight() - 1 >= getHeight() - 35) {
                    mapY--;
                }
            }
        }
        else if(getGameAction(keyCode) == Canvas.LEFT) {
```

```
            if(cursorX - 1 >= 0) {
                cursorX--;
            }
            else {
                if(mapX + 1 <= 0) {
                    mapX++;
                }
            }
        }
        else if(getGameAction(keyCode) == Canvas.RIGHT) {
            if(cursorX + 1 <= getWidth() && cursorX + 1 <=
mapImage.getWidth()) {
                cursorX++;
            }
            else {
                if(mapX + mapImage.getWidth() - 1 >= getWidth()) {
                    mapX--;
                }
            }
        }
        else if(keyPressed.equals("5")) {
            if(showHelp)
                showHelp = false;
            else if(!showIconHelp)
                showHelp = true;
        }
        else if(keyPressed.equals("0")) {
            if(showIconHelp)
                showIconHelp = false;
            else if(!showHelp)
                showIconHelp = true;
        } else if(getGameAction(keyCode) == Canvas.FIRE && (markerX != -500 &&
markerY != -500) && !keyPressed.equals("5")) {
            mapX = - (markerX - mapX - getWidth()/2);
            mapY = - (markerY - mapY - getHeight()/2);

            if(mapX < - mapImage.getWidth() + getWidth())
                mapX = - mapImage.getWidth() + getWidth();
            else if(mapX > 0)
                mapX = 0;

            if(mapY < - mapImage.getHeight() + getHeight() - 35)
                mapY = - mapImage.getHeight() + getHeight() - 35;
            else if(mapY > 40)
                mapY = 40;
        } else if(keyCode == -6 || keyCode == -7) {
            cmdList.commandAction(new Command("Exit", Command.EXIT, 1), this);
        }

        calculaPosicoesGPS();
        calculaCoordenadasImagem(""+btLat, ""+btLong);

        repaint();
    }

    /* Chamada quando uma tecla é despressionada */
    protected  void keyReleased(int keyCode) {
    }

    /* Chamada quando uma tecla é segurada x tempo */
    protected  void keyRepeated(int keyCode) {
        keyPressed = getKeyName(keyCode);
```

```
        if(getGameAction(keyCode) == Canvas.UP) {
            if(cursorY - 10 >= 40) {
                cursorY-=10;
            }
            else {
                if(mapY + 10 <= 40) {
                    mapY+=10;
                }
            }
        }
        else if(getGameAction(keyCode) == Canvas.DOWN) {
            if(cursorY + 10 <= getHeight() - 36  && cursorY + 10 <=
mapImage.getHeight()) {
                cursorY+=10;
            }
            else {
                if(mapY + mapImage.getHeight() - 10 >= getHeight() - 35) {
                    mapY-=10;
                }
            }
        }
        else if(getGameAction(keyCode) == Canvas.LEFT) {
            if(cursorX - 10 >= 0) {
                cursorX-=10;
            }
            else {
                if(mapX + 10 <= 0) {
                    mapX+=10;
                }
            }
        }
        else if(getGameAction(keyCode) == Canvas.RIGHT) {
            if(cursorX + 10 <= getWidth()  && cursorX + 10 <=
mapImage.getWidth()) {
                cursorX+=10;
            }
            else {
                if(mapX + mapImage.getWidth() - 10 >= getWidth()) {
                    mapX-=10;
                }
            }
        }

        calculaPosicoesGPS();
        calculaCoordenadasImagem(""+btLat, ""+btLong);

        repaint();
    }

    /**
     * Called when the pointer is dragged.
     */
    protected  void pointerDragged(int x, int y) {
    }

    /**
     * Called when the pointer is pressed.
     */
    protected  void pointerPressed(int x, int y) {
    }

    /**
     * Called when the pointer is released.
```

```
     */
    protected  void pointerReleased(int x, int y) {
    }

    /* Tarefa que recebe as informações correctas a partir do bluetooth */
    public void run() {
        while(btConn.isBluetoothOn()) {
            if(btConn.isDataCorrect()) {
                btRua = btConn.getRua();
                btLat = btConn.getLat();
                btLong = btConn.getLong();
                btTemp = btConn.getTemp();
                btLuz = btConn.getLuz();
                btO2 = btConn.getO2();
                btCO = btConn.getCO();
                btCO2 = btConn.getCO2();
                btBat = btConn.getBat();
                btHumi = btConn.getHumidity();

                keyPressed(10000);
            }

            try {
                Thread.sleep(4000);
            } catch (InterruptedException ex) {}
        }
    }

}
```

BluetoothConnect.class

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.microedition.io.*;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Display;

/**
 *
 * @author Filipe Santos
 */
public class BluetoothConnect extends Thread implements DiscoveryListener {

    /* Variáveis de estado da ligação bluetooth */
    private boolean exit = false, foundDevice = false, correctData = false,
finished = false, started = false, errorConnecting = false;
    private StreamConnection streamCon = null;
    private InputStream inputStream = null;
    private OutputStream outputStream = null;

    private String message = null, rua = null;

    /* Constantes utilizadas no cálculo dos parâmetros fornecidos */
```

```
    private final float C1 = (float) -4.0, C2 = (float) 0.0405, C3 = (float) -
0.0000028, T1 = (float) 0.01, T2 = (float) 0.00008;

    /* Variáveis que irão conter os valores recebidos por bluetooth */
    private int humi;
    private float lat = 0, lon = 0, temp = 0, rh_lin = 0, humidity = 0, luz =
0, o2 = 0, co = 0, co2 = 0, bat = 0;
    private String btName = "", btAddr = "", btStatus = "", btAddrError = "";

    private Display disp;
    private LocalDevice dev;

    /* Constructor da classe BluetoothConnect */
    public BluetoothConnect(Display disp) {
        this.disp = disp;
        btName = "";
        btAddr = "";
    }

    /* Função que permite verificar se estão sendo recebidos dados */
    public boolean isRunning() {
        return !exit;
    }

    /* Desliga a ligação bluetooth */
    public void stopBluetoothConnection() {
        exit = true;
    }

    /* Função que retorna o valor do nome da rua recebido */
    public String getBTName() {
        return btName;
    }

    /* Permite verificar se os dados recebidos estão correctos */
    public boolean isDataCorrect() {
        return correctData;
    }

    /* Permite verificar se o dispositivo a que se pretende ligar está
         disponivel */
    public boolean isDeviceFound() {
        return foundDevice;
    }

    /* Retorna o estado do bluetooth */
    public String getStatus() {
        return btStatus;
    }

    /* Tarefa para obtenção dos dados através do bluetooth */
    public void run() {
        if(!isBluetoothOn()) {
            Alert a = new Alert("ERROR", "Turn Bluetooth ON.", null,
AlertType.ERROR);
            a.setTimeout(Alert.FOREVER);
            disp.setCurrent(a, disp.getCurrent());

            exit = true;
        }

        while(!exit) {
            correctData = false;
```

```
            finished = false;
            //foundDevice = false;

            btAddr = "";
            btName = "";

            btStatus = "Searching...";

            startSearch();

            while(!finished);

            btStatus = "Finished";

            if(!btAddr.equals("")) {
                /* Inicia licação bluetooth */
                try {
                    streamCon = (StreamConnection)
Connector.open("btspp://"+btAddr+":1"+
                            ";authenticate=false;master=false;encrypt=false",
                            Connector.READ_WRITE);

                    btStatus = "Connecting to "+btName;

                    outputStream = streamCon.openOutputStream();

                    inputStream = streamCon.openInputStream();

                    /* Recebe dados até que os delimitadores do pacote
recebido sejam os correctos */
                    do {
                        outputStream.write('c');
                        outputStream.flush();

                        byte[] buffer = new byte[42];
                        inputStream.read(buffer);

                        message = new String(buffer);
                    } while(message.charAt(0) != 0x7E || message.charAt(41) !=
0x90);

                    btStatus = "Got message from "+btName;

                    foundDevice = true;

                } catch(Exception e) {
                    btStatus = "Connection error to "+btName;
                    btAddrError = btAddr;

                    foundDevice = false;

                    btAddr = "";
                    btName = "";
                }

                finally {
                    if (streamCon != null) {
                        try {
                            streamCon.close();
                        } catch (IOException ignored) {}

                        streamCon = null;
                    }
```

```
                    if (inputStream != null) {
                        try {
                            inputStream.close();
                        } catch (IOException ignored) {}

                        inputStream = null;
                    }
                    if (outputStream != null) {
                        try {
                            outputStream.flush();
                            outputStream.close();
                        } catch (IOException ex) {}

                        outputStream = null;
                    }

                    btStatus = "Closed connection";

                    /* Caso seja encontrado um dispositivo, efectua os
cálculos dos valores */
                    if(foundDevice) {
                        temp = (float)(((message.charAt(1) << 8) +
(message.charAt(2)))*0.01 - 40);

                        humi = (message.charAt(3) << 8) + (message.charAt(4));
                        rh_lin = (float) (C3*humi*humi + C2*humi + C1);
                        humidity = (float) ((temp - 25)*(T1 + T2*humi) +
rh_lin);

                        if(humidity > 100)
                            humidity = (float) 100;
                        else if(humidity < 0.1)
                            humidity = (float) 0.1;

                        luz = (float) (((message.charAt(5) << 8) +
(message.charAt(6)))*2.56/1024);
                        luz = (float) (0.625*luz*10000);

                        o2 = (float) ((((((message.charAt(7) << 8) +
(message.charAt(8)))*1000*2.56*1.78)/1024/40)*100)/100);

                        co = (float) (((((((message.charAt(9) << 8) +
(message.charAt(10)))*2.56/1024)*10000/101)/1.374)*100)/100);

                        co2 = (float) ((message.charAt(11) << 8) +
(message.charAt(12)));

                        bat = (float) ((((message.charAt(13) << 8) +
(message.charAt(14)))*2.56)/1024);

                        lat = (float) ((message.charAt(15)) +
(((message.charAt(16) << 16) + (message.charAt(17) << 8) +
(message.charAt(18)))*0.000001));
                        lon = (float) -((message.charAt(19)) +
(((message.charAt(20) << 16) + (message.charAt(21) << 8) +
(message.charAt(22)))*0.000001));
                        rua = message.substring(23, 41);

                        correctData = true;

                        btStatus = "Data received from "+btName;
                    }
                }
```

```
            }

            try {
                Thread.sleep(10000);
            } catch (InterruptedException ex) {}
        }
    }

    /* Função que retorna o valor da rua recebido */
    public String getRua() {
        if(rua != null)
            return rua;
        else return "Street not available";
    }

    /* Função que retorna o valor da latitude recebido */
    public float getLat() {
        return lat;
    }

    /* Função que retorna o valor da longitude recebido */
    public float getLong() {
        return lon;
    }

    /* Função que retorna o valor da temperatura recebido */
    public float getTemp() {
        return temp;
    }

    /* Função que retorna o valor da humidade recebido */
    public float getHumidity() {
        return humidity;
    }

    /* Função que retorna o valor da luz recebido */
    public float getLuz() {
        return luz;
    }

    /* Função que retorna o valor de O2 recebido */
    public float getO2() {
        return o2;
    }

    /* Função que retorna o valor de CO recebido */
    public float getCO() {
        return co;
    }

    /* Função que retorna o valor de CO2 recebido */
    public float getCO2() {
        return co2;
    }

    /* Função que retorna o valor da bateria recebido */
    public float getBat() {
        return bat;
    }

    /* Caso seja encontrado um dispositivo bluetooth esta Função Ã© invocada
*/
    public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
```

```
        try {
            if (btDevice.getBluetoothAddress().startsWith("000780") &&
!btDevice.getBluetoothAddress().equals(btAddrError) &&
btDevice.getFriendlyName(false).startsWith("WT")) {
                // || btName.startsWith("WT11")) { //wt11-'0007809265bf' wt12-
00078089881c
                btAddr = btDevice.getBluetoothAddress();

                btName = btDevice.getFriendlyName(false);

                dev.getDiscoveryAgent().cancelInquiry(this);
            }
        } catch (IOException ex) {}
    }

    public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
    }

    public void serviceSearchCompleted(int transID, int respCode) {
    }

    /* Invocada quando acaba a pesquisa por dispositivos bluetooth */
    public void inquiryCompleted(int discType) {
        btAddrError = "";

        finished = true;
    }

    /* Inicia a pesquisa de dispositivos bluetooth */
    public void startSearch() {
        try {
            started =
dev.getDiscoveryAgent().startInquiry(DiscoveryAgent.GIAC, this);
        } catch (BluetoothStateException ex) {}
    }

    /* Função que permite verificar se o bluetooth está ligado ou nÃ£o */
    public boolean isBluetoothOn() {
        try {
            dev = LocalDevice.getLocalDevice();
        } catch (BluetoothStateException ex) {
            return false;
        }
        return true;
    }
}
```

ANEXO H – *Firmware* dos vários módulos do sistema

Índice

AmbiSensor

```
/*****************************************************************************
* Nome do ficheiro : ambientsensor.c                                         *
* Autor            : Filipe Santos                                           *
* Data                          : 28/02/2009                                 *
* Descrição        : Este ficheiro é usado para o ATmega324                  *
*                  : comunicar com módulo ZigBee XBEE,                       *
*                  : módulo Bluetooth WT11, Hitachi display,                 *
*                                    : e sensores TGS5042, C20, KE-25, S1087 *
*                                    : e SHT15                               *
******************************************************************************/

#define F_CPU 8000000UL // Velocidade do microcontrolador

/****************************** LIVRARIAS ************************************/
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <avr/power.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include "hd44780_hw.h"
#include "hd44780.h"
#include "avr_compat.h"
/*****************************************************************************/

/*************************** NOMENCLATURA E CONSTANTES ***********************/
#define XBEESLEEP       PC4     // Pino ligado ao pino sleep do XBEE
#define XBEESTATUS  PC5         // Pino ligado ao pino status do XBEE
#define LED             PD6     // Led de indicação de funcionamento
#define TX                      PC0     // Pino para TX Síncrono Sensor CO2
#define RX                      PC1             // Pino para RX Síncrono Sensor CO2
#define SHDNV5          PB3             // Controlo de Shutdown do Step-up de 5V
#define DATA            PC2             // Pino PC2 usado como linha DATA para o SHT15
(dados)
#define SCK             PC3     // Pino PC3 usado como linha SCK para o SHT15 (relogio)
typedef union                           // Estrutura para o sensor de humidade e
temperatura
{ unsigned int i;
  float f;
} value;
enum {TEMP,HUMI};

#define noACK 0                         // Nomenclatura para recepção de nível zero
#define ACK   1                         // Nomenclatura para recepção de nível um
// Comandos do sensor STH15
#define STATUS_REG_W 0x06   //000   0011    0
#define STATUS_REG_R 0x07   //000   0011    1
#define MEASURE_TEMP 0x03   //000   0001    1
#define MEASURE_HUMI 0x05   //000   0010    1
#define RESET1       0x1e   //000   1111    0
// Operações comuns
#define HI(x) (x>>8)
#define LO(x) (x&0xff)
#define BSET(p,b)   ((p) |=  (1<<b))   // Set bit
#define BCLR(p,b)   ((p) &= ~(1<<b))   // Clear bit


/********************* INICIALIZAÇÃO DE VARIÁVEIS ****************************/
volatile char letra;            // Byte auxiliar para as comunicações UART
volatile int segundos=0;        // Variável global para tempo (segundos)
// Variáveis que mantém os valores actuais dos sensores
volatile int temperatura=0,humidade=0,co2=0,co=0,o2=0,luz=0,bateria=0;
volatile int displaymode=0,btresponse=0; // Variáveis auxiliares para a visualização

// Declaração dos "Function pointers"
```

```
char s_write_byte(unsigned char value);
char s_read_byte(unsigned char ack);
void s_transstart(void);
void s_connectionreset(void);
char s_softreset(void);
char s_read_statusreg(unsigned char *p_value, unsigned char *p_checksum);
char s_write_statusreg(unsigned char *p_value);
char s_measure(unsigned char *p_value, unsigned char *p_checksum, unsigned char mode);
void calc_sth11(float *p_humidity ,float *p_temperature);
void USART_Init( unsigned int baud );
void USART_Transmit( unsigned char data );
unsigned char USART_Receive( void );
void USART0_Init( unsigned int baud );
void USART0_Transmit( unsigned char data );
unsigned char USART0_Receive( void );
void configTimer2(void);
void principal(void);
void UART_SIM_TX(unsigned char letra);
unsigned char UART_SIM_RX(void);
void repostaBLUETOOTH(unsigned char comando);
void respostaXBEE(void);
int medirTensao( char adn );
int medeCO2(void);
void displayInfo(int dm);
unsigned char   checksum_1_byte (unsigned char * data_in,unsigned short
number_of_bytes_to_read, short array_index_counter);
static int UART_putChar(char c, FILE *stream);
static FILE USART_out = FDEV_SETUP_STREAM(UART_putChar, NULL, _FDEV_SETUP_WRITE);
static FILE lcd_stream = FDEV_SETUP_STREAM(lcd_putc_stream, NULL, _FDEV_SETUP_WRITE);

/************************ INÍCIO DA EXECUÇÃO PROGRAMA *************************/
int main()
{
        // Configuração inicial dos vários componentes e das interrupções de tempo
        BSET(DDRC,XBEESLEEP);   // Coloca o pino XBEESLEEP como output
        BCLR(PORTC,XBEESLEEP);  // Define nível 0 - XBEE activo
        BSET(DDRB,SHDNV5);              // Pino SHDNV5 como output
        BSET(PORTB,SHDNV5);             // Define nível 1 - Step-up 5V desactivo
        BSET(DDRD,LED);                 // Coloca o pino LED como output
        BSET(PORTD,LED);                // Desliga LED
        BSET(DDRD,7);                   // Coloca o pino AMPOP como output
        BCLR(PORTD,7);                  // Desliga AMPOP
        BSET(DDRC,TX);                  // Coloca o pino TX como output
        BCLR(PORTC,TX);                 // Define nível 0 no TX
        BCLR(DDRC,RX);                  // RX como input
        BCLR(PORTC,RX);                 // Define nível 0 no RX

        BSET(DDRC,SCK);                 // Coloca o pino SCK como output
        BSET(DDRC,DATA);                // coloca o pino DATA como output
        BCLR(PORTC,SCK);                // Define nível 0 no SCK

        BSET(PCICR,PCIE3);              // Interrupção para o botão
        BSET(PCMSK3,PD5);               // Activa interrupção de alteração no pino PD5
PCINT29

        clock_prescale_set(clock_div_1);        // Pre-scale do uC a 0 8MHz x 1
        USART_Init(8);                  // Incialização UART - Bluetooth 115200bps
8bit+stopbit
        USART0_Init(51);                // Incialização UART - XBEE 9600bps 8bit+stopbit
        // CONFIGURAÇÃO DO TEMPO DE INTERRUPÇÃO QUE CONTROLAR O PERÍODO DE AMOSTRAS
        configTimer2();                 // Configuração do RTC
        _delay_ms(1000);                // Espera 1 s para o cristal de relógio
estabilizar
                                                // (NOTA: é preciso esperar 20ms
pelo sensor de humidade)
        BSET(PORTC,XBEESLEEP);  // Adormece XBEE
        stdout = &lcd_stream;   // Impressão selecionada para o display
        DIDR0=0x00;                             // Configuração doa ADCs
        ADMUX = 0xC0;                   // Escolha do ADC 0
        ADCSRA = 0x86;                  // Escolha da forma de saída do resultado de
conversão

        set_sleep_mode(SLEEP_MODE_PWR_SAVE); // define o nível de adormecimento
        sei();                                  // Activa as interrupções arranca o RTC
        segundos=0;                             // Incializa variável de tempo
```

```
        // ciclo principal do programa
        for(;;)
        {
                // Se não estiver a responder a pedidos de Bluetooth adormece
                // e fica à espera da próxima interrupção
                if(!btresponse){
                        BSET(PCMSK3,PD2);               //  Activa interrupção recepção
bluetooth PD2
                        sleep_enable();
                    sleep_mode();                       // Adormece o uC
                    sleep_disable();            // Após a interrupção de acordar
                        power_all_enable();             // Obriga o acordar de todos os
sistemas
                }
        }
}

/****************************** FUNÇÕES ************************************/

/* principal - Função chamada periódicamente pelo RTC para construir mensagem API
 * para XBEE com os valores amostrados dos sensores.
 * Acorda o XBEE envia a mensagem.E termina ordenando o XBEE a adormecer.
*/
void principal(void){
        value humi_val,temp_val;
        unsigned char error=0,checksum,cs=0;
        int i=0;
        // Inicialização da msg ZigBee para API do XBEE
        char msg[32] = {0x7e,0x00,0x1c,0x10,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xee};
        BCLR(PORTC,XBEESLEEP);  // Acorda XBEE
        _delay_ms(14);                  // Espera que o XBEE acorde 13.2ms

        // Recolhe medida de humidade + temperatura do SHT15
        error+=s_measure((unsigned char*) &humi_val.i,&checksum,HUMI);  // Mede humi
        error+=s_measure((unsigned char*) &temp_val.i,&checksum,TEMP);  // Mede temp

        if(error!=0) s_connectionreset();// No caso da comunicação falhar com o SHT15
    else{
          temperatura = temp_val.i;     // regista valor inteiro da temperatura
          humidade = humi_val.i;        // regista valro inteiro da humidade
    }
        // Preench a msg sepraando as variáveis em dois bytes
        // Exemplo: variável de 16 bits, vai em dois bytes seguindo primeiro a parte
        // mais significativa depois a menos significativa
        msg[17] = HI(temperatura);
        msg[18] = LO(temperatura);
        msg[19] = HI(humidade);
        msg[20] = LO(humidade);
        msg[21] = HI(luz);
        msg[22] = LO(luz);
        msg[23] = HI(o2);
        msg[24] = LO(o2);
        msg[25] = HI(co);
        msg[26] = LO(co);
        msg[27] = HI(co2);
        msg[28] = LO(co2);
        msg[29] = HI(bateria);
        msg[30] = LO(bateria);
        // Cálculo de checksum da mensagem
        cs = checksum_1_byte((unsigned char*) &msg,28,3);
        msg[31] = cs;
        //loop_until_bit_is_set(PORTC,XBEESTATUS); // Não é necessario, já esperamos
        // Envio da mensagem
        for(i=0;i<32;i++){
                USART0_Transmit(msg[i]);  // Envia mensagem pela porta 0
        }
        USART0_Transmit('\r');
        _delay_us(1100);                // Espera que o último byte tenha sido enviado
        BSET(PORTC,XBEESLEEP);  // Adormece XBEE
}
```

```
/* mediTensao - Função que amostra o ADC indicado pelo campo adn
 * realiza um algoritmo para melhorar a precisão da medida
 * através da média de 16 amostras. Retorna um valor de 10 bit's
*/
int medirTensao( char adn ){
        int aux1=0,aux2=0,auxt=0;
        int i;
        ADMUX=0xC0+adn; // Seleciona o sensor a medir
        for(i=0;i<16;i++){
                ADCSRA |= (1<<ADSC);                            // Inicia a conversão
                loop_until_bit_is_set(ADCSRA, ADIF);// Espera conversão terminar
                // ADCL + ADCH = 10bit's
                aux1 = ADCL;
                aux2 = ADCH;
                aux1 = (aux1);
                aux2 = (aux2<<8);
                aux1 = aux2+aux1;
                auxt = auxt+aux1;
        }
        auxt = (auxt>>4);
        //auxt = (float)((auxt*2.561)/1023); // Conversão ADC 10 bits para decimal
        return auxt;
}

/* respostaXBEE - Função extra implementada para testes
 * Se receber uma mensagem de dados via rede ZigBee
 * Apresenta no display o nível de sinal - RSSI
*/
void respostaXBEE(void){
        letra = USART0_Receive();
        if(letra == 0x7E){      // se recebeu mensagem da rede XBEE
        letra = USART0_Receive(); // espera que a mensagem seja recebida
        letra = USART0_Receive();
        letra = USART0_Receive(); //  ''
                if(letra == 0x88){                              // se recebeu resposta
RSSI do modulo
                        letra = USART0_Receive();       // espera que a mensagem seja
recebida
                        letra = USART0_Receive();
                        letra = USART0_Receive();
                        letra = USART0_Receive();
                        letra = USART0_Receive();       //    ''
                        lcd_init(LCD_DISP_ON);          // Apresenta valor no display
                        stdout = &lcd_stream;
                        printf("RSSI: -%idBm",letra);
                }else{
                        _delay_ms(30);
                        USART0_Transmit(0x7E);          // pergunta RSSI da última
menssagem
                        USART0_Transmit(0x00);
                        USART0_Transmit(0x04);
                        USART0_Transmit(0x08);
                        USART0_Transmit(0x01);
                        USART0_Transmit(0x44);
                        USART0_Transmit(0x42);
                        USART0_Transmit(0x70);
                }
        }

}

/* repostaBLUETOOTH - Função para entregar os dados a um requisitante via bluetooth
 * O comando de pergunta aceite é a letra 'c'
 * Responde msg com os dados dos sensores, posição GPS do módulo e nome da rua
*/
void repostaBLUETOOTH(unsigned char comando){
        int i =0;
        char msg[42] = {0x7e,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                        'C','a','m','p','u','s',' ','d','a',' ',
                                        'P','e','n','t','e','a','d','a',0x00};
        switch(comando){
                case 'e':       // Se receber a letra 'e' devolve 'OK'
                        USART_Transmit('O');
```

```
                        USART_Transmit('K');
                        _delay_ms(2);
                        btresponse = 0;
                        break;
                case 'c':
                        // valores amostrados dos sensores
                        msg[1] = HI(temperatura);
                        msg[2] = LO(temperatura);
                        msg[3] = HI(humidade);
                        msg[4] = LO(humidade);
                        msg[5] = HI(luz);
                        msg[6] = LO(luz);
                        msg[7] = HI(o2);
                        msg[8] = LO(o2);
                        msg[9] = HI(co);
                        msg[10] = LO(co);
                        msg[11] = HI(co2);
                        msg[12] = LO(co2);
                        msg[13] = HI(bateria);
                        msg[14] = LO(bateria);
                        // Posicão GPS  32.658500   N -16.924000  W (esquina sul do
edifico)
                        msg[15] = 0x20;
                        msg[16] = 0x0A;
                        msg[17] = 0x0C;
                        msg[18] = 0x44;
                        msg[19] = 0x10;
                        msg[20] = 0x0E;
                        msg[21] = 0x19;
                        msg[22] = 0x60;
                        msg[41] = 0x90;
                        for(i=0;i<42;i++){
                                USART_Transmit(msg[i]); // Envia dados por Bluetooth
                        }
                        _delay_ms(2);
                        btresponse = 0;
                        break;
                default: // Em caso de comando não reconhecido
                        USART_Transmit(comando);
                        USART_Transmit('E');
                        USART_Transmit('R');
                        USART_Transmit('R');
                        USART_Transmit('O');
                        USART_Transmit('R');
                        _delay_ms(2);
                        btresponse = 0;
                        break;

        }
}

/* medeCO2 - Lê a reposta do sensor de CO2 via a UART criada por software
 * Devole total num inteiro
*/
int medeCO2(void){
        int total=0;
        unsigned char comando='0';
        while(comando != 'Z'){ // Escuta mensagem do CO2, exemplo: "z 00030"
                comando = UART_SIM_RX();
        }
        comando = UART_SIM_RX(); // Ignora espaço
        total += (((int) UART_SIM_RX())-0x30)*100000;
        total += (((int) UART_SIM_RX())-0x30)*10000;
        total += (((int) UART_SIM_RX())-0x30)*1000;
        total += (((int) UART_SIM_RX())-0x30)*100;
        total += (((int) UART_SIM_RX())-0x30)*10;
        return total;
}

/************************ FUNÇÕES AUXILIARES *********************************/
/* UART_SIM_TX - Simula a emissão de dados por um pino do uC como se
 * de uma porta UART se tratasse
 * Transmite a 9600bps 8bit's mais 1 stop bit
*/
void UART_SIM_TX(unsigned char letra){
```

```
        unsigned char bitT=102,bitValor = 0; // multiplos de us = tempo de um bit
        unsigned char byteComprimento = 8;
        PORTC&=~(_BV(TX));                                      // Envia bit de inicio
        _delay_us(bitT);                                        // Espera 1 bit
        while(byteComprimento > 0){
                bitValor = letra & 0x01;                // Busca bit menos significativo
                // Coloca no pino TX o valor do bit
                if(bitValor == 0){
                        PORTC&=~(_BV(TX));
                }else{
                        PORTC|=(1<<TX);
                }
                _delay_us(bitT);                                // Espera 1 bit
                letra = letra>>1;                               // Passa ao proximo bit
da letra
                byteComprimento--;
        }
        PORTC|=(1<<TX);                                         // Envia stop bit
        _delay_us(bitT);                                        // Espera 2 bits

}
/* UART_SIM_RX - Simula a recepção de dados por um pino do uC como se
 * de uma porta UART se tratasse
 * Recebe a 9600bps 8bit's mais 1 stop bit
*/
unsigned char UART_SIM_RX(void){
        unsigned char bitT=102,bitT2=51; // múltiplos de us = tempo de um bit
        unsigned char byteComprimento = 0,letra=0;
        // espera pelo start bit LOW
        // ou faz X ciclos à espera e desiste para não prender o programa aqui +/- 10ms
        while(bit_is_set(PINC,RX)){
        }
        // espera 1,5bits
        _delay_us(bitT2);
        _delay_us(bitT2);
        _delay_us(bitT2);
        while(byteComprimento < 8){
                // 1 bit
                if(bit_is_set(PINC,RX)){
                        letra |= (1<<byteComprimento);
                }
                byteComprimento++;
                _delay_us(bitT);
        }
        // espera pelo stop bit acabar
        _delay_us(bitT2);
        return letra;
}


/* checksum_1_byte - Calcula o byte de checksum para a mensagem API do XBEE.
 * Soma todos os bytes, excepção dos bytes start e lenght = 3 primeiros bytes
 * Subtrai o resultado de 8 bits a 255 = 0xFF (hexadecimal). Retorna resultado
*/
unsigned char checksum_1_byte (unsigned char * data_in,unsigned short
number_of_bytes_to_read,short array_index_counter){
        unsigned char   checksum;
        checksum = 0;
    number_of_bytes_to_read+=array_index_counter;
    while (array_index_counter != number_of_bytes_to_read){
      array_index_counter++;
      checksum += data_in[array_index_counter - 1];
        }
        checksum = 0xFF-checksum;
    return checksum;
}

/* configTimer2 - Configura o timer do 2 do uC para trabalhar como um Real Time Clock.
 * Soma todos os bytes, excepção dos bytes start e lenght = 3 primeiros bytes
 * Períodos para diferentes prescalers: 0x05 = 1s, 0x06 = 2s, 0x04 = 0,5s
*/
void configTimer2(void){
        ASSR  =0x20; // Activa modo assincrono
        TIMSK2=0x00; // Desactiva todas as interrupções para alteração dos registos
        OCR2A =0x00;
```

```
        TCCR2A=0x00; // Configura modo normal
        TIMSK2=0x01; // Activa interrupção de overflow
        TCCR2B=0x05; // Inicia o timer com prescaler de 128 = 1s
}

/* displayInfo - Função para apresentar os dados no display
 * Apresenta conjuntos de 2 dados de cada vez que pressionado o botão
 * Aplica as equações de conversão
*/
void displayInfo(int dm){
        switch(dm){     // Casos de visualização
                        case 1: // Temperatura e humidade
                                humi=(float)humidade;   // Converte inteiro para float
                                temp=(float)temperatura;
                                calc_sth11(&humi,&temp); // Converte os valores (por
referência)
                                BSET(PORTC,RX);
                                BCLR(PORTB,SHDNV5);
                                lcd_init(LCD_DISP_ON);
                                stdout = &lcd_stream;
                                printf("Temp: %5.1f C\nHumi: %5.1f%%rh",temp,humi);
                        break;
                        case 2: //      Oxigénio e luminosidade
                                float humi,temp;
                                float lu = (float) luz;
                                BSET(PORTC,RX);
                                BCLR(PORTB,SHDNV5);
                                lcd_init(LCD_DISP_ON);
                                stdout = &lcd_stream;
                                lu = (float) (lu*2.56*0.625*10000/1024);  // Conversão
da luminosidade
                                // Cálculo de O2 ganho do amplificador = 40
                                float o2aux = (float) o2;
                                o2aux = (float) ((o2aux*1000*2.56*1.78)/1024/40); //
Conversão O2
                                printf("Luz: %ilux\nO2: %2.1f%%",(int) lu,o2aux);
                        break;
                        case 3: // CO e CO2
                                BSET(PORTC,RX);
                                BCLR(PORTB,SHDNV5);
                                lcd_init(LCD_DISP_ON);
                                stdout = &lcd_stream;
                                float coaux = 0;
                                coaux = (float) (((co*2.56/1024)*10000/101)/1.374); //
Conversão CO
                                printf("CO: %3.1fppm\nCO2: %ippm",coaux,co2);
                        break;
                        case 4:
                                BSET(PORTC,RX);
                                BCLR(PORTB,SHDNV5);
                                lcd_init(LCD_DISP_ON);
                                stdout = &lcd_stream;
                                float bat = 0; // 9.98Kohm 13.67Kohm Ganho do divsidor =
0.578
                                bat = (float) bateria*2.56*1.990/1024;
                                printf("BAT: +%1.2fv",bat);
                        break;
                        default:  // Desliga display
                                if(segundos < 52){
                                        BSET(PORTB,SHDNV5); // Desliga alimentação de 5V
                                        BCLR(PORTC,RX);
                                }
                                displaymode=0;
                                lcd_init(LCD_DISP_OFF);
                        break;
                }
}

/***************************** INTERRUPÇÕES ***********************************/
/* TIMER2_OVF_vect - Função executada a cada interrupção do RTC = a cada segundo
 * Contabiliza os segundos e reinicia a 60s = 0
 * Segue uma ordem de amostragem dos vários sensores para obter o menor consumo
*/
ISR(TIMER2_OVF_vect){
        BCLR(PORTD,LED);                // Liga LED
```

```
        segundos++;
        if(segundos == 50){           // Aos 50s mede luminosidade antes de ligar o C20
                luz = medirTensao(0x00);// evitando ruídos do elevado consumo do sensor.
                btresponse = 0;       // Proteccão para não ficar sempre ligado

        }
        if(segundos == 51){           // Mede antes de ligar o sensor CO2
                BSET(PORTD,7);        // Liga amplificador operacional
                _delay_us(2);         // Espera que estabilize o seu funcionamento
                o2 = medirTensao(0x01); // Mede o valor do sensor de O2
                _delay_us(2);
                co = medirTensao(0x02); // Mede o valor do sensor de CO
                BCLR(PORTD,7);        // Desliga ampop para poupar energia, cerca de 2mA
                bateria = medirTensao(0x03); // Mede tensão nas baterias
                _delay_us(2);
        }
        if(segundos == 52){           // Aos 52s liga sensor de CO2 para este trabalhar
                BSET(PORTC,TX);       // nos próximos 7 segundos acumulando 8 valores
                BSET(PORTC,RX);       // Coloca os pinos da comunição série a nível 1
                BCLR(PORTB,SHDNV5);   // Liga CO2 pelo conversor DC-DC de 5V pelo mosfet
                lcd_init(LCD_DISP_OFF);
        }
        if(segundos == 59){           // Aos 59 segundos escuta a resposta do sensor
                co2 = medeCO2();
                BSET(PORTB,SHDNV5);   // Desliga alimentação do sensor no mosfet
                BCLR(PORTC,TX);
                BCLR(PORTC,RX);
        }
        if(segundos == 60){           // Ao fim de 1 minuto envia os dados pela rede
                segundos = 0;
                principal();
                displayInfo(displaymode); // Após enviar os dados / desligar o XBEE
                // volta a ligar o display evitando consumos máximos elevados
                // pois quando se liga o display liga o snesor de CO2
                // (usar num futuro display de 3.3V)
        }
        BSET(PORTD,LED);              // Desliga LED
}

/* USART1_RX_vect - Função executada aquando recepção de pedidos por bluetooth
 * Recebe o comando
 * Chama a função de resposta
*/
ISR(USART1_RX_vect){
        BCLR(PORTD,LED);
    letra = USART_Receive();
        repostaBLUETOOTH(letra);
}

/* USART0_RX_vect - Função executada aquando recepção de pedidos por XBee
 * Chama a função de resposta
*/
ISR(USART0_RX_vect){
    BCLR(PORTD,LED);
        respostaXBEE();
        BSET(PORTD,LED);
}

/* PCINT3_vect - Função executada cada vez que é pressionado o botão
 * Chama a função de resposta
*/
ISR(PCINT3_vect){
        if(bit_is_set(PIND,PD2)){     // Veririfca se o botão ainda está pressionado
        _delay_ms(50);                // Espera para não responder repetidamente
        if(!bit_is_set(PIND,PD5)){    // Se já libertou o botão
                displaymode++;
                displayInfo(displaymode);    // Apresenta o próximo conjunto de dados
                _delay_ms(200);
        }
        }else{
                BCLR(PCMSK3,PD2);             // Escuta interrupção de PD2 PCINT26 RXD1
                btresponse = 1;               // Permite recepção de pedidos bluetooth
        }
}
```

```
/************************** Comunicação USART ********************************/
// Funções inerentes à arquitectura AVR
// Função de incialização da UART 1
void USART_Init( unsigned int baud ){
        /* Set baud rate */
        UBRR1H = (unsigned char)(baud>>8);
        UBRR1L = (unsigned char)baud;
        /* rx interrupts enabled, Enable receiver and transmitter */
        UCSR1B = (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1);
        /* Set frame format: 8data, 2stop bit */
        UCSR1C = (0<<USBS1)|(3<<UCSZ10);
        /* Set divider  */
        UCSR1A |= (1<<U2X1);
}

// Função de transmissão de 1 caracter pela UART 1
void USART_Transmit( unsigned char letra ){
        /* Wait for empty transmit buffer */
        while ( !( UCSR1A & (1<<UDRE1)) );
        /* Put data into buffer, sends the data */
        UDR1 = letra;
}

// Função de recepção de 1 caracter pela UART 1
unsigned char USART_Receive( void ){
        /* Wait for data to be received */
        while ( !(UCSR1A & (1<<RXC1)) );
        /* Get and return received data from buffer */
        return UDR1;
}

static int UART_putChar(char c, FILE *stream)
{
        if(c == '\n') UART_putChar('\r',stream);
        USART_Transmit(c);
        return 0;
}

void USART0_Init( unsigned int ubrr ){
        UBRR0H = (unsigned char)(ubrr>>8);      // Velocidade de comunicação
        UBRR0L = (unsigned char)ubrr;           // UBRR0H 8bit com UBRR0L 8b = ubrr 18b
        UCSR0B = (1<<RXEN0)|(1<<TXEN0);         // Rx Tx enable
        UCSR0C = (3<<UCSZ00);
}

void USART0_Transmit( unsigned char data ){
        /* Wait for empty transmit buffer */
        while ( !( UCSR0A & (1<<UDRE0)) )
        ;
        /* Put data into buffer, sends the data */
        UDR0 = data;
}

unsigned char USART0_Receive( void ){
        /* Wait for data to be received */
        while ( !(UCSR0A & (1<<RXC0)) );
        /* Get and return received data from buffer */
        return UDR0;
}

/************************  FUNÇÕES PRÒPRIAS DO SHT15  **************************/
//-------------------------------------------------------------------------------
char s_write_byte(unsigned char value)
//-------------------------------------------------------------------------------
// writes a byte on the Sensibus and checks the acknowledge
{

  unsigned char i,error=0;
  for (i=0x80;i>0;i/=2)     //shift bit for masking
  { DDRC|=(1<<DATA);        // DATA como output
        if (i & value)
        {
                PORTC|=(1<<DATA);  //masking value with i , write to SENSI-BUS
        }
    else
```

```
        {
                PORTC&=~(_BV(DATA));
        }
        _delay_us(1);
    PORTC|=(1<<SCK);        //clk for SENSI-BUS
//    _nop_();_nop_();_nop_();//pulsewidth approx. 5 us
        _delay_us(5);
    PORTC&=~(_BV(SCK));
  }
  //release DATA-line
  DDRC|=(0<<DATA);
  PORTC|=(1<<DATA);
  DDRC&=~(_BV(DATA));


 // _delay_us(2);
  PORTC|=(1<<SCK);
  //_delay_us(2);           //clk #9 for ack
  if (bit_is_set(PINC,DATA))
  error=3;
  else error=0;             //check ack (DATA will be pulled down by SHT11)
  PORTC&=~(_BV(SCK));
  return error;             //error=1 in case of no acknowledge
}

//-----------------------------------------------------------------------------
char s_read_byte(unsigned char ack)
//-----------------------------------------------------------------------------
// reads a byte form the Sensibus and gives an acknowledge in case of "ack=1"
{
  unsigned char i,val=0;

  PORTC|=(1<<DATA);         //release DATA-line
  DDRC&=~(_BV(DATA));       //release DATA-line
  for (i=0x80;i>0;i/=2)     //shift bit for masking
  { PORTC|=(1<<SCK);
        _delay_us(2);           //clk for SENSI-BUS
    if(bit_is_set(PINC,DATA)) val=(val | i);        //read bit
    PORTC&=~(_BV(SCK));
        _delay_us(2);
  }
  DDRC|=(1<<DATA);
  if (ack){
        DDRC|=(1<<DATA);
        PORTC&=~(_BV(DATA));    //in case of "ack==1" pull down DATA-Line
  }

 PORTC|=(1<<SCK);           //clk #9 for ack
//  _nop_();_nop_();_nop_();//pulsewidth approx. 5 us
_delay_us(5);
  PORTC&=~(_BV(SCK));
  PORTC|=(1<<DATA);         //release DATA-line
  DDRC&=~(_BV(DATA));
  return val;
}

//-----------------------------------------------------------------------------
void s_transstart(void)
//-----------------------------------------------------------------------------
// generates a transmission start
//       _____         ________
// DATA:      |_______|
//           ___     ___
// SCK : ___|   |___|   |______
{
        DDRC|=(1<<DATA); // DATA pin as output
   PORTC|=(1<<DATA); PORTC&=~(_BV(SCK));//Initial state
//   _nop_();
_delay_us(2);
   PORTC|=(1<<SCK);
//   _nop_();
_delay_us(2);
   PORTC&=~(_BV(DATA));
//   _nop_();
_delay_us(2);
```

```
   PORTC&=~(_BV(SCK));
//   _nop_();_nop_();_nop_();
_delay_us(5);
   PORTC|=(1<<SCK);
//   _nop_();
_delay_us(2);
   PORTC|=(1<<DATA);
//   _nop_();
_delay_us(2);
   PORTC&=~(_BV(SCK));

}

//-----------------------------------------------------------------------------
void s_connectionreset(void)
//-----------------------------------------------------------------------------
// communication reset: DATA-line=1 and at least 9 SCK cycles followed transstart
//         _____________________________________________________         ________
// DATA:                                                      |_______|
//           _   _   _   _   _   _   _   _   _   _   _        ___     ___
// SCK : __| |_| |_| |_| |_| |_| |_| |_| |_| |_| |_| |______|   |___|   |______
{
  unsigned char i;
  DDRC|=(1<<DATA);                       // DATA pin as output
  PORTC|=(1<<DATA);
  PORTC&=~(_BV(SCK));           //Initial state
  for(i=0;i<9;i++)          //9 SCK cycles
  { PORTC|=(1<<SCK);
        _delay_us(3);
    PORTC&=~(_BV(SCK));
        _delay_us(1);
  }
  s_transstart();           //transmission start
}

//-----------------------------------------------------------------------------
char s_softreset(void)
//-----------------------------------------------------------------------------
// resets the sensor by a softreset
{
  unsigned char error=0;
  s_connectionreset();       //reset communication
  error+=s_write_byte(RESET1);//send RESET-command to sensor
  return error;              //error=1 in case of no response from the sensor
}

//-----------------------------------------------------------------------------
char s_read_statusreg(unsigned char *p_value, unsigned char *p_checksum)
//-----------------------------------------------------------------------------
// reads the status register with checksum (8-bit)
{
  unsigned char error=0;
  s_connectionreset(); //s_transstart(); //transmission start
  error=s_write_byte(STATUS_REG_R); //send command to sensor
  *p_value=s_read_byte(ACK);        //read status register (8-bit)
  *p_checksum=s_read_byte(noACK);   //read checksum (8-bit)
  return error;                     //error=1 in case of no response form sensor
}

//-----------------------------------------------------------------------------
char s_write_statusreg(unsigned char *p_value)
//-----------------------------------------------------------------------------
// writes the status register with checksum (8-bit)
{
  unsigned char error=0;
  s_transstart();                   //transmission start
  error+=s_write_byte(STATUS_REG_W);//send command to sensor
  error+=s_write_byte(*p_value);    //send value of status register
  return error;                     //error>=1 in case of no response from sensor
}

//-----------------------------------------------------------------------------
char s_measure(unsigned char *p_value,unsigned char *p_checksum,unsigned char mode)
//-----------------------------------------------------------------------------
// makes a measurement (humidity/temperature) with checksum
```

```
{
  unsigned error=0;
  unsigned int i;

  s_transstart();                   //transmission start
  switch(mode){                     //send command to sensor
    case TEMP   : error+=s_write_byte(MEASURE_TEMP); break;
    case HUMI   : error+=s_write_byte(MEASURE_HUMI); break;
    default     : break;
  }

  for (i=0;i<20000;i++)
  {
        _delay_us(100);
        if(bit_is_clear(PINC,DATA)) break;
  } //wait until sensor has finished the measurement
//_delay_us(2);
  if(bit_is_set(PINC,DATA)) error+=7;  // or timeout (~2 sec.) is reached
  *(p_value+1)  = s_read_byte(ACK);    //read the first byte (MSB)
  *(p_value)    = s_read_byte(ACK);    //read the second byte (LSB)
  *p_checksum   = s_read_byte(noACK);  //read checksum
  return error;
}

void calc_sth11(float *p_humidity ,float *p_temperature)
        //------------------------------------------------------------------------
        // calculates temperature [C] and humidity [%RH]
        // input : humi [Ticks] (12 bit)
        // temp [Ticks] (14 bit)
        // output: humi [%RH]
        // temp [C]
{
        const float C1=-4.0; // for 12 Bit
        const float C2= 0.0405; // for 12 Bit
        const float C3=-0.0000028; // for 12 Bit
        const float T1=0.01; // for 14 Bit @ 5V
        const float T2=0.00008; // for 14 Bit @ 5V
        float rh=*p_humidity; // rh: Humidity [Ticks] 12 Bit
        float t=*p_temperature; // t: Temperature [Ticks] 14 Bit
        float rh_lin; // rh_lin: Humidity linear
        float rh_true; // rh_true: Temperature compensated humidity
        float t_C; // t_C : Temperature [C]
        t_C=t*0.01 - 40; //calc. Temperature from ticks to [C]
        rh_lin=C3*rh*rh + C2*rh + C1; //calc. Humidity from ticks to [%RH]
        rh_true=(t_C-25)*(T1+T2*rh)+rh_lin; //calc. Temp compensated humidity [%RH]
        if(rh_true>100)rh_true=100; //cut if the value is outside of
        if(rh_true<0.1)rh_true=0.1; //the physical possible range
        *p_temperature=t_C; //return temperature [C]
        *p_humidity=rh_true; //return humidity[%RH]
}
/**************************  FIM FIRMWARE  ***********************************/
```

Atleta

```
/*****************************************************************************
* Nome do ficheiro : athlete.c                                               *
* Autor            : Filipe Santos                                           *
* Data             : 06/07/2009                                              *
* Descrição        : Este ficheiro é usado para o ATmega324                  *
*                  : comunicar com módulo ZigBee XBEE,                       *
*                  : módulo GPS LS20032, acelerómetro LIS3V2,                *
*                  : sensor bat cardíaco RMCM-01, Botão e LED                *
*****************************************************************************/

#define F_CPU 8000000UL // Velocidade do microcontrolador

/****************************** LIVRARIAS ************************************/
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <avr/power.h>
#include <avr/sleep.h>
#include <util/delay.h>

/********************* NOMENCLATURA E CONSTANTES *****************************/
#define XBEESLEEP       PC1  // Pino ligado ao pino sleep do XBEE
#define XBEESTATUS      PC2
#define LED             PC3  // Led de feedback
#define XBEERESET       PC0  // Pino para TX Síncrono Sensor CO2
#define CS              PB4  // Pino Chip Select para o Accelerometro
// Operações comuns
#define HI(x) (x>>8)
#define LO(x) (x&0xff)
#define BSET(p,b)   ((p) |=  (1<<b))   // Set bit
#define BCLR(p,b)   ((p) &= ~(1<<b))   // Clear bit

/********************* INICIALIZAÇÃO DE VARIÁVEIS ****************************/
volatile char letraaux,letraaux2; // Bytes auxiliares para as comunicações UART
volatile int segundos=0,bat=0,gpsi=0,sync=0,i=0,amostras=0,nmed=0,aux =
0,xbeei=0,xbeeit=0,syncx=0,rssi=0;
volatile int media = 0,totalrssi=0,contando = 0;
volatile char msgaux[90];  // Msg XBee: 17 para cabecalho + 72 carga +  1 checksum
volatile char msgGPS[42];  // Msg GPS: tamanho da msg GPGGA do protocolo NMEA
volatile char msgXBEE[36]; // Msg
volatile char msg[200];    // Vector de medidas de aceleração

// Declaração dos ponteiros das funções
void USART_Init( unsigned int baud );
void USART_Transmit( unsigned char data );
unsigned char USART_Receive( void );
void USART0_Init( unsigned int baud );
void USART0_Transmit( unsigned char data );
unsigned char USART0_Receive( void );
void SPI_MasterInit(void);
char SPI_MasterTransmit(volatile char cData);
void SPI_SlaveInit(void);
char SPI_SlaveReceive(void);
char ler_registo(volatile char registo);
void escrever_registo(volatile char registo, char valor);
void configTimer2(void);
void iniciaContador(void);
int paraContador(void);
unsigned char   checksum_1_byte (unsigned char * data_in,unsigned short
number_of_bytes_to_read, short array_index_counter);

int main()
{
        int segundosant=0,aux2=0,j=0,indice = 0; // Variáveis auxiliares para o programa
```

```
        char msgStart[17] =
{0x7e,0x00,0x56,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
// Estrutura inicial da msg para ZigBee API
// Comprimento 56 = 90 bytes
// Comprmento + tipo + endereço coordenador
        for(j=0;j<17;j++){ // Inicializa a msg a enviar com a primeira parte do formato
                msgaux[j] = msgStart[j];
        }
        for(j=17;j<90;j++){ // Garante que os valores são incializados a zero
                msgaux[j] = 0x00;
        }

        BSET(DDRC,LED);                 // Coloca o pino LED como output
        BSET(PORTC,LED);                // Desliga LED
        BSET(DDRC,XBEERESET);           // Coloca o pino Reset de Xbee como output
        BSET(PORTC,XBEERESET);          // Retira reset xbee
        BSET(DDRC,XBEESLEEP);           // Coloca o pino Sleep de Xbee como output
        BCLR(PORTC,XBEESLEEP);          // Retira sleep xbee
        BSET(PORTC,XBEESTATUS);         // Coloca pino a nível 1 para leitura do status
        BCLR(DDRC,XBEESTATUS);          // Coloca o pino Status de Xbee como input
        clock_prescale_set(0);          // uC a correr a 8MHz x 1
        USART_Init(51);                 // 9600 8bit + stopbit XBEE
        USART0_Init(8);                 // 57600 8bit + stopbit GPS
        configTimer2();                 // Configuração do RTC que controla o módulo
        _delay_ms(1000); // Espera que cristal estabilize
        // Configurar Timer1 para batimento
        DDRA=0x00;
        TCCR1A=0x00;
        TIMSK1=0x21;     // Activa interrupção ICP para parar contagem a cada Batimento
        TCCR1B=0x45;
        // Prepara pino com interrupcao para medir o dutty cycle do rssi
        BSET(PORTC,PC5);
        BCLR(DDRC,PC5);  // Coloca pino de interrupcao como input e a high
        // Configuracao - inicializacao do acelerometro
        BSET(DDRB,CS);
        SPI_MasterInit();
        BSET(PORTB,CS);
        BCLR(PORTB,CS);
        SPI_MasterTransmit(32);
        SPI_MasterTransmit(135);
        BSET(PORTB,CS);
        escrever_registo(0x21,0x80); // Configura Acelerómetro para gama 6 g

        BCLR(PORTC,LED);
        sei();
        for(;;)
        {
                if (TCNT1>16000) // Reinicializa timer1 dos batimentos cardíacos
                {                // Artificio para nao ultrapassar os 2 segundos = 30BPM
                        TCNT1=0x0000;
                        bat = 16000;
                }
                if (TCNT0>200) // Reinicializa timer0
                {              // Artificio para não medir RSSI superior a máximo
                        TCNT0=0x00;
                }
                // A CADA NOVO SEGUNDO COMPILA E ENVIA OS DADOS REGISTADOS
           if(segundosant < segundos){

                // 1 - Primeira mensagem GPS+Bat+Vel+Dist+RSSI - TIPO A0
                msgaux[17] = 0xA0;
                // Selecciona e converte dados gps para a mensagem
                aux2 = ((int) msgGPS[18] - 0x30)*1000;
                aux2 += ((int) msgGPS[19] - 0x30)*100;
                aux2 += ((int) msgGPS[20] - 0x30)*10;
                aux2 += ((int) msgGPS[21] - 0x30)*1;
                msgaux[18] = HI(aux2);
                msgaux[19] = LO(aux2);
                aux2 = ((int) msgGPS[23] - 0x30)*1000;
                aux2 += ((int) msgGPS[24] - 0x30)*100;
                aux2 += ((int) msgGPS[25] - 0x30)*10;
                aux2 += ((int) msgGPS[26] - 0x30)*1;
                msgaux[20] = HI(aux2);
                msgaux[21] = LO(aux2);
                msgaux[22] = msgGPS[28];
```

```
                // e recebe longitude
                aux2 = ((int) msgGPS[30] - 0x30)*10000;
                aux2 += ((int) msgGPS[31] - 0x30)*1000;
                aux2 += ((int) msgGPS[32] - 0x30)*100;
                aux2 += ((int) msgGPS[33] - 0x30)*10;
                aux2 += ((int) msgGPS[34] - 0x30)*1;
                msgaux[23] = HI(aux2);
                msgaux[24] = LO(aux2);
                aux2 = ((int) msgGPS[36] - 0x30)*1000;
                aux2 += ((int) msgGPS[37] - 0x30)*100;
                aux2 += ((int) msgGPS[38] - 0x30)*10;
                aux2 += ((int) msgGPS[39] - 0x30)*1;
                msgaux[25] = HI(aux2);
                msgaux[26] = LO(aux2);
                msgaux[27] = msgGPS[41];
                // Regista batimento cardiaco
                bat=div(bat,100).quot;
                bat=div(4688,bat).quot;
                msgaux[28] = bat;
                // Vel e Distancia via acelerómetro opcional
                // OPTAR por calculo da velocidade e distancia no uC ou deixar no PC
                msgaux[29] = 0x00; // Velocidade a partir de acc (opção)
                msgaux[30] = 0x00; // Distância a partir do acc (opção)
                // Endereços de torres e respectiva potência
                // a partir deste momento cancela recepcao de rssi – para não interferir
                BCLR(UCSR1B,RXEN1);
                for(j=31;j<67;j++){
                        msgaux[j] = msgXBEE[j-31]; // Junta torres e RSSI
                }
                // Reinicia processo de aquisicao de 4 distancias por RSSI (endereco +
rssi)
                xbeei = 0;
                xbeeit = 0;
                syncx = 0;
                rssi = 0;
                totalrssi = 0;
                contando = 0;
                for(j=0;j<36;j++){
                        msgXBEE[j] = 0x00; // Limpa vesctor por precaução para próxima
                }
                // Reactiva recepcao de msg dos nós de referência XBEE
                BSET(UCSR1B,RXEN1);
                msgaux[88] = indice;
                // Calcula o checksum da msg
                msgaux[89] = checksum_1_byte((unsigned char*) &msgaux,86,3);

                for(j=0;j<90;j++){ // Envio da 1 mensagem
                        USART_Transmit(msgaux[j]);
                        _delay_us(175); //
                }
                // 2 – Segunda mensagem com 1/3 da Acc - TIPO A1
                msgaux[17] = 0xA1;
                for(j=0;j<70;j++){
                        msgaux[j+18] = msg[j]; // Carrega a parte 1 de 3 da aceleração
                }
                msgaux[88] = indice;
                // Calcula o checksum da msg
                msgaux[89] = checksum_1_byte((unsigned char*) &msgaux,86,3);
                _delay_ms(50); // Pesquisar quanto tempo devo esperar para o envio da
primeira msg ou esperar pela resposta de sucesso da rede XBEE para enviar a segunda msg
                // Envio da 2 mensagem
                for(j=0;j<90;j++){
                        USART_Transmit(msgaux[j]);
                        _delay_us(175); // Remover
                }
                // 3 – Terceira mensagem com 2/3 da Acc - TIPO A2
                msgaux[17] = 0xA2;
                for(j=0;j<70;j++){
                    msgaux[j+18] = msg[j+70]; // Carrega a parte 2 de 3 da aceleração
                }
                msgaux[88] = indice;
                // Calcula o checksum da msg
                msgaux[89] = checksum_1_byte((unsigned char*) &msgaux,86,3);
                _delay_ms(50);
                for(j=0;j<90;j++){
```

```
                        USART_Transmit(msgaux[j]); // Envio da 3 mensagem
                        _delay_us(175); // Remover
                }
                // 4 – Quarta mensagem com 3/3 da Acc - TIPO A3
                msgaux[17] = 0xA3;
                for(j=0;j<70;j++){
                    msgaux[j+18] = msg[j+140]; // Carrega a parte 3 de 3 da aceleração
                }
                msgaux[88] = indice;
                // Calcula o checksum da msg
                msgaux[89] = checksum_1_byte((unsigned char*) &msgaux,86,3);

                _delay_ms(50); // Pesquisar ''
                for(j=0;j<90;j++){
                        USART_Transmit(msgaux[j]); // Envio da 4 mensagem
                        _delay_us(175); // Remover
                }
                // Colocando a variável auxiliary igual ao segundo actual – fará com o
ciclo espere pelo próximo Segundo. Entretanto o microcontrolador realiza as outras
tarefas definidas pelas suas interrupções
                segundosant = segundos;
                // Reactiva recepcao de GPS (é cancelado após receber uma localização)
                BSET(UCSR0B,RXEN0);
                indice++; // Indice de sequencia de conjunto de msg's A0+A1+A2+A3
                if(indice == 256){ // Não foi utilizado (fica a ideia)
                        indice =0;
                }
                }
                if(segundosant == 0){ // no início ou em caso de overflow
                        segundosant = segundos;
                }

        }
}

/********************************* INTERRUPÇÕES *********************************/
// Interrupção do RTC
ISR(TIMER2_OVF_vect)
{
        TCNT2 = 0xBE; // Reinicia número base do contador para “dispara” aos 2 ms
        aux = ler_registo(171)<<8;
        aux += ler_registo(170); // Junta byte mais significativo com LSB
        if(aux > 0xF000){
                aux = -(65535-aux); // Converte para decimal negativo se for o caso
        }
        media+=aux;// Soma até 5 amostras de 2ms
        nmed++;
        if(nmed == 5){       // Realiza a média das 5 amostras
                nmed = 0;
                media = media/5;
                msg[amostras] = HI(media); // Resgita MSB e LSB no vector de acelerações
                msg[amostras+1] = LO(media);
                amostras+=2;
        }// Período registado de 10 ms = 100 amostras por segundo => x (MSB+LSB) = 200B
        if(amostras==200){
                segundos++;
                amostras = 0;
        }
}
// Interrupção de overflow do contador do batimento cardíaco
ISR(TIMER1_OVF_vect){
        TCNT1=0x0000;
}
// Interrupção de overflow do contador da largura do sinal PWM que corresponde ao RSSI
ISR(TIMER0_OVF_vect){
        TCNT0 = 0x00;
}
// Caso receba um impulso cardíaco regista o tempo decorrido do anterior
ISR(TIMER1_CAPT_vect){
        bat=ICR1;
        TCNT1=0x0000;
}
// Recepção do GPS
ISR(USART0_RX_vect)
{       letraaux = USART0_Receive();
```

```
        if((letraaux == '$') && (sync == 0)){
                sync = 1; // Pré-sincroniza
        }
        if(sync == 1){
                msgGPS[gpsi] = letraaux;
                // Verifica se é a msg pretendida $GCGGA
                if((gpsi == 5) && ((msgGPS[5] != 'A') || (msgGPS[4] != 'G') ) ){
                        // Se não volta a espera por sincronismo do $
                        gpsi = 100;
                }
                gpsi++;
                if(gpsi > 41){
                        if(gpsi ==42){ // Recebeu com sucesso
                                BCLR(UCSR0B,RXEN0); // Pára a  recepcao do GPS
                        }
                        gpsi = 0;
                        sync = 0;

                }
        }
}
// Interrupção do pino que conta o dutty cicle do sinal PWM do RSSI
// Dispara quando o sinal sobe-> inicia contagem. Dispara quando desce-> pára o contador
ISR(PCINT2_vect){
        if(bit_is_set(PINC,PC5)){
                // Inicia contador
                contando = 1;
                iniciaContador();
        }
        if(!bit_is_set(PINC,PC5) && contando){
                // Para contador
                rssi += paraContador();
                totalrssi++;
                contando = 0;
                // Desliga a interrupcao
                BCLR(PCICR,PCIE2); // Disable PCMSK2 pin change interrupts
                BCLR(PCMSK2,PC5);  // Tell pin change mask to listen to PD5 PCINT29
        }
}
// Inicia contador do PWM do nível de RSSI
void iniciaContador(void){
        TCNT0 = 0x00;
        TCCR0A=0x00;
        TIMSK0=0x21;
        TCCR0B=0x02;
}
// Para contador e devolve resultado
int paraContador(void){
        TCCR0B=0x00;
        return TCNT0;
}
// Interrupção de recepção de msg do XBEE
// Verifica se é uma msg dos nós de referência e ordena contagem do periodo do sinal
RSSI
// Junta num vector os endereços dos nós de referência escutados e valor de RSSI
respectivo
ISR(USART1_RX_vect) {
        letraaux2 = USART_Receive();
        if((letraaux2 == 0x7E) && (syncx == 0)){ // Verifica se é o inicio de uma msg
                syncx = 1; // Se não volta a espera pelo inicio
                xbeei = 0;
        }
        if(syncx == 1){ // Se já destectou o inicio de uma msg
                xbeei++;
                if((xbeei == 4)&&(letraaux2 != 0x90)){// Verifica se é uma msg de dados
                     // Se não volta a sincronizar (evita registar msg de sucesso 0x88)
                        xbeei = 0;
                        syncx=0;
                }
                if(xbeei >4){
                        msgXBEE[xbeeit] = letraaux2;
                        // Liga interrupcao que mede dutty cicle do rssi
                        if(xbeei==6 || xbeei==8 || xbeei==10){
                                BSET(PCICR,PCIE2); // Activa interrupção
                                BSET(PCMSK2,PC5); //  no pino PD5 PCINT29
```

```
                        }
                        xbeeit++;
                       // Deixa registar até 12 bytes = endereço do remetente
                        if(xbeei == 12){
                                BSET(PORTC,LED);
                                if(rssi > 0 && totalrssi > 0){ // Regista RSSI respec.
                                        msgXBEE[xbeeit] = rssi/totalrssi;
                                }else{// Verifica gama do valor medido
                                        if(bit_is_set(PORTC,PC5)){
                                                msgXBEE[xbeeit] = 200;
                                        }else{
                                                msgXBEE[xbeeit] = 0;
                                        }
                                }
                                xbeeit++;
                                xbeei = 0;
                                syncx = 0;
                                rssi = 0;
                                totalrssi = 0;
                                if(xbeeit==35){
                                        BCLR(UCSR1B,RXEN1); // Pára recepcao do XBEE
                                }
                        }
                }
        }
}

/*********************** Comunicação USART ***************************/

void USART_Init( unsigned int baud ) {
          UBRR1H = (unsigned char)(baud>>8);        // Velocidade
          UBRR1L = (unsigned char)baud;             //
          UCSR1B = (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1); // Interrupção de RX TX

          UCSR1C = (3<<UCSZ10);
}

void USART_Transmit( unsigned char letra ) {
          /* Wait for empty transmit buffer */
          while ( !( UCSR1A & (1<<UDRE1)) );
          /* Put data into buffer, sends the data */
          UDR1 = letra;
}

unsigned char USART_Receive( void ) {
          /* Wait for data to be received */
          while ( !(UCSR1A & (1<<RXC1)) );
          /* Get and return received data from buffer */
          return UDR1;
}

void USART0_Init( unsigned int ubrr ){
          UBRR0H = (unsigned char)(ubrr>>8); // Vecolidade comunicação
          UBRR0L = (unsigned char)ubrr;      //
          UCSR0B = (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);  // Interrupção de Rx Tx enable
          UCSR0C = (3<<UCSZ00);
}

void USART0_Transmit( unsigned char data )
{
          /* Wait for empty transmit buffer */
          while ( !( UCSR0A & (1<<UDRE0)) );
          /* Put data into buffer, sends the data */
          UDR0 = data;
}

unsigned char USART0_Receive( void ) {
          /* Wait for data to be received */
          while ( !(UCSR0A & (1<<RXC0)) );
          /* Get and return received data from buffer */
          return UDR0;
}

/*********************** FIM Comunicação USART ************************/
```

```
/****************** FUNCOES AUXILIARES ******************************/

/* checksum_1_byte - Calcula o byte de checksum para a mensagem API do XBEE.
 * Soma todos os bytes, excepção dos bytes start e lenght = 3 primeiros bytes
 * Subtrai o resultado de 8 bits a 255 = 0xFF (hexadecimal). Retorna resultado
*/
unsigned char checksum_1_byte (unsigned char * data_in,unsigned short
number_of_bytes_to_read,short array_index_counter){
        unsigned char   checksum;
        checksum = 0;
    number_of_bytes_to_read+=array_index_counter;
    while (array_index_counter != number_of_bytes_to_read){
      array_index_counter++;
      checksum += data_in[array_index_counter - 1];
        }
        checksum = 0xFF-checksum;
    return checksum;
}

/* configTimer2 - Configura o timer do 2 do uC para trabalhar como um Real Time Clock.
 * Soma todos os bytes, excepção dos bytes start e lenght = 3 primeiros bytes
 * Períodos para diferentes prescalers: 0x05 = 1s, 0x06 = 2s, 0x04 = 0,5s
*/
void configTimer2(void){
        ASSR  =0x20; // Activa modo assincrono
        TIMSK2=0x00; // Desactiva todas as interrupções para alteração dos registos
        OCR2A =0x00;
        TCNT2 =0xBE;  // inicia a contagem a 190 para levar 66 a chegar a 256 +-= 2ms
        TCCR2A=0x00; // Configura modo normal
        TIMSK2=0x01; // Activa interrupção de overflow
        TCCR2B=0x01; // Inicia o timer com prescaler de 1 T= 0.0078125s
}

/************************    Comunicação SPI    ************************/
// usado para comuncia com o acelerómetro
void SPI_MasterInit(void) {
         /* Set MOSI and SCK output, all others input */
         DDRB = (1<<DDB5)|(1<<DDB7)|(1<<DDB4);
         /* Enable SPI, Master, set clock rate fck/16 */
         SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}
char SPI_MasterTransmit(volatile char cData) {
         /* Start transmission */
         SPDR = cData;
         /* Wait for transmission complete */
         while(!(SPSR & (1<<SPIF)));
         return SPDR;
}

char ler_registo(volatile char registo) {
        unsigned char valor;
        SPI_MasterInit();
        BSET(PORTB,CS);
        BCLR(PORTB,CS);
        SPI_MasterTransmit(registo);
        valor = SPI_MasterTransmit(0);
        BSET(PORTB,CS);
        return valor;
}
void escrever_registo(volatile char registo, char valor) {
        SPI_MasterInit();
        BSET(PORTB,CS);
        BCLR(PORTB,CS);
        SPI_MasterTransmit(registo);
        SPI_MasterTransmit(valor);
        BSET(PORTB,CS);
}
```

Torre

```
/*****************************************************************************
* Nome do ficheiro : athlete.c                                               *
* Autor            : Filipe Santos                                           *
* Data             : 06/07/2009                                              *
* Descrição        : Este ficheiro é usado para o ATmega168                  *
*                  : comunicar com módulo ZigBee XBEE e LED                  *
*                  : (módulo GPS LS20032 – não implementado apenas com ATmega324)
*****************************************************************************/

#define F_CPU 8000000UL // Velocidade do microcontrolador

/****************************** LIVRARIAS ***********************************/
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <avr/power.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <util/delay.h>

/********************* NOMENCLATURA E CONSTANTES ************************/
#define XBEESLEEP   PD2             // Pino ligado ao pino sleep do XBEE
#define XBEESTATUS  PD4
#define LED         PB2             // Led de feedback
#define XBEECTS     PD3
// Funções comuns
#define HI(x) (x>>8)
#define LO(x) (x&0xff)

#define BSET(p,b)   ((p) |=  (1<<b))   // Set bit
#define BCLR(p,b)   ((p) &= ~(1<<b))   // Clear bit


/********************* VARIÁVEIS GLOBAIS ****************************/
char letraaux;                  // Byte auxiliar para as comunicações UART
volatile int segundos=0,bat=0,gpsi=0,sync=0,i=0,amostras=0,nmed=0,aux = 0;
// Variável global para segundos e restantes variáveis auxiliares ao programa;
volatile int
segundosant=0,segundosant2=0,xbeei=0,xbeeit=0,syncx=0,rssi=0,letraaux2=0x00,syncBroadcas
t=0;                        // Veriaveis auxiliares para programa e para Sync de Torres
volatile int media = 0,totalrssi=0,contando = 0;
volatile char msgaux[90];  // 17 para cabecalho + 72 carga +  1 checksum
volatile char msgGPS[42];  // vector de medidas <- Dummy só para versão com ATmega324
volatile char msgXBEE[36]; // Vector 4 enderecos de torres + RSSI respectivo

// Declaração das funções
void USART_Init( unsigned int baud );
void USART_Transmit( unsigned char data );
unsigned char USART_Receive( void );
void iniciaContador(void);
int paraContador(void);


void configTimer2(void);


unsigned char   checksum_1_byte (unsigned char * data_in,unsigned short
number_of_bytes_to_read, short array_index_counter);

int main()
{
        int aux2=0,j=0,guarda = 0;
        char msgStart[17] =
{0x7e,0x00,0x10,0x10,0x00,0x00,0x13,0xA2,0x00,0x40,0x31,0x4A,0xDF,0xFF,0xFE,0x01,0x00};
```

```
  // Estrutura inicial da msg para ZigBee API
  // Comprimento 56 = 90 bytes
  // Comprmento + tipo + endereço coordenador

// Inicializacao da msg completa a enviar
for(j=0;j<17;j++){
        msgaux[j] = msgStart[j];
}
for(j=17;j<90;j++){
        msgaux[j] = 0x00;
}
for(j=0;j<42;j++){
        msgGPS[j] = 0x00; // Só funciona se for ATMega324

}

BSET(DDRB,LED);      // Coloca o pino LED como output
BCLR(PORTB,LED);     // Liga LED
BSET(PORTD,PD5);
BCLR(DDRD,PD5);      // Coloca pino de interrupcao como input e a high
BSET(DDRD,XBEESLEEP);         // Coloca o pino Sleep de Xbee como output
BCLR(PORTD,XBEESLEEP);        // Retira sleep xbee
BSET(PORTD,XBEESTATUS);       //
BCLR(DDRD,XBEESTATUS);        // Coloca o pino Status de Xbee como input
BSET(PORTD,XBEECTS);
BCLR(DDRD,XBEECTS);

clock_prescale_set(0);        // uC a correr a 8MHz
USART_Init(51);               // 57600 8bit + stopbit XBEE
configTimer2();               // Conigura RTC
_delay_ms(1000);              // espera que cristal estabilize
BSET(PORTB,LED);     // Desliga led
sei();
for(;;)
{
        if (TCNT0>200){ // Reinicializar timer1
                TCNT0=0x00;
        }           // Artificio para nao ultrapassar os máximo de RSSI
        if(segundosant2 < segundos){
                guarda = 1;
                // 1 - Primeira mensagem GPS+RSSI - TIPO B0 para o coordenador
                msgaux[2] = 0x3D;
                msgaux[5] = 0x00;
                msgaux[6] = 0x00;
                msgaux[7] = 0x00;
                msgaux[8] = 0x00;
                msgaux[9] = 0x00;
                msgaux[10] = 0x00;
                msgaux[11] = 0x00; // Endereço do coordenador
                msgaux[12] = 0x00;
                msgaux[13] = 0xFF;
                msgaux[14] = 0xFE;
                msgaux[15] = 0x00;
                msgaux[17] = 0xB0;
                // Processa GPS (Só com o GPS presente – com uC ATMega 324)
                aux2 = ((int) msgGPS[18] - 0x30)*1000;
                aux2 += ((int) msgGPS[19] - 0x30)*100;
                aux2 += ((int) msgGPS[20] - 0x30)*10;
                aux2 += ((int) msgGPS[21] - 0x30)*1;
                msgaux[18] = 0x00;//HI(aux2);
                msgaux[19] = 0x00;//LO(aux2);
                aux2 = ((int) msgGPS[23] - 0x30)*1000;
                aux2 += ((int) msgGPS[24] - 0x30)*100;
                aux2 += ((int) msgGPS[25] - 0x30)*10;
                aux2 += ((int) msgGPS[26] - 0x30)*1;
                msgaux[20] = 0x00;//HI(aux2);
                msgaux[21] = 0x00;//LO(aux2);
                msgaux[22] = 0x00;//msgGPS[28];
                // e recebe longitude
                aux2 = ((int) msgGPS[30] - 0x30)*10000;
                aux2 += ((int) msgGPS[31] - 0x30)*1000;
                aux2 += ((int) msgGPS[32] - 0x30)*100;
                aux2 += ((int) msgGPS[33] - 0x30)*10;
                aux2 += ((int) msgGPS[34] - 0x30)*1;
                msgaux[23] = 0x00;//HI(aux2);
```

```
                        msgaux[24] = 0x00;//LO(aux2);
                        aux2 = ((int) msgGPS[36] - 0x30)*1000;
                        aux2 += ((int) msgGPS[37] - 0x30)*100;
                        aux2 += ((int) msgGPS[38] - 0x30)*10;
                        aux2 += ((int) msgGPS[39] - 0x30)*1;
                        msgaux[25] = 0x00;//HI(aux2);
                        msgaux[26] = 0x00;//LO(aux2);
                        msgaux[27] = 0x00;//msgGPS[41];


                        // Junta endereço + RSSI das outras torres de referência
                        // A partir deste momento cancela recepcao de rssi
                        BCLR(UCSR0B,RXEN0);
                        for(j=28;j<64;j++){
                                msgaux[j] = msgXBEE[j-28];
                        }
                        // Reinicia processo de aquisicao de 4 distancias por rssi
                        xbeei = 0;
                        xbeeit = 0;
                        syncx = 0;
                        rssi = 0;
                        totalrssi = 0;
                        contando = 0;
                        for(j=0;j<36;j++){
                                msgXBEE[j] = 0x00;
                        }
                        // Reactiva recepção de torres XBEE
                        BSET(UCSR0B,RXEN0);
                        // Cálculo do checksum
                        msgaux[64] = checksum_1_byte((unsigned char*) &msgaux,61,3);

                        BCLR(PORTB,LED); // acende led
                        //BCLR(PORTD,XBEESLEEP);                // Retira sleep xbee
                        _delay_ms(15);
                        // Envio da 1 mensagem - Mensagem para base a indicar posição
actual (opção de localizar atraves de RSSI das outras torres)
                        for(j=0;j<65;j++){
                                USART_Transmit(msgaux[j]);
                        }
                        _delay_ms(1); // Esperra envio do último byte
                        BSET(PORTB,LED);
                        segundosant2 = segundos+15; // Envia para o PC de 8 em 8 s
                }

                // A cada novo segundo processa e envia os dados registados
                if(segundosant < segundos){
                        _delay_ms(15);
                        for(j=0;j<17;j++){
                                msgaux[j] = msgStart[j];
                        }
                        // 2 - Segunda mensagem envio do broadcast de referência para os
atletas Tipo 0xB1 + segundos
                        msgaux[2] = 0x10;
                        // endereco de broadcast FFFF
                        msgaux[5] = 0x00;
                        msgaux[6] = 0x00;
                        msgaux[7] = 0x00;
                        msgaux[8] = 0x00;
                        msgaux[9] = 0x00;
                        msgaux[10] = 0x00;
                        msgaux[11] = 0xFF;
                        msgaux[12] = 0xFF;
                        msgaux[13] = 0xFF;
                        msgaux[14] = 0xFE;
                        msgaux[15] = 0x01;
                        msgaux[17] = 0xB1;
                        msgaux[18] = LO(segundos);
                        // Cálculo de checksum
                        msgaux[19] = checksum_1_byte((unsigned char*) &msgaux,16,3);
                        //_delay_ms(500); // Pesquisar quanto tempo devo esperar ou
esperar pela resposta de sucesso + RAND
                        BCLR(PORTB,LED);
                        // Envio da 2 mensagem
                        for(j=0;j<20;j++)
                        {
```

```
                                USART_Transmit(msgaux[j]);
                        }
                        _delay_ms(1);
                        BSET(PORTB,LED);
                        segundosant = segundos+15; // Envia broadcast de 8 em 8 segundos
                }
                if(segundosant == 0 || segundosant > segundos+36){
                        segundosant = segundos;
                        segundosant2 = segundos;
                }
        }
}


/********************** INTERRUPÇÔES ************************/
// Interrupção do RTC
ISR(TIMER2_OVF_vect) {
                segundos++;
}

ISR(TIMER0_OVF_vect){
        TCNT0 = 0x00;
}

// Interrupção do pino que conta o dutty cicle do sinal PWM do RSSI
// Dispara quando o sinal sobe-> inicia contagem. Dispara quando desce-> pára o contador
ISR(PCINT2_vect){
        if(bit_is_set(PINC,PC5)){
                // Inicia contador
                contando = 1;
                iniciaContador();
        }
        if(!bit_is_set(PINC,PC5) && contando){
                // Para contador
                rssi += paraContador();
                totalrssi++;
                contando = 0;
                // Desliga a interrupcao
                BCLR(PCICR,PCIE2); // Disable PCMSK2 pin change interrupts
                BCLR(PCMSK2,PC5);  // Tell pin change mask to listen to PD5 PCINT29
        }
}
// Inicia contador do PWM do nível de RSSI
void iniciaContador(void){
        TCNT0 = 0x00;
        TCCR0A=0x00;
        TIMSK0=0x21;
        TCCR0B=0x02;
}
// Para contador e devolve resultado
int paraContador(void){
        TCCR0B=0x00;
        return TCNT0;
}
// Interrupção de recepção de msg do XBEE
// Verifica se é uma msg dos nós de referência e ordena contagem do periodo do sinal
RSSI
// Junta num vector os endereços dos nós de referência escutados e valor de RSSI
respectivo
ISR(USART1_RX_vect) {
        letraaux2 = USART_Receive();
        if((letraaux2 == 0x7E) && (syncx == 0)){ // Verifica se é o inicio de uma msg
                syncx = 1; // Se não volta a espera pelo inicio
                xbeei = 0;
        }
        if(syncx == 1){ // Se já destectou o inicio de uma msg
                xbeei++;
                if((xbeei == 4)&&(letraaux2 != 0x90)){// Verifica se é uma msg de dados
                     // Se não volta a sincronizar (evita registar msg de sucesso 0x88)
                        xbeei = 0;
                        syncx=0;
                }
                if(xbeei >4){
                        msgXBEE[xbeeit] = letraaux2;
```

```
                        // Liga interrupcao que mede dutty cicle do rssi
                        if(xbeei==6 || xbeei==8 || xbeei==10){
                                BSET(PCICR,PCIE2); // Activa interrupção
                                BSET(PCMSK2,PC5); //  no pino PD5 PCINT29
                        }
                        xbeeit++;
                       // Deixa registar até 12 bytes = endereço do remetente
                        if(xbeei == 12){
                                BSET(PORTC,LED);
                                if(rssi > 0 && totalrssi > 0){ // Regista RSSI respec.
                                        msgXBEE[xbeeit] = rssi/totalrssi;
                                }else{// Verifica gama do valor medido
                                        if(bit_is_set(PORTC,PC5)){
                                                msgXBEE[xbeeit] = 200;
                                        }else{
                                                msgXBEE[xbeeit] = 0;
                                        }
                                }
                                xbeeit++;
                                xbeei = 0;
                                syncx = 0;
                                rssi = 0;
                                totalrssi = 0;
                                if(xbeeit==35){
                                        BCLR(UCSR1B,RXEN1); // Pára recepcao do XBEE
                                }
                        }
                }
        }
}

/************************ Comunicação USART ****************************/

void USART_Init( unsigned int ubrr ) {
         UBRR0H = (unsigned char)(ubrr>>8);
         UBRR0L = (unsigned char)ubrr;
         UCSR0B = (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0); // Rx Tx enable (1<<RXCIE0)|
         UCSR0C = (3<<UCSZ00);
}

void USART_Transmit( unsigned char data ) {
         /* Wait for empty transmit buffer */
         while ( !( UCSR0A & (1<<UDRE0)) );
         while(!bit_is_set(PORTD,XBEECTS));
         /* Put data into buffer, sends the data */
         UDR0 = data;
}

unsigned char USART_Receive( void ) {
         /* Wait for data to be received */
         while ( !(UCSR0A & (1<<RXC0)) );
         /* Get and return received data from buffer */
         return UDR0;
}

/************************ FIM Comunicação USART ************************/

/******************* FUNCOES AUXILIARES *******************************/

/* checksum_1_byte - Calcula o byte de checksum para a mensagem API do XBEE.
 * Soma todos os bytes, excepção dos bytes start e lenght = 3 primeiros bytes
 * Subtrai o resultado de 8 bits a 255 = 0xFF (hexadecimal). Retorna resultado
*/
unsigned char checksum_1_byte (unsigned char * data_in,unsigned short
number_of_bytes_to_read,short array_index_counter){
        unsigned char   checksum;
        checksum = 0;
    number_of_bytes_to_read+=array_index_counter;
    while (array_index_counter != number_of_bytes_to_read){
      array_index_counter++;
      checksum += data_in[array_index_counter - 1];
        }
        checksum = 0xFF-checksum;
    return checksum;
}
```

```
/* configTimer2 - Configura o timer do 2 do uC para trabalhar como um Real Time Clock.
 * Soma todos os bytes, excepção dos bytes start e lenght = 3 primeiros bytes
 * Períodos para diferentes prescalers: 0x05 = 1s, 0x06 = 2s, 0x04 = 0,5s
*/
void configTimer2(void){
        ASSR  =0x20; // Activa modo assincrono
        TIMSK2=0x00; // Desactiva todas as interrupções para alteração dos registos
        OCR2A =0x00;
        TCNT2 =0xBE;  // inicia a contagem a 190 para levar 66 a chegar a 256 +-= 2ms
        TCCR2A=0x00; // Configura modo normal
        TIMSK2=0x01; // Activa interrupção de overflow
        TCCR2B=0x04; // Inicia o timer com prescaler de 1 T= 0.5s
}
```

ANEXO I – Código fonte da aplicação para a Internet

Extracto do ficheiro "index.php"

```
<!-- Formulario de consulta -->
<form id="form" action="graph1.php" method="get" target="graph">
Start <input type="text" name="data" value="<?php echo date("d-m-Y");?>" size="9"
       style="text-align:center;border:1px solid #606060;">
End <input type="text" name="enddata" value="<?php echo date("d-m-Y");?>" size="9"
       style="text-align:center;border:1px solid #606060;">
<select id="sensor" name="sensor" onchange="submit();">    <!-- Lista de opções -->
       <option value="0" selected>Humidity</option>
       <option value="1" >Temperature</option>
       <option value="2" >Light</option>
       <option value="3" >Volt</option>
       <option value="4" >O2</option>
       <option value="5" >CO</option>
       <option value="6" >CO2</option>
</select>
<input type="submit" value=">>" style="border:1px solid gray;" /><!-- Botão pesquisa -->
</form>
```

Ficheiro "graph1.php"

```
<?php
// Dados de acesso ao servidor que contŽm a base de dados do sistema
$ipServidor = "PCWINDOWS";                  // Nome ou IP do servidor
$bdDestino = "sensoresdb";                  // Nome da base de dados
$nomeTabelaDB = "ambientmeasurement";     // Tabela
require('mysql4.php');                      // Carrega Livraria de acesso ‡ base de dados
MySQL
$dbObject = new sql_db($ipServidor, 'root', '', $bdDestino, false);    // liga-
se ^ base de dados
$chs = array(0=>'%',1=>'C',2=>'Lx',3=>'V',4=>'%',5=>'ppm',6=>'ppm'); // Unidades dos gr‡
ficos
$campoTab = array(0=>'humidity',1=>'temperature',2=>'light',3=>'voltage',
                  4=>'o2',5=>'co',6=>'co2'); // Campos da base de dados
$campoTabNome = array(0=>'Humidity (%)',1=>'Temperature (.C)',2=>'Light (Lux)',3=>'Volta
ge (v)',
                      4=>'O2 (%)',5=>'CO (ppm)',6=>'CO2 (ppm)');  // Nomes dos par‰metro
s

$data = date("d-m-
Y");        // Recebe data de inicio dos dados via método GET do formulário
if(isset($_GET['data'])){
    $data = $_GET['data'];
}
$enddata = date("d-m-Y");    // Recebe data de fim dos dados
if(isset($_GET['enddata'])){
    $enddata = $_GET['enddata'];
}
$modulo = 1;    // Recebe ID do módulo de monitoriza‹o caso seja utilizado mais módulos
if(isset($_GET['amid'])){
    $modulo = $_GET['amid'];
}
list($d,$m,$Y) = explode("-",$data);    // Separa campos da data de início
list($de,$me,$Ye) = explode("-",$enddata);
$monthList = array(array('m'=>$m,'Y'=>$Y));
if($me > $m || $Ye>$Y){    // Constrói uma lista de meses/ano de dados a consultar
    $maux = $m;
    $Yaux = $Y;
    while($Yaux.$maux!=$Ye.$me){
        $maux++;
        if($maux == 13){
            $maux = 1;
            $Yaux++;
        }
        if(strlen($maux)<2)$maux="0".$maux;
        $monthList[] = array('m'=>$maux,'Y'=>$Yaux);
```

```
    }
}elseif(($m > $me) && $Y >=$Ye){    // Confirma que as datas inseridas estão correctas
    echo "End date is smaller than start date!";
    exit;
}


$diai = $d;
$diaf = $de;
$anoa = $Y;
$mesa = $m;
// calculo da data/hora inicial de pesquisa
$horai = date("Y-m-d H:i:s.001",mktime(0,0,1,$mesa,$diai,$anoa));
$horaiquinze = $horai;
$iH = 0;
$valoresSensor = array();
// calculo da data/hora final de pesquisa
$horafquinze = date("Y-m-d H:i:s.001",mktime(23,59,59,$me,$de,$Ye));
// Busca os dados para cada ms do período de pesquisa selecionado
foreach ($monthList as $mY){
    $m = $mY['m'];
    $Y = $mY['Y'];
    if(strlen($m)<2)$m="0".$m; // Normaliza comprimento do ms Exemplo: 2 passa a 02
    // Query SQL de consulta dos dados na base de dados
    $sql = "SELECT *
        FROM ".$nomeTabelaDB."
        WHERE date_time BETWEEN  '".$horaiquinze."' AND '".$horafquinze."' ORDER BY date
_time ASC";
    $aQueryResult = $dbObject->sql_query($sql);    // Execução da query
    $valor = 0;
    while ($row = @mysql_fetch_array($aQueryResult, MYSQL_ASSOC)){// Se recebeu dados
        list($dateq,$timeq) = explode(" ",$row['date_time']);// regista data da entrada
        list($y,$m,$d) = explode("-",$dateq);         // separa os campos da data e hora
        list($H,$min,$s) = explode(":",$timeq);
        list($s,$ms) = explode(".",$s);
        $timeStamp = mktime($H,$min,$s,$m,$d,$y);    // produz um nœmero inteiro em ms
       $datax[] = $timeStamp;
        // Conforme o sensor escolhido no formulario aplica a função de conversão
        switch ($sensor){
            case 0:
                if($row['humidity'] > 0){
                    $humidade = round((-39.6+0.01*$row['temperature']-25.0) *
(0.01+0.00008*$row['humidity']) -4.0+0.0405*$row['humidity']-0.0000028 *
$row['humidity']*$row['humidity'],2);
                    $valor = $humidade;
                }else{
                    $valor = 0;
                }

                break;
            case 1:
                if($row['temperature'] > 0){
                    $temp = round((-39.6+0.01*$row['temperature']),2);
                    $valor = $temp;
                }else{
                    $valor = 0;
                }
                break;
            case 2:
                if($row['light'] > 0){
                    $valor = ($row['light']*2.56*0.625*10000/1024);

                }else{
                    $valor = 0;
                }
                if(empty($valoresSensor)){
                    $valor = 50;
                }
                break;
            case 3:
                if($row['voltage'] > 0){
                    $voltage = $row['voltage']*2.56*1.990/1024;
                    $valor = $voltage;
                }else{
```

```
                    $valor = 0;
                }
                break;
            case 4:
                $valor = (($row['o2']*1000*2.56*1.58)/1024/40);
                if(empty($valoresSensor)){
                    $valor = 15;
                }

                break;

            case 5:
                $valor = ((($row['co']*2.56/1024)*10000/101)/1.374);
                if(empty($valoresSensor)){
                    $valor = 7.55;
                }
                break;

            case 6:
                $valor = $row['co2'];
                if(empty($valoresSensor)){
                    $valor = 300;
                }
                break;
            default:
                break;
        }
        $valoresSensor[] = $valor;    // Regista os dados dos v‡rios meses
    }
}

// Carrega livrarias de desneho de gráficos
include ("src/jpgraph.php");
include ("src/jpgraph_scatter.php");
// Define os par‰metros dos gr‡ficos
$graph = new Graph(400,200,date("YmdHis"));
$graph->img->SetMargin(40,40,40,40);
$graph->img->SetAntiAliasing();
$graph->SetScale("linlin");
$graph->xaxis-
>SetXStartEndValues(mktime(0,0,1,$mesa,$diai,$anoa),mktime(23,59,59,$me,$de,$Ye));
$graph->title->Set($campoTabNome[$sensor]);    // Carrega a legenda dos eixos
$graph->title->SetFont(FF_FONT1,FS_BOLD);
if(!empty($valoresSensor)){    // Se a lista não estiver vazia carrga os valores
    $sp1 = new ScatterPlot($valoresSensor,$datax);
    $sp1->SetLinkPoints(true,"blue",1);
    $sp1->mark->SetType(MARK_CIRCLE);
    $sp1->mark->SetFillColor("blue");
    $sp1->mark->SetWidth(0);
    $graph->Add($sp1);
}
$graph->Stroke();        // Desenha o gráfico
$dbObject->sql_close();    // Fecha ligação à base de dados
?>
```

ANEXO J – Artigo apresentado na conferência AMIES 2009

Application of ZigBee and Bluetooth to Urban Ambient Monitoring and Guidance

Presented at:

8th International Conference and Workshop
Ambient Intelligence and Embedded Systems
23 - 25 September, 2009
Madeira, Portugal

Application of ZigBee and Bluetooth to Urban Ambient Monitoring and Guidance

Filipe E. S. Santos
Mathematics and Engineering Department
University of Madeira
Funchal, Portugal
fsantos@uma.pt

Joaquim A. R. Azevedo
Mathematics and Engineering Department
University of Madeira
Funchal, Portugal
jara@uma.pt

Abstract - **This work focuses on development of a prototype for a Wireless Sensor Network (WSN) that monitors various environmental parameters of interest in urban areas based on ZigBee protocol. This is performed through a small device that can be placed anywhere in a city. First, it is studied the operation of ZigBee protocol. Second, it was chosen and tested a ZigBee module and sensors from the market. Then, it was developed a module that monitors: humidity, temperature, light, carbon monoxide, carbon dioxide and oxygen. These data are measured and sent periodically to a base station connected to a computer. These data are stored and processed for presentation on the Internet. To demonstrate the capabilities of the network, each node is equipped with Bluetooth, so that passersby where the scope of the network with their actual cell-phone can get information of the environmental quality on that area like joggers, and also guide passerby like tourists.**

Keywords-component; ZigBee; Bluetooth; Sensors; Localization; Low power

I. INTRODUCTION

Given the growing interest on the population life quality, it's important to monitor environmental parameters, especially in urban areas. Currently, monitoring is done through large and expensive devices, which are not always connected to their control center and are in a small number for the coverage area. The connection of a ambient monitoring to a wireless network creates new possibilities [1]. Projects for wireless air monitoring have been developed in the last years [2], specially based on ZigBee technology due to its low power consumption and low cost [3]. This work purpose is to develop a complete prototype of a small node for WSN with high integration of sensors with low power consumption and low cost, and to give a step further on networks integration adding Bluetooth interface. This integration can contribute for a better citizen's life. For example, was proved that polluted areas with gases like CO decrease the athlete's performance [4]. With this interface joggers can check on their phone the present air conditions, changing their route for less polluted areas. It is an opportunity to use the ZigBee network to transport small amounts of information to passerby. For example information of some event near the monitoring place.

II. WIRELESS AMBIENT MONITORING SYSTEM

In this section we will present the components that, with their characteristics, made the possibility to achieve our purposes of relatively small size and low power module for monitoring environmental parameters.

A. Sensors

Sensors where chosen from market, giving especially attention to the small size, low power and reasonably priced.

The CO2 sensor is the C20 solid-state sensor produced by Gas Sensing Solutions (GSS), which detects the concentration of CO2 by measuring absorption of infrared light [5]. This technology delivers high speed (startup of 2 s), sunlight immunity, accuracy and especially low power consumption (<100mW). The C20 is a sensor provided in a complete module, fully factory calibrated, which already includes the processing of measured data (conversion and linearization), providing the measured value via UART.

The CO sensor used is the TGS5042 from Figaro. It is a battery operable electrochemical sensor that offers a current output that varies linearly with the CO concentration in air [6]. Manufacturer indicates the calibration data. To monitor CO, it is necessary to convert sensor current output to voltage, and then can be read by the microcontroller ADC. Sensor offset correction and temperature compensation table is carried out by the microcontroller.

The Oxygen sensor is KE-25, a unique galvanic cell type oxygen sensor produced by Figaro [7]. The sensor does not require external power supply, but due to small scale of voltage it is used an OP-Amp to amplify the sensor output.

For temperature and humidity it is used the SHT15, which is a dual sensor in a single chip produced by Sensirion [8]. It is largely used on WSN due to its low power consumption and tiny size. It comes calibrated from factory, and gives a digital output. The interface is similar to I2C but not standard. So is needed to embed the protocol on the microcontroller program.

For light sensing it is used two photodiodes from Hamamatsu, S1087 and S1087-01 to measure Photosynthetic Active Radiation and Total Solar Radiation respectively [9].

Again due to small current output an OP-Amp circuit is used to convert the signal to a proportional voltage.

B. *ZigBee*

ZigBee technology, is vastly used on similar applications, since it has the characteristics suitable to route sensor readings periodically to a center base station wirelessly and still with low power consumption.

The variable dynamic topology (peer-to-peer, star, cluster-tree or mesh) [10] simplifies the placement of the nodes for ambient monitoring on an urban area.

For the ZigBee module, the XBEE from DIGI is used since it works within the ZigBee protocol, it is low cost, low power and especially easy of use. The XBEE offers a simple UART interface to the application, being one of the main merits that the developer need not to be an expert in ZigBee technology and can therefore only work on the application.

The XBEE module is commercialized in two versions: XBEE OEM and XBEE PRO, differentiated specially by the sensitivity level and output level. They offer an indoor/urban RF range of up to 40m and 100m respectively. On-line of sight they can reach the 120m and 1,6km respectively [11].

To establish a ZigBee network, it is necessary to program the XBEE's with the appropriate firmware and identify two types of ZigBee devices: the network controller – Coordinator; and the Routers/End Devices, since the difference between Routers and End Devices is to be always active for Routers and sleeping whenever not transmitting for End Devices. The hardware interface is completed with the connection of one digital I/O pin from the microcontroller to the XBEE sleep pin.

Since the nodes will be stationary on urban buildings it is proposed the cluster-tree topology as the best choice, since it is possible to define who will be the End Device and who will act as a router, requesting different power levels supply. The topology is showed in the system architecture on Fig. 1.

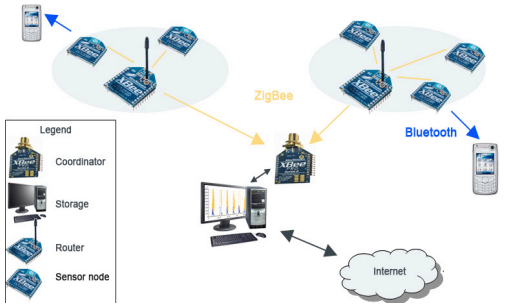


Figure 1. Wireless Ambient Monitoring System architecture

C. *Wireless sensor node prototype*

The wireless sensor board was developed using a small 8bit microcontroller, ATmega324 that is low power and low cost [12]. For the design was considered the power source, so was added a power circuitry consisting on DC-DC converters to simplify the limitations on power source voltage. The microcontroller controls the shutdown on the power circuitry for the CO_2 sensor since it does not have that feature. The node architecture is shown on Fig. 2. To complete the module, a small display, button and LED were added to provide direct access to module data. The assembled circuit is shown in Fig. 3.

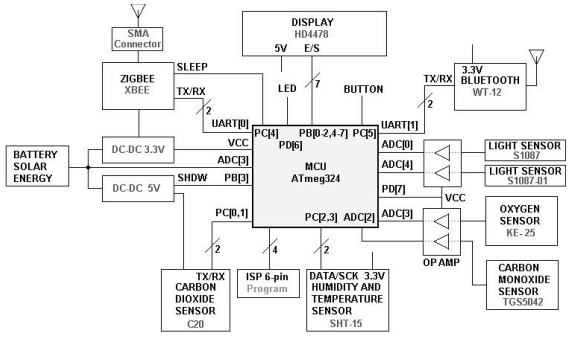


Figure 2. Wireless sensor node architecture

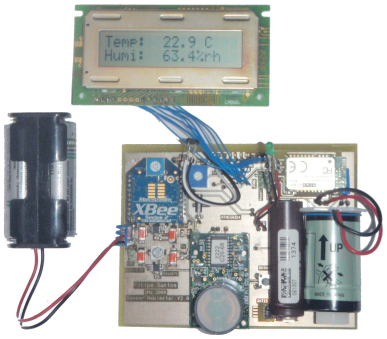


Figure 3. Wireless sensor prototyepe

The wireless sensor node is programmed to acquire sensor data, wrap it in to a ZigBee message addressed to the coordinator and send it through the XBEE every minute. It also has registered the GPS position and street name for Bluetooth purpose.

D. *Bluetooth interface*

The wireless sensor node is equipped with Bluetooth module GIGA WT-11, that is a class 1 device, offering an RF range of up to 300 meters [13]. It exhibits small size, transparent functionality and low power consumption. The node is programmed to answer at Bluetooth requests, with node location (GPS coordinates), Street name and last measured sensor data.

A small Java ME application was developed for cell-phone since it is one of the most present languages on cell phones [14]. The application starts showing a preloaded map of the area, and begins to search for wireless sensor nodes. The first node found is inquired and, after data is received, the map is marked and centered on the node position. The sensors values and the street name are also presented. After 10 seconds starts searching and inquire again. Cell-phone interfaces are shown in Fig. 4.

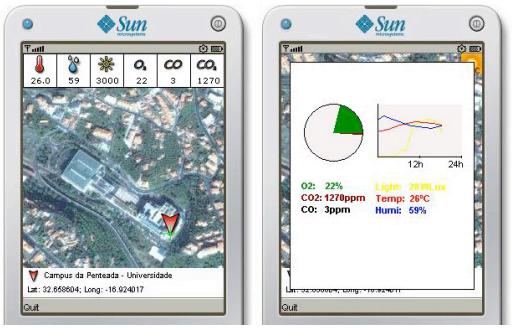


Figure 4. Cell-phone bluetooth application

A second module was built to test the Bluetooth application transition. The two wireless sensor modules were placed on a building wall with a distance of 100 meters and tests were made walking between them with the cell-phone application running. The results showed that at normal walking speed the application changed from one node information to the other after some seconds (< 30s). But at higher speed's (like running) some times only one node was discovered.

E. Web Interface

The XBEE coordinator is connected to a central computer and receives each message sent from the End Devices. The computer has a small Java application always running to receive and store every message to a MySQL database. A web page was developed to show the ambient monitored parameters through Internet in real-time, using graphics for better perception, as it is shown in Fig. 5.

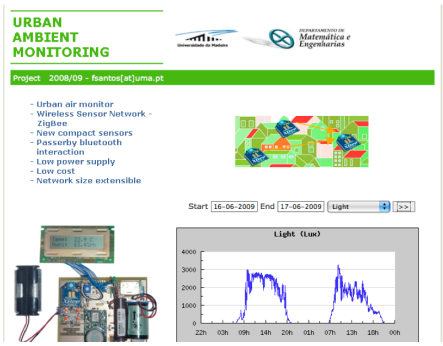


Figure 5. System web page interface

III. RESULTS

The module was placed near a Window of the Laboratory of the university during the full month of July 2009.

Every sensor responds in general in correspondence to their description. The light sensor responds in accordance to light variance, as shown on Fig. 6. The light registers the luminance near a windows, and in a particular hour of the morning receives direct sunlight.

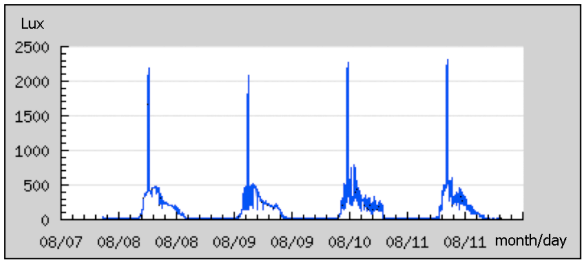


Figure 6. Iluminance response for 4 days

Temperature and Humidity sensor exhibits good quality data, as shown in Fig. 7.

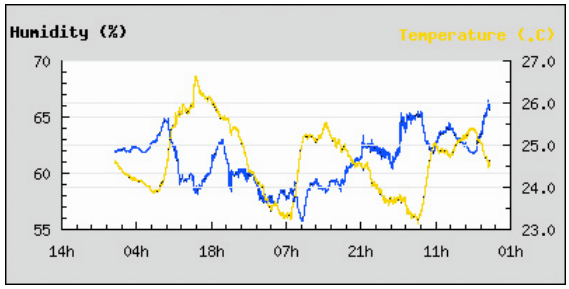


Figure 7. Temperature and humidity response for 3 days

O_2 sensor exhibits a stable line around 21 %. CO sensor only registers values above 10ppm, but responds quickly with the proximity of a lighter, as shown on Fig. 8.

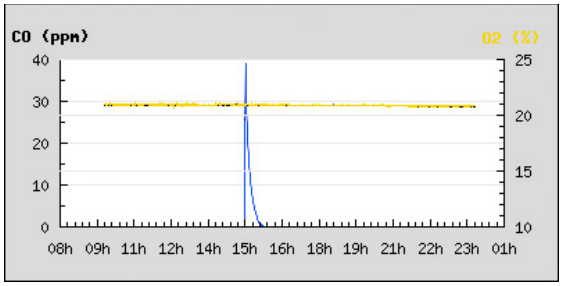


Figure 8. O_2 and CO sensor response for 1 day

For the CO_2, a low pass filter of 8 seconds was used but still on low level of dioxide carbon it responds noisy, as shown on Fig. 8. It is to be decided if the filter bandwidth should be increased, since it will request more ON time, thus contributing to higher energy consumption. The Fig. 9 show the CO2 concentration for two consecutive days near a road. It show a repetitive increase during the daylight time.

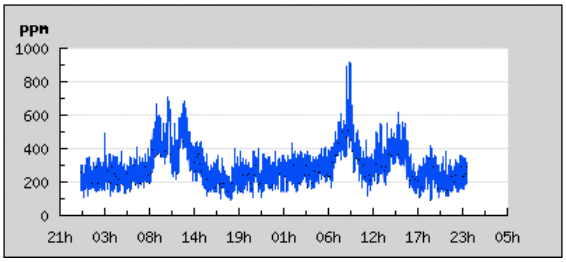


Figure 9. CO_2 sensor response for 2 days

One of the main goals was to obtain low power consumption. This was limited due to the CO_2 sensor power requirements and to continuous Bluetooth operation. Although with intelligent power management, putting to sleep every component when not needed, was possible to increase the duration of a pair of batteries (2xAA 2500mAh) from 22 hours when all components active to 7 full days, as showed in Fig. 9

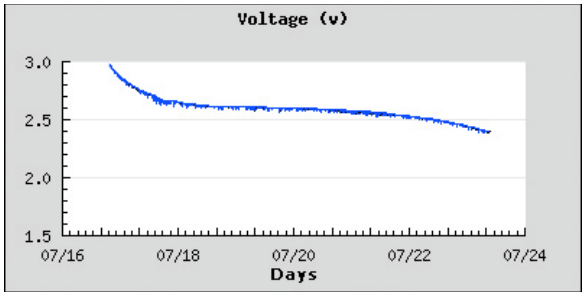


Figure 10. Battery voltage discharge

The size of the prototype is relative small, and can even be reduced. Having in mind the operational temperature and humidity range of every component, this prototype exhibits the specifications indicated on Tab. 1.

TABLE I. WIRELESS MODULE SPECIFICATIONS

Parameter	**Values**	
	Values	***Observation***
Current consumption	Avg. < 15mA Max. < 180mA	@ 2.6 V
Humidity Range	10% to 90%	Limited by O2 sensor
Temperature Range	5ºC to 40ºC	Limited by O2 sensor
Size	144 x 89 x 45mm	Including protection box
Input voltage	1.3 to 3.3V	

IV. CONCLUSION

This work describes the implementation of a prototype for an urban ambient monitoring system. This work demonstrated that ZigBee technology combined with the current sensory technology allows the creation of modules with a high level of integration of features, while being of low power consumption and small size.

It is demonstrated that using Bluetooth gives people instant access to important data. Therefore placing some of these modules in the city creates diverse possibilities of information to present to people. This work is a middle step to ubiquitous networks.

The project is still in development. Future work will be: improve box impermeability; build more modules to form an complete network and place them in urban area for test the network and sensors in the long term; add solar panels to some of the nodes to become energy independent; and add system functionality to alter Bluetooth information, directly from central computer.

ACKNOWLEDGMENT

This work was supported by the CITMA scholarship.

REFERENCES

[1] ZigBee Aliance. ZigBee Specification, 2008. <http://www.zigbee.org/> November, 2008.

[2] Jong-Won Kwon, Yong-Man Park, Sang-Jun Koo, Hiesik Kim, "Design of Air Pollution Monitoring System Using ZigBee Networks for Ubiquitous-City," iccit, pp.1024-1031, 2007 International Conference on Convergence Information Technology (ICCIT 2007), 2007.

[3] P. Kinney, “ZigBee Technology: Wireless Control That Simply Works,” Communications Design Conference, October 2003.

[4] A. J. Carlisle; e N. C. C. Sharp, "Exercise and outdoor ambient air pollution". British Journal of Sports Medicine, 01 Agosto de 2001.

[5] C20, User Instructions, GSS 2008.

[6] TGS5042, Product Informaiton, Figaro Engineering Inc.

[7] KE-25/KE-50, Product Information, Figaro Engineering Inc.

[8] SHT1x, Datasheet, Sensirion, Jully 2008.

[9] S1087, Datasheet, Si Photodiode.

[10] Abd-El-Barr, M.I.; Youssef, M.A.M.; Al-Otaibi, M.M. “WIRELESS SENSOR NETWORKS - PART I: TOPOLOGY AND DESIGN ISSUES,” in: Electrical and Computer Engineering, 2005, Canadian Conference.

[11] XBee/XBee-PRO, Datasheet, Digi International, 2008.

[12] ATmega324P, Datasheet, ATMEL.

[13] WT-11, Datasheet v2.8, BlueGiga Technologies, 28 August, 2008.

[14] Sam Mason, Elise Korolev, “Native and Java ME Development on Symbian OS,” Published by the Symbian Developer Network Version: 1.0 – March 2008.