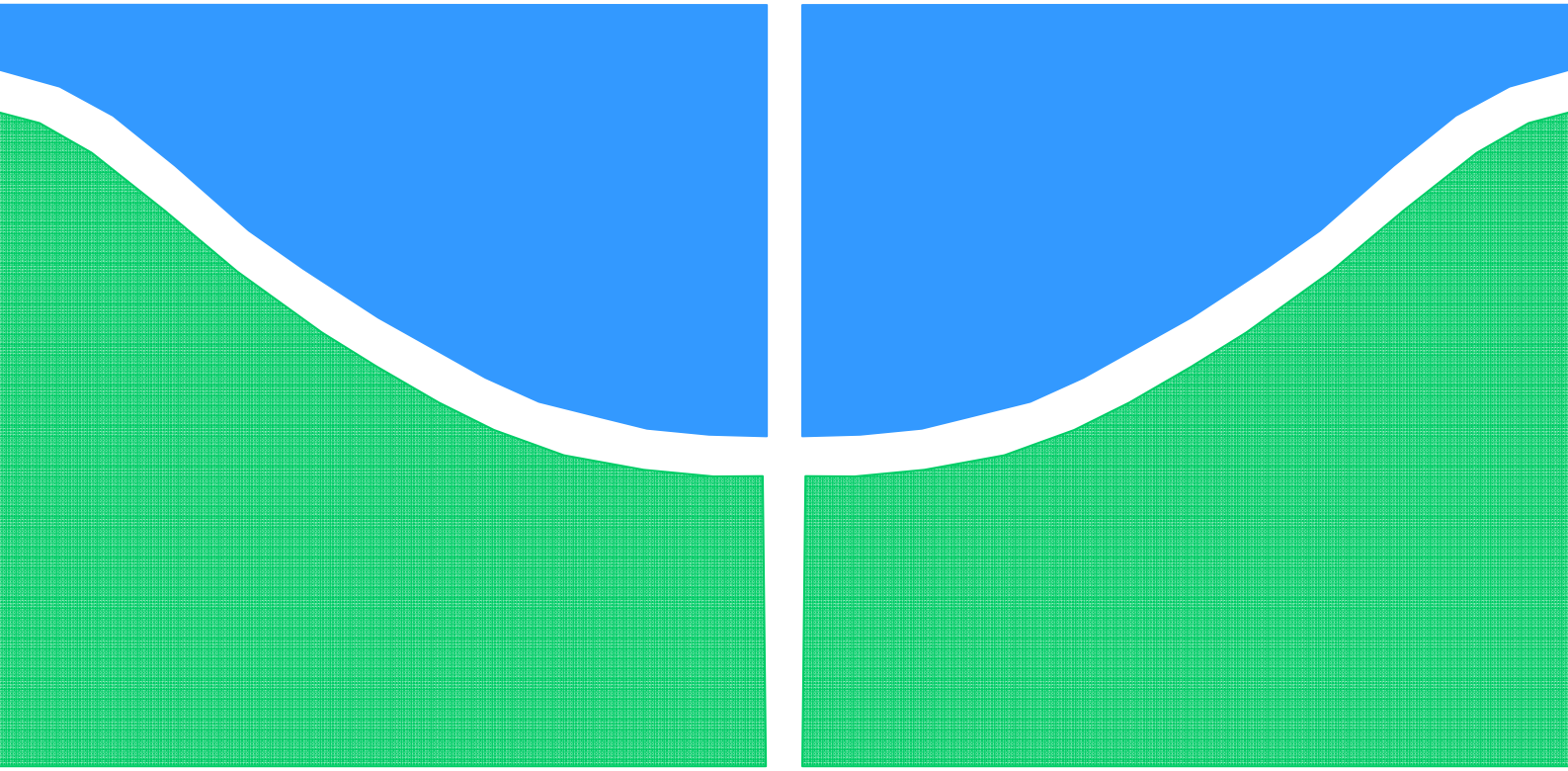

TRABALHO DE GRADUAÇÃO

REDE DE SENSORES SEM FIO PARA AUTOMAÇÃO PREDIAL COM MÓDULOS MESHBEAN

Rodrigo Bertuol de Queiroz
Raphael Carvalho de Almeida Azevedo

Brasília, julho de 2009

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Rede de sensores sem fio para automação predial com módulos MeshBean.

Rodrigo Bertuol de Queiroz
Raphael Carvalho de Almeida Azevedo

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

Banca Examinadora

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

Prof. Geovany Araújo Borges, UnB/ ENE _____

Prof. Ricardo Zelenovsky, UnB/ ENE _____

Dedicatória(s)

A Deus, a meus pais Adriana e Ernandes, a meus irmãos Arthur e Frederico, minha namorada Mayana, amigos e familiares.

Raphael Carvalho de Almeida Azevedo

Dedico este trabalho a minha família, em especial, a minha mãe Sandra e meu Pai Pachel, aos meus amigos e a minha namorada Carolina.

Rodrigo Bertuol de Queiroz

Agradecimentos

A minha mãe Sandra pela sua força, doação e carinho.
Ao meu pai Pachel por me ensinar os valores da honestidade, do caráter e da simplicidade.
Aos meus irmãos pelo companheirismo e amizade incondicional.
A Minha namorada Carolina por me apoiar, me ajudar e acreditar em mim.
Ao meu amigo e companheiro Raphael pela amizade e apoio nos momentos em que precisei.
Ao professor Adolfo pela oportunidade que me concedeu e pelo conhecimento transmitido.
Aos amigos do LARA e do LAVSI que estiveram sempre dispostos a contribuir com este trabalho.

Rodrigo Bertuol de Queiroz

Primeiramente a Deus pela oportunidade a mim concedida.
A meus pais, Adriana e Ernandes por terem dedicado toda a vida em função dos filhos, na busca incessante de fornecer a melhor educação possível. Muito obrigado pelo exemplo de vida que me deram.
A meus irmãos, Arthur e Frederico pelo companheirismo e momentos de felicidade que me proporcionam.
A meus avós, Eny e Antônio por todo o apoio fornecido nesta minha caminhada.
A meus avós, Izabel e Patrocínio, cujas lembranças me trazem sorrisos. Sei que acompanham meus passos de onde estão.
A minha namorada Mayana, por toda a paciência e auxílio durante a universidade.
A meu amigo Rodrigo, por todos os trabalhos e estudos que realizamos juntos.
Aos meus amigos do Senado Federal pelo companheirismo e apoio técnico que em muito contribuiu para o presente projeto.
A meus amigos Thiago e Guilherme, a todo o pessoal do LAVSI e LARA, em especial a Helger, Paulo, Felipe e Luís pelo auxílio nos trabalhos. E, por fim, a todos aqueles que, direta ou indiretamente, me auxiliaram na graduação.

OBRIGADO.

Raphael Carvalho de Almeida Azevedo

RESUMO

Com a utilização de módulos MeshBean implementou-se uma rede de sensores sem fio para automação de equipamentos de ar condicionado. Utilizou-se o gerenciador de tarefas BitCloud fornecido pelo fabricante para o desenvolvimento da aplicação embarcada da rede. Com a rede constituída, a partir da Norma técnica ISSO 7730 definiu-se a temperatura de conforto para o laboratório e avaliou-se o desempenho do sistema no aspecto de economia de energia.

ABSTRACT

It was made an implementation of a wireless sensor network for automation of air conditioner equipments using MeshBean modules. The task manager BitCloud provided by the manufacturer was used to develop the embedded software loaded on the network. Once the network was established, the comfort temperature of the laboratory was set up according to the Technical Standard ISO 7730, and it was evaluated the performance of the system according to energy saving.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

G	Giga	
M	Mega	
K	*Kilo*	
m	mili	
h	coeficiente de convecção	[(m/s)1/2]
h	hora	
s	segundo	

Símbolos Gregos

μ	micro	10^{-6}

Subscritos

vest	vestuário
pele	pele
vap	vapor
sat	saturação
ar	ar
rad	radiante

Siglas

ASHARE	American Society of Heating, Refrigerating and Air-Conditioning Engineers
°C	Graus Celsius
AD	Analog to Digital
API	Application Programming Interface
BTU	British Thermal Unit
CAV	Convenção Americana de Ventilação
CFTV	Circuito Fechado de TV
dB	Decibel
dBm	Mili Decibel
EEPROM	Electrically Erasable Programmable Read-Only Memory
GHz	Giga Hertz
HVAC	Heating Ventilating and Air Conditioning
I2C	Inter-Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
JTAG	Join Test Action Group
kB	Kilo Bytes
Kbps	Kilo bits por segundo
LARA	Laboratório de Robótica e Automação
LAVSI	Laboratório de Visão e Sistemas Inteligentes
mA	Mili Amperes

MCU	Microcontroller Unit
MHz	Mega Hertz
mim	Minuto
MS-DOS	Microsoft Disk Operation System
PER	Packet Erro Rate
PMV	Predicted Mean Vote
PPD	Predicted Percentage of Dissatisfied
RF	Radio Freqüência
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
V	Volts
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Network
Zigbee	Padrão IEEE 802.15.4 de comunicação sem fio

1 INTRODUÇÃO

Apresenta uma visão geral sobre as redes de sensores sem fio.

1.1 ASPECTOS GERAIS

As redes de sensores sem fio apresentam ampla aplicação, entre elas estão as aplicações militares, meteorológica, monitoramento de veículos, agricultura de precisão, automação predial, controle de patrimônio, entre outras [1]. Em automação predial, as redes de sensores sem fio são uma excelente ferramenta especialmente no *retroffiting* de instalações. Isso se deve ao fato de que a tecnologia sem fio permite rápida e fácil instalação dispensando a implantação de uma infra-estrutura dispendiosa. Outro aspecto é que em muitos edifícios não há espaço para esta infra-estrutura.

A rede de sensores facilita a implantação de ambientes inteligentes com alto nível de conforto das instalações e com economia dos recursos energéticos. A economia de energia ocorre devido à gestão racional e interligada dos sistemas monitorados, como o de iluminação, aquecimento e refrigeração e quadros elétricos. Outro aspecto é o da segurança das instalações, por exemplo, a interligação do sistema de prevenção e combate a incêndio com o sistema de ar condicionado. Assim, quando os sensores de fumaça detectam um incêndio, os dutos de ventilação são estancados evitando a proliferação das chamas.

O estabelecimento de parâmetros de conforto térmico tem grande importância na automação de sistemas de ar condicionado, pois é necessário determinar a temperatura e a ventilação de conforto para controlar o sistema. Então, conhecendo as condições de conforto térmico adequadas para cada situação é possível proporcionar um ambiente agradável para seus ocupantes.

1.2 OBJETIVO

Instalar uma rede de sensores sem fio com o módulo Zigbit para o controle de equipamentos de ar condicionado no ambiente do laboratório LAVSI.

1.3 TRABALHOS ANTERIORES.

O primeiro trabalho a projetar uma rede de sensores sem fio para o laboratório LAVSI implementou um sistema como o ilustrado na Figura 1 [2]. A principal diferença dessa versão para a proposta neste trabalho está nos módulos utilizados. A versão anterior utilizou o *transceiver* Xbee, que é um dispositivo configurável e não permite aplicações embarcadas. O sistema proposto utiliza o ZigBit que possui um sistema cooperativo multitarefa permitindo o carregamento de aplicações.

Todo o sistema anterior é gerenciado por comandos AT (originado da palavra *attention*) enviados por um software hospedado em um computador. Os comandos AT são enviados para o módulo coordenador que, por sua vez, envia comandos para os nós sensores e atuadores. Por meio desses comandos o coordenador pode, por exemplo, solicitar dados que os nós sensores estão configurados para enviar.

O nó atuador utiliza um microcontrolador que é responsável pela atuação sobre o sistema de ar condicionado. No nó atuador o Xbee funciona apenas como um canal de comunicação entre o computador e o microcontrolador, portanto, parte de suas funcionalidades ficam inutilizadas.

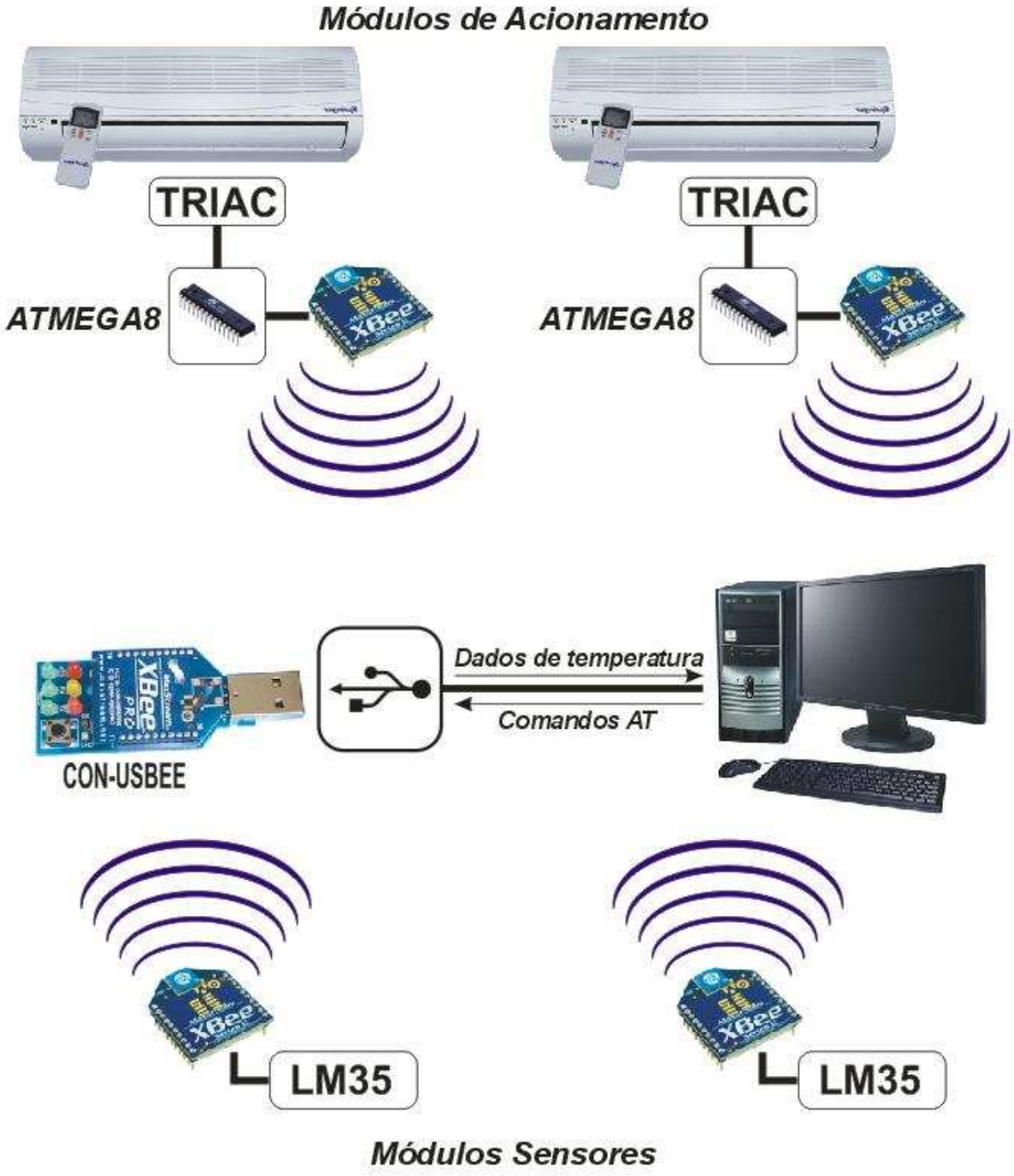

Figura 1- rede de sensores sem fio do projeto anterior.

O sistema da Figura 1 apresenta essa configuração devido às características do Xbee que pode funcionar como um canal transparente de comunicação ou como dispositivo configurável. Entretanto, esse dispositivo não é programável. Sua configuração é feita por meio da placa CON-USBEE. Operacionalmente, o Xbee configurado como dispositivo final responde a solicitações feitas pelo módulo coordenador. Ou seja, o dispositivo tem que receber um comando para comunicar-se com o coordenador. Até mesmo o coordenador tem que receber um comando AT do software supervisório para comunicar-se com os outros dispositivos da rede. Por exemplo: para mudar o dispositivo com o qual o coordenador está se comunicando, o supervisório manda um comando para o coordenador alterando o endereço do dispositivo final.

2 COMFORTO TÉRMICO

Uma visão a respeito do conceito de conforto térmico, das normas técnicas utilizadas atualmente e da norma ISO 7730 que é utilizada neste projeto.

2.1 CONFORTO TÉRMICO

A norma técnica ASHRAE 55 define conforto térmico como: "a condição da mente que expressa satisfação com o ambiente térmico". Do ponto de vista físico, uma situação de conforto é aquela que permite a manutenção da temperatura interna do corpo humano sem a necessidade de acionamento dos mecanismos reguladores, ou seja, um ambiente em que há equilíbrio entre o calor produzido pelo organismo e o calor perdido para o ambiente.

A sensação de conforto esta relacionada a variáveis pessoais e ambientais, portanto pode ser considerada uma sensação subjetiva. Duas pessoas submetidas às mesmas condições físicas (temperatura, umidade do ar, velocidade do vento) podem ter impressões diferentes em relação ao estado térmico do ambiente. Devido a essa subjetividade, avalia-se o estado que agrada o maior número de pessoas possível para quantizar uma situação de conforto térmico.

Os primeiros estudos a respeito de uma situação de conforto térmico foram realizados em 1916 pela Comissão Americana de Ventilação (CAV), esta análise teve como objetivo encontrar uma relação entre a produtividade dos operários e as condições térmicas às quais estavam expostos.

Após os resultados obtidos pela CAV, diversos estudos foram realizados em ambientes com condições térmicas adversas, chegando a diversas equações e índices que podem caracterizar uma situação de conforto térmico. Surgiu então a necessidade de padronização destes índices com a criação de normas técnicas.

A norma ISO 7730 baseia-se nos estudos do professor dinamarquês Ole Fanger com câmaras climáticas, e é amplamente utilizada em projetos de diversas áreas com o objetivo de proporcionar uma situação confortável aos ambientes internos. As normas técnicas mais utilizadas atualmente serão analisadas, porém será dada maior ênfase à norma ISO 7730.

2.2 NORMAS TÉCNICAS

As normas técnicas de conforto térmico são fundamentais na determinação da temperatura ideal para manter o conforto térmico do ambiente. A seguir são apresentadas as principais normas relacionadas com conforto térmico.

2.2.1 ASHRAE 55 (Condições térmicas de um ambiente para ocupação humana)

O objetivo desta norma é: "*determinar uma combinação de fatores pessoais e condições térmicas de ambientes internos de uma maneira que se torne aceitável a 80% dos ocupantes"* [3].

Esta norma utiliza temperatura, umidade do ar, radiação térmica e velocidade do ar como parâmetros físicos do ambiente, e para os parâmetros pessoais, o tipo de vestimenta que as pessoas estão utilizando e o nível de atividade exercida no ambiente.

2.2.2 ASHRAE 62 (Ventilação e qualidade do ar aceitável em ambientes internos)

O objetivo desta norma é: "determinar taxas de ventilação e níveis de qualidade do ar aceitáveis para seres humanos em ambientes internos que evitem danos a saúde" [3].

Esta norma também se aplica a qualquer ambiente fechado, desde que não exista norma específica para o ambiente ou que não entre em conflito com a AHSRAE 55.

A ASHRAE 62 normatiza o controle da quantidade de partículas dispersas no ar determinando taxas de ventilação aceitáveis ou a instalação de filtros de ar e coletores de poeira no sistema HVAC.

2.2.3 ASHRAE 113 (Métodos de ensaios para difusão de ar em ambientes fechados)

O objetivo desta norma é: "*definir métodos recursivos de testes para definir o desempenho da difusão do ar em sistemas de escoamento de ar em ambientes fechados"* [3].

Esta norma assume que os ocupantes dos ambientes utilizam vestimentas típicas para ambientes fechados e executam atividades leves ou sedentárias e que os níveis de temperatura média radiante e umidade do ar estejam dentro dos limites especificados para uma situação de conforto de acordo com a norma ASHRAE 55 ou ISO 7730.

A ASHRAE 113 auxilia o desenvolvimento de um layout para os sistemas HVAC que proporciona um ambiente agradável termicamente com base no deslocamento de ar, na velocidade do vento e na temperatura do ar devidos às zonas de aquecimento e resfriamento. A ASHRAE 113 não se aplica, portanto, a ambientes com apenas uma saída para o fluxo do ar [3].

2.2.4 CR 1752 (Ventilação em construções – Critérios de designer para ambientes fechados)

Esta norma fornece assistência para prover níveis aceitáveis de fluxo do ar em ambientes fechados com ventilação forçada por aparelhos de ventilação ou de condicionamento do ar [3].

Para esta assistência, o relatório técnico CR 1752 se baseia em algumas normas ISO, que especificam uma situação de conforto térmico e acústico devido à emissão de ruídos sonoros pelos aparelhos de ventilação e ar condicionado no ambiente.

2.2.5 ISO 7726 (Ergonomia em um ambiente térmico – Instrumentos para a medição de grandezas físicas)

O objetivo desta norma é: *"especificar os requisitos mínimos assim como os métodos de medidas para a aquisição dos dados e grandezas físicas que caracterizam um ambiente termicamente"* [3].

Esta norma é citada em diversas outras devido à necessidade de obtenção das grandezas físicas do ambiente (temperatura, umidade do ar, velocidade do vento, entre outras) para a determinação ou avaliação do nível de conforto.

2.2.6 ISO 7933 (Ambientes Quentes – Determinação analítica e interpretação do stress térmico a partir do cálculo da taxa de suor requerida)

Uma situação de *stress* provoca uma perturbação psicológica ou biológica devido a parâmetros pessoais (doença, emoção, condições de vida, entre outros) ou parâmetros externos (frio, calor, entre outros). O *stress* térmico ocorre quando uma pessoa está exposta a situações extremas de frio ou calor. Quando exposto a estas condições, o ser humano pode apresentar debilitação física e ter alterações das reações psicosensoriais [4].

A ISO 7933 especifica métodos analíticos para determinar e avaliar o nível de stress térmico a que um sujeito está submetido em ambientes com altas temperaturas, a partir de cálculos do balanço da temperatura corpórea e da taxa de suor do indivíduo.

Nesta norma, as taxas de suor são determinadas em função das condições térmicas em que as pessoas estão submetidas, além de especificar os parâmetros que podem ser modificados com o objetivo de reduzir os riscos à saúde humana [3].

2.2.7 ISO 8996 (Ergonomia – Determinação do calor produzido devido ao metabolismo)

O conhecimento da taxa metabólica de uma pessoa é necessário para uma avaliação do balanço térmico entre o calor produzido pelo corpo, e o calor liberado para o ambiente (situação de conforto térmico).

Esta norma apresenta três métodos diferentes para se estimar a taxa metabólica de um indivíduo, assim como a precisão destas estimações. A ISO 8996 disponibiliza tabelas de taxas metabólicas que foram criadas a partir de estudos de indivíduos executando atividades típicas do cotidiano [3].

2.2.8 ISO 9920 (Estimativa do isolamento térmico e resistência evaporativa de vestimentas)

O balanço térmico entre o indivíduo e o ambiente possui uma relação direta com o tipo de vestimenta utilizada. A sensação térmica percebida pelas pessoas é diretamente influenciada pela resistência evaporativa e pelo isolamento térmico do tecido.

A ISO 9920 especifica métodos de avaliação das características físicas das vestimentas com tecidos uniformes em seus estados normais, ou seja, a norma não leva em consideração a absorção do suor pelo tecido, a utilização de tecidos especiais (tecidos com aquecimento forçado, entre outros) e desconsidera a cobertura irregular do corpo humano com o tecido [3].

2.2.9 ISO 7730 (Ambientes térmicos moderados – Determinação dos índices PMV e PPD e especificações das condições para o conforto térmico)

Esta norma internacional está em vigência desde o início dos anos 80, e baseia-se em um modelo estático de transferência de calor calibrado a partir de um grande número de pessoas em uma câmara climática.

A norma tem como objetivo apresentar um método de previsão da sensação térmica e do grau de desconforto de pessoas expostas em ambientes térmicos moderados, assim como especificar condições de ambiente térmico aceitáveis para o conforto [5].

Esta norma leva em consideração para uma situação de conforto parâmetros individuais e ambientais. A partir destes parâmetros a sensação térmica pode ser estimada pelo cálculo do índice do voto médio (PMV) ou pela porcentagem de pessoas descontentes no ambiente através do índice PPD [4].

Devido à grande aceitação desta norma internacionalmente e às características físicas do ambiente proposto para o controle, esta norma foi escolhida para o presente trabalho, e será, portanto, discutida detalhadamente.

2.2.10 OUTRAS NORMAS

Diversas outras normas foram criadas para atender a situações e ambientes específicos, ou mesmo para auxiliar as normas citadas anteriormente como é o caso da norma ISO 7243 que é amplamente utilizada em ambientes onde a utilização da ISO 7730 não se aplica.

A ISO/TR 11079 (Ambientes frios – índice de isolação) é análoga a ISO 7933 e avalia o stress térmico em que pessoas estão submetidas quando estão sujeitas a baixas temperaturas [3].

A norma 9886 é utilizada para avaliar o esforço térmico corpóreo a partir de medições fisiológicas [3].

2.3 ISO 7730 (Ambientes térmicos moderados – Determinação dos índices PMV e PPD e especificações das condições para o conforto térmico)

Segundo esta norma internacional, um determinado ambiente se encontra confortável termicamente, quando não mais que 10% das pessoas presentes estejam insatisfeitas com as condições térmicas [6].

Após diversos estudos com pessoas em câmaras climatizadas, foi possível encontrar uma relação matemática que descrevesse a quantidade de pessoas descontentes com as condições a que estão

submetidas. Para se chegar a este índice, é necessário seguir uma metodologia de cálculo descrita em detalhes na norma.

Primeiramente é necessário realizar a quantização dos parâmetros físicos e individuais que são levados em consideração pela norma, seguindo a norma ISO 7726. Obtidas todas as variáveis, é possível obter a acumulação energética do corpo utilizando a equação de conforto que é explicada pela ISO 7730 [5].

Com o valor da acumulação energética corpórea é possível determinar os seguintes índices descritos pela norma:

- PMV (*Predicted Mean Vote*): escala quantitativa da sensação de calor/frio do ambiente analisado [6].
- PPD (*Predicted Percentage of Dissatisied*): representa a porcentagem de pessoas insatisfeitas com as situações do ambiente. Este índice é obtido a partir do índice do PMV [6].

2.3.1 PARÂMETROS DO CONFORTO

O corpo humano produz calor a partir das reações químicas realizadas pelo metabolismo corpóreo que está diretamente relacionado com a atividade exercida. O ser humano realiza trocas de calor com o meio ambiente por fenômenos de condução, convecção, radiação e evaporação. A troca de calor entre a superfície da pele e o ambiente pode ser dificultada ou facilitada, dependendo do tipo de vestimenta que a pessoa está utilizando. A perda de calor do corpo por evaporação depende principalmente da umidade relativa do ar, e da velocidade do vento, assim como a convecção depende da temperatura do ar e do fluxo de ar. Portanto os parâmetros mais importantes para o conforto térmico segundo a ISO 7730 podem ser divididos da seguinte forma [6]:

Parâmetros individuais:

- Atividade exercida
- Vestuário utilizado

Parâmetros físicos:

- Temperatura do ar
- Umidade relativa do ar
- Velocidade do ar
- Temperatura média radiante

2.3.2 ANÁLISE DOS PARÂMETROS

2.3.2.1 ATIVIDADE EXERCIDA

O corpo humano produz calor e perde calor para o meio ambiente em que se encontra. Essas perdas de calor podem ocorrer pela respiração ou pela pele. Podem-se dividir as perdas de calor da seguinte maneira:

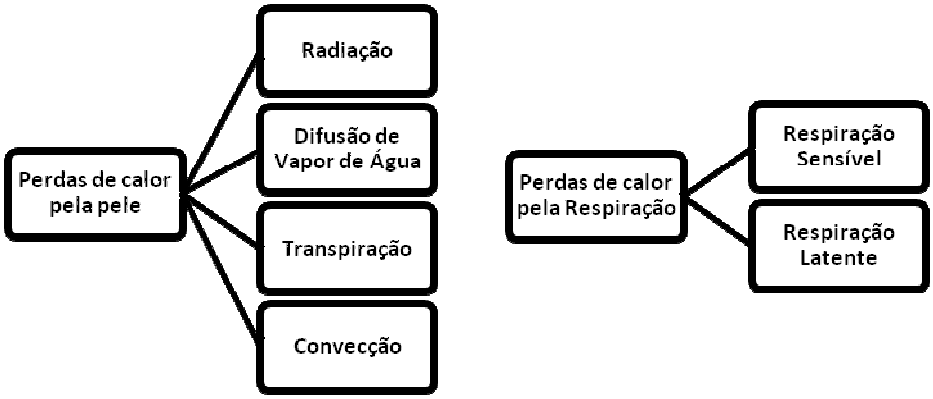

Figura 2- perdas de calor pelo corpo para o ambiente.

Para a o equacionamento do índice PMV, considera-se que a taxa metabólica (M) de um ser humano será igual à dissipação de calor, ou seja, considera-se que não existe armazenamento de calor pelo corpo.

O metabolismo se divide em metabolismo basal (repouso absoluto) e em metabolismo físico, em que a taxa de consumo de energia pelo corpo se encontra diretamente relacionada com o esforço físico executado.

Em relação à atividade física, o índice PMV leva em consideração o trabalho executado pela pessoa, porém para o conforto térmico em ambientes internos, considera-se que o trabalho executado pelas pessoas é nulo, ou seja, pessoas sentadas, ou caminhando.

A taxa metabólica foi normalizada segundo a ISO 7730 tendo como base o metabolismo de uma pessoa sentada ou descansando e possui como unidade *met*, relacionada da seguinte forma [6]:

$$1 \text{ met} = 58.15 \ {}^{W}\!/_{m^2} \tag{1}$$

A norma disponibiliza uma tabela dos valores de metabolismo em met e em W/m^2 de acordo com os seguintes níveis de atividades:

Tabela 1- valores de metabolismo em função do nível de atividade [6].

Atividade Física Executada	Metabolismo (W/pessoa)	Metabolismo (met)	Metabolismo (W/m^2)
Deitado	85	0.8	47
Sentado descansando	104	1.0	58
Atividade sedentária	126	1.2	70
De pé (atividade leve)	167	1.6	93
De pé (atividade média)	210	2.0	117
Grande atividade	315	3.0	175

2.3.2.2 VESTUÁRIO

Como o conforto térmico está diretamente relacionado ao equilíbrio entre o calor produzido pelo corpo e as perdas de calor para o ambiente, o vestuário possui uma influência direta na sensação térmica devido à resistência térmica característica de cada tecido. O vestuário pode facilitar ou dificultar a transferência de calor do corpo para o ambiente.

Para a equação do conforto, a resistência térmica (I_{vest}) possui uma unidade própria *clo* que se relaciona com a resistência térmica da vestimenta da seguinte maneira [6]:

$$1\ clo = 0.155\ {}^{m^2K}\!/_{W} \quad (2)$$

Para o cálculo do índice PMV, as resistências térmicas de diferentes tipos de vestuário foram organizadas em forma de tabela apresentada a seguir.

Tabela 2- valores de resistência térmica em função do vestuário [6].

Vestuário	Restiência Térmica (I_{vest} - clo)	Resistência Térmica (I_{vest} - $\frac{m^2K}{W}$)
Nú	0	0
Calções	0.1	0.016
Vestuário Tropical	0.3	0.047
Vestuário Leve de Verão	0.5	0.078
Vestuário de Trabalho	0.7	0.124
Vestuário de Inverno (interiores)	1.0	0.155
Fato Completo	1.5	0.233

2.3.2.4 UMIDADE DO AR

O vapor de água existente na atmosfera tem a sua quantidade afetada em função da temperatura e da pressão atmosférica. A essa quantidade de vapor de água dá-se o nome de umidade do ar.

Este parâmetro é essencial para o cálculo da pressão de vapor da água, que está relacionado com o cálculo da pressão de saturação da seguinte maneira [6].

$$p_{vap} = HR.p_{sat}(T) \quad (3)$$

Onde:

$p_{vap} \rightarrow$ pressão de vapor

$HR \rightarrow$ umidade relativa do ar

$p_{sat} \rightarrow$ pressão de saturação do ar

$T \rightarrow$ temperatura do ar

Onde a pressão de saturação do ar (em função da temperatura) pode ser calculada da seguinte maneira:

$$p_{sat}(T) = 1000.e^{\left(16.6356 - \frac{4030.183}{T+235}\right)} \quad (4)$$

2.3.2.5 TEMPERATURA MÉDIA RADIANTE

Todos os corpos estão em constante troca de calor com o meio em que se encontram. Essa troca de calor através da radiação é devida a temperatura média dos corpos, que podemos denominar temperatura média radiante.

O valor desta temperatura pode ser maior ou menor do que a do ambiente influenciando diretamente na temperatura e nas trocas de calor dentro do ambiente. Conseqüentemente, a temperatura média radiante influencia o índice do PMV e o conforto térmico sentido pelas pessoas que ocupam o local.

2.3.2.6 VELOCIDADE DO AR

Devido a diferenças de temperatura em um ambiente, deslocamentos de ar ocorrem através de um fenômeno chamado de convecção natural, onde o ar quente se desloca para alturas maiores que o ar frio.

Os ambientes com deslocamentos forçados de ar (com a presença de ventiladores ou aparelhos de ar-condicionado) influenciam a sensação térmica de um indivíduo por facilitar a evaporação das moléculas de água presentes na superfície do corpo, provocando uma idéia de que a temperatura sentida é inferior à real.

2.3.3 CÁLCULO DA EQUAÇÃO DO CONFORTO

A equação do conforto leva em consideração o balanço da transferência de calor em um ambiente com o calor produzido pelo indivíduo e pode ser analisada da seguinte maneira [6]:

$$\begin{aligned}
S = \quad & \text{(Acumulação de calor)} \\
M - W \quad & \text{(Metabolismo e trabalho)} \\
-3,05.10^{-3}.\left(5733 - 6,99.(M-W) - p_{vap}\right) \quad & \text{(Difusão de vapor)} \\
-0,42.\left((M-W) - 58,15\right) \quad & \text{(Transpiração)} \\
-1,7.10^{-5}.M.(5867 - p_{vap}) \quad & \text{(Respiração latente)} \\
-0,0014.M.(34 - T_{ar}) \quad & \text{(Respiração sensível)} \\
-3,96.10^{-8}.f_{vest}.\left(\left(T_{vest}+273\right)^4 - \left(T_{rad}+273\right)^4\right) \quad & \text{(Radiação)} \\
-f_{vest}.h.\left(T_{vest} - T_{ar}\right) \quad & \text{(Convecção)}
\end{aligned} \tag{5}$$

Onde:

$S \rightarrow$ é o termo de acumulação de energia no corpo em W/m^2

$M \rightarrow$ é a taxa de metabolismo em W/m^2

$W \rightarrow$ é o trabalho realizado pelo indivíduo no ambiente

$p_{vap} \rightarrow$ é a pressão parcial do vapor de água no ambiente

$T_{ar} \rightarrow$ é a temperatura seca do ar ambiente

$f_{vest} \rightarrow$ é um fator adimensional relacionado ao vestuário

$T_{vest} \rightarrow$ é a temperatura da parte externa do vestuário

$T_{rad} \rightarrow$ é a temperatura média radiante dos elementos no espaço

$h \rightarrow$ é o coeficiente de convecção entre a superfície do vestuário e o ar exeterior

A variável f_{vest} está relacionada com a resistência térmica do tecido da seguinte maneira:

$$\begin{aligned}
f_{vest} &= 1,00 + 1,29.I_{vest} \qquad \text{para } I_{vest} < 0,078\, m^2K/W \\
f_{vest} &= 1,05 + 0,645.I_{vest} \qquad \text{para } I_{vest} \geq 0,078\, m^2K/W
\end{aligned} \tag{6}$$

A temperatura da parte externa do vestuário pode ser obtida a partir da equação não linear a seguir [6]:

$$T_{vest} = T_{pele} - I_{vest}.\left\{3,96.10^{-8}.f_{vest}.\left(\left(T_{vest}+273\right)^4 - \left(T_{rad}+273\right)^4\right) + f_{vest}.h.\left(T_{vest} - T_{ar}\right)\right\} \tag{7}$$

Onde:

$T_{vest} \rightarrow$ é a temperatura da parte externa do vestuário em °C

$T_{pele} \rightarrow$ é a temperatura da pele em °C

$I_{vest} \rightarrow$ é a resistência térmica do vestuário em m^2K/W

$f_{vest} \rightarrow$ é o fator adimensional relacionado ao vestuário

$T_{rad} \rightarrow$ é a temperatura média radiante dos elementos no espaço

$T_{ar} \rightarrow$ é a temperatura seca do ar ambiente

$h \rightarrow$ é o coeficiente de convecção entre a superfície do vestuário e o ar exterior

Para calcular a temperatura da pele, podemos utilizar a seguinte equação (6):

$$T_{pele} = 35,7 - 0,0275.(M - W) \tag{8}$$

O coeficiente de convecção natural e forçada pode ser determinado a partir das seguintes relações respectivamente [6]:

$$\begin{aligned} h &= 2,38.(T_{vest} - T_{ar})^{0,25} \\ h &= 12,1\sqrt{v} \end{aligned} \tag{9}$$

Onde:

$v \rightarrow$ é a velocidade do ar em $^{m}/_{s}$

2.3.4 CÁLCULO DO PMV

Após o cálculo da equação do conforto, é possível obter o valor do índice PMV a partir do valor da acumulação de energia obtido e do metabolismo realizado pelo corpo humano no ambiente pela seguinte relação [6]:

$$PMV = \left(0,303.e^{-0,036.M} + 0,028\right).S \tag{10}$$

Que possui os seguintes significados:

Figura 3- interpretação do índice PMV.

2.3.5 CÁLCULO DO PPD

Conhecido o valor do índice PMV, é possível estimar a porcentagem de pessoas desconfortáveis termicamente no ambiente em questão, utilizando a equação do índice PPD [6]:

$$PPD = 100 - 95.e^{\left(-0,03353.PMV^4 - 0,2179.PMV^2\right)} \tag{11}$$

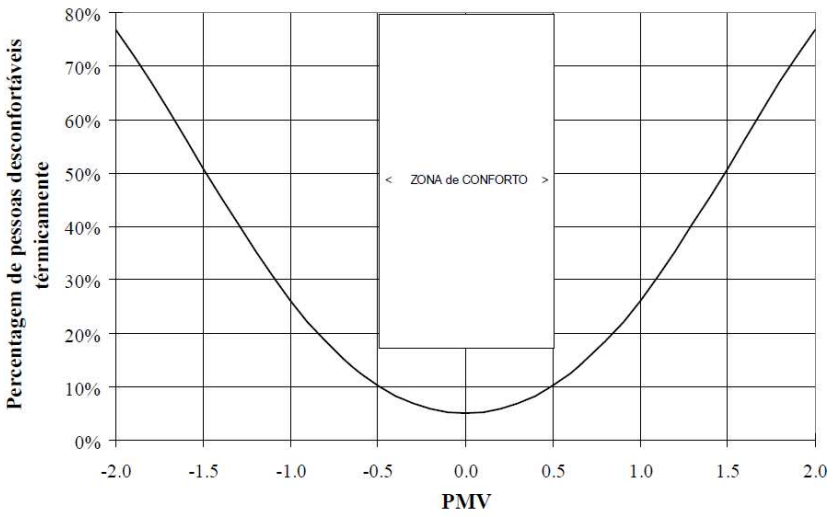

Figura 4- relação do índice PPD com o PMV.

2.3.6 OUTRAS CONSIDERAÇÕES DA ISO 7730

A norma do conforto térmico ainda leva em consideração para uma situação agradável um ambiente interno [6]:

- Ambientes térmicos aceitáveis são aqueles em que não mais que 10% dos ocupantes se sintam descontentes.
- A velocidade do ar, no inverno tem que ser inferior a 0,15 m/s, com temperaturas entre 20 e 24 ºC. No verão, a velocidade do ar deve ser inferior a 0,25 m/s com temperaturas entre 23 e 26 ºC.
- A diferença de temperatura do ar a 1,1 m e 0,1 m do chão não deve exceder 3 ºC.

A partir da norma ISO 7730 será determinada a temperatura ideal para garantir o conforto térmico do laboratório LAVSI. A temperatura será a única variável controlada, no entanto, pretende-se o controle da umidade do ar e da ventilação em trabalhos futuros.

3 HARDWARE

Este capítulo contempla o hardware utilizado para instalação da rede de sensores sem fio. Além de explicar brevemente os protocolos ZigBee e BACNet.

3.1 KIT DE DESENVOLVIMENTO MESHBEAN

O kit de desenvolvimento da Meshnetics (MeshBean2) contém um hardware com diversos periféricos que possibilitam o desenvolvimento de aplicações wireless com a tecnologia Zigbee. Dentre as funcionalidades disponíveis têm-se botões, DIP switches, sensores, leds, conector para JTAG e conectores para o uso de outros recursos do ZigBit.

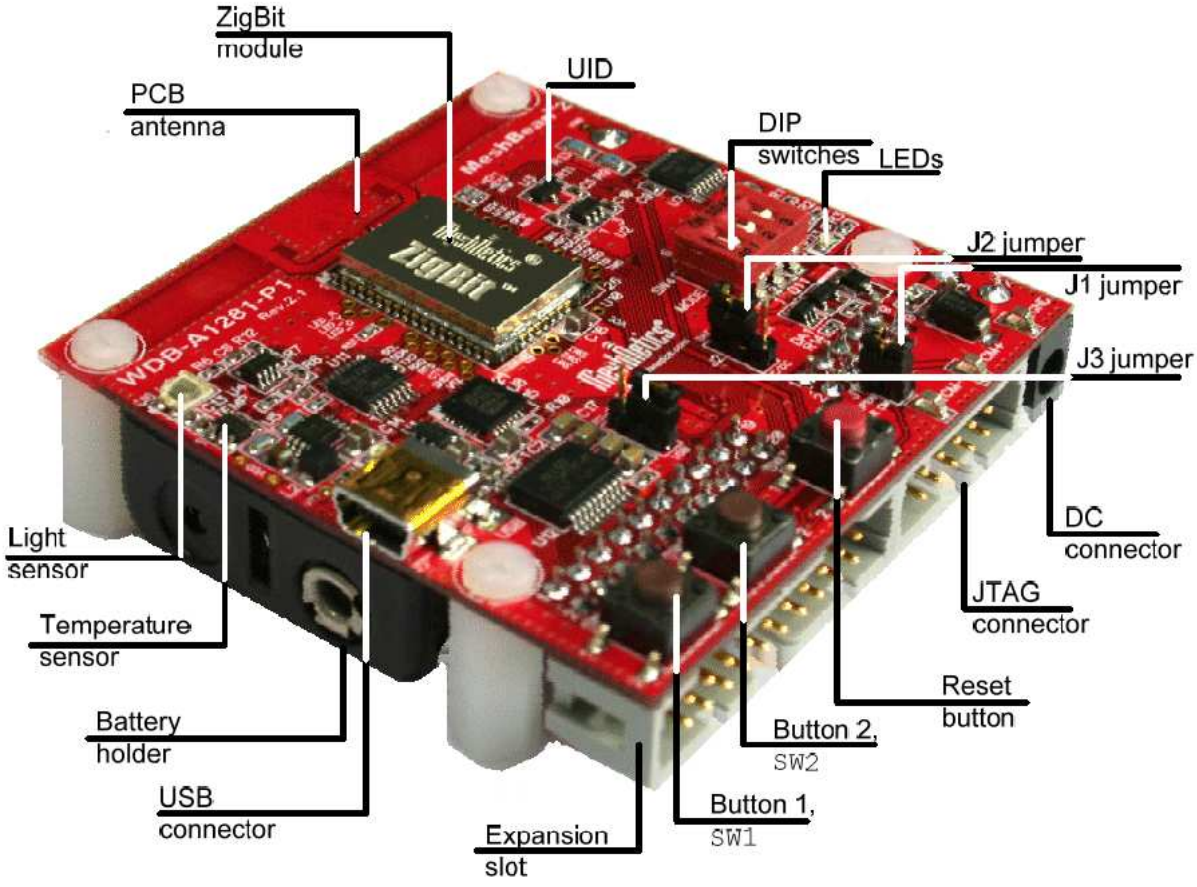

Figura 5- kit de desenvolvimento Meshbean2 [7].

O microcontrolador do kit vem de fábrica com um bootloader que permite carregar aplicações via USB a partir de uma aplicação em MS-DOS. Os kits vêm com uma aplicação demo carregada chamada WSNDemo, que implementa uma WSN e apresenta os dados em software hospedado em um computador. É possível utilizar comandos AT para os módulos ZigBit, contudo, os comandos AT são uma aplicação, o SerialNet. Como somente uma aplicação pode ser carregada no módulo, não é possível ter uma aplicação de usuário funcionando paralelamente aos comandos AT.

Há três sensores disponíveis no Kit: o de luminosidade TSL2550T da TAOS, o de temperatura LM73CIMK da National Semiconductors e o de bateria implementado com um divisor resistivo. Os dois primeiros estão conectados em paralelo ao barramento I2C e o último está ligado ao conversor AD.

Para proporcionar a comunicação USB o Kit possui o CP2102, conversor de USB para UART, da marca Silicon Labs. Este dispositivo faz a interface de RS-232 para USB. Quando conectado ao computador, o driver fornecido pela Silicon Labs permite que a porta USB seja visível pelo computador como uma porta COM de determinado número.

Alguns cuidados devem ser tomados em relação à utilização da placa, pois a incorreta posição dos jumpers com relação à alimentação e ao tipo de comunicação utilizados podem danificar o dispositivo permanentemente. Deve-se ainda atentar para o fato de que o hardware é sensível à eletricidade estática.

3.2 MÓDULO ZEGBEE ZIGBIT

O ZigBit é um módulo que contém em um mesmo encapsulamento o MCU ATmega1281 e o transceiver de RF AT86RF230. Existe a opção do módulo com a antena em chip e sem a antena. Este produto originalmente comercializado pela MeshNetics passou a ser comercializado e distribuído pela ATMEL por isso alguns componentes, principalmente de software, tiveram seus nomes alterados.

A MeshNetics disponibiliza três tipos de configuração da pilha: o ZigBeeNet agora BitCloud, SerialNet e OpenMAC. ZigBeeNet é uma plataforma de desenvolvimento de software ZigBee PRO certificado para aplicações wireless de tempo real com agendamento de tarefa. O SerialNet permite a manipulação dos módulos por comandos AT via interface serial. OpenMAC é uma implementação de código aberto da camada IEEE802.15.4 MAC para desenvolvedores experientes de software embarcado.

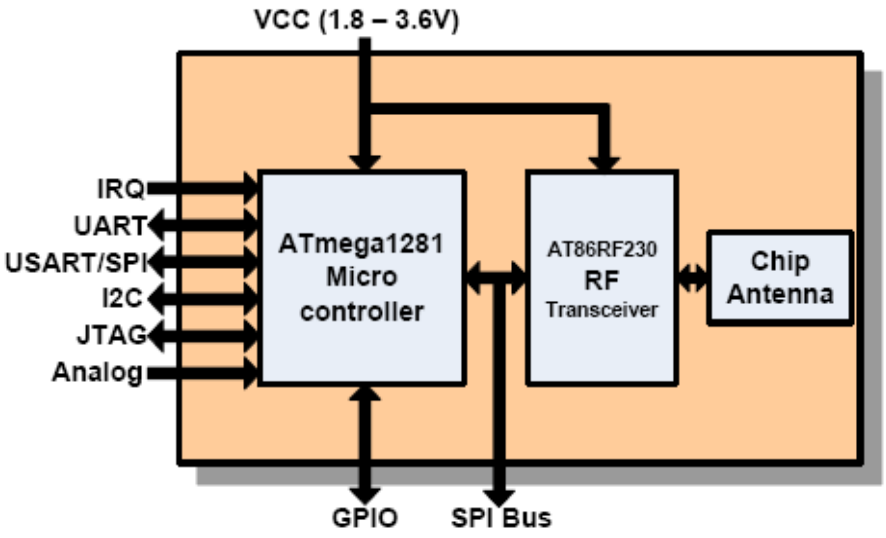

Figura 6- módulo ZigBit [8].

Algumas características do dispositivo são listadas na Tabela 3.

Tabela 3- características do módulo ZigBit.

Processing type	Valor	Unidade	Observações
Alimentação	1,8-3	V	
Freqüência mínima	2400	GHz	
Freqüência máxima	2483,5	GHz	
Número de canais	16	-	
Largura de Banda por canal	5	MHz	
Potência de saída	3-17	dBm	Ajustável em 16 níveis
Sensibilidade	-101	dBm	PER=1%
Taxa de transferência	250	Kbps	
Impedância saída (TX)/entrada (RX)	100	Ohms	Para saída balanceada.
Rejeição de canal adjacente	27	dB	
Rejeição de canal alternado	53	dB	
Alimentação	1.8-3	V	
Consumo de corrente RX	19	mA	
Consumo de corrente TX	18	mA	
Consumo de corrente modo *sleep*	6	mA	
Memória flash	128	kB	
Memória EPROM	4	kB	
Memória RAM	8	kB	
Temperatura de operação	40-85	°C	

O módulo zigbit apresenta algumas funcionalidades que merecem destaque, são elas:

- USART/SPI, I2C, 1-wire
- UART com controle CTS/RTS
- JTAG
- Até 25 GPIOs
- 2 entradas IRQ
- 4 entradas ADC

3.3 TECNOLOGIAS E PROTOCOLOS

Alguns padrões de comunicação importantes para este trabalho ou para aprimoramentos futuros são apresentados nesta sessão.

3.3.1 PROTOCOLO DE COMUNICAÇÃO BACNET

BACnet (*Building Automation and Control Networks*) é um protocolo de comunicação de dados aberto específico para automação e controle predial. A principal razão de utilizar este protocolo padronizado é a interligação dos diversos dispositivos utilizados em automação predial e também a comunicação destes com sistemas de supervisão predial [9]. O controle de equipamentos de condicionamento térmico é apenas um dos objetos de interesse em instalações prediais. Outros sistemas são: detecção e combate a incêndio, controle de iluminação, sistemas de segurança, alarmes, CFTV e automação de elevadores. Cada um desses sistemas possui diversos fabricantes e a utilização de um protocolo padrão possibilita a liberdade de escolha de diferentes marcas de equipamentos para o *retroffitting* das instalações. Dentre outras vantagens podem ser citadas o compartilhamento de dados entre os sistemas citados, a gestão de eventos e alarmes, o agendamento coordenado das operações do sistema, a comunicação com dispositivos remotos e a gestão de toda a rede.

O BACnet é importante para tornar os dispositivos desenvolvidos integráveis a qualquer dispositivo BACnet. No entanto devido à grande complexidade da implementação da pilha do BACnet e às limitações do gerenciador de tarefas disponibilizado pelo fabricante do Zigbit, o BACnet não foi implementado neste trabalho.

3.3.2 PROTOCOLO DE COMUNICAÇÃO SEM FIO ZIGBEE

O ZigBee é o padrão líder mundial para a implementação de redes de sensores sem fio de baixo custo, baixa taxa de transmissão, e curto alcance. A aliança ZigBee é um conjunto de companhias que trabalham juntas para desenvolver um padrão aberto e global para essas redes sem fio. A aliança ZigBee e o grupo do IEEE 802.15.4 são as fontes oficiais para implementação de redes sem fio baseadas no padrão IEEE 802.15.4 e ZigBee [10],

Dentre as principais tecnologias disponíveis para aplicações wireless, como Bluetooth e Wi-Fi, o ZigBee é a mais barata, menos complexa e com menor consumo de energia. Por isso é ideal para fins de automação predial.

Características dos sistemas baseados no padrão IEEE 802.15.4:

- Taxas de transmissão de 250 kbps, 40 kbps, e 20 kbps;
- Modos de endereçamento 16-bit *short* e 64-bit IEEE *addressing*;
- Suporte para dispositivos de latência crítica, com os joysticks;
- Acesso de canal do tipo CSMA-CA;
- Estabilização automática da rede pelo coordenador;
- Protocolo *handshaked* para transferência segura de dados;
- Gerenciamento de Energia para assegurar baixo consumo;
- 16 canais na banda de 2.4GHz ISM, e 10 canais na banda de 915MHz I e um canal na banda de 868MHz.

Visto as características do hardware e o padrão ZigBee é necessário detalhar os aspectos da programação utilizada para o desenvolvimento da aplicação. A próxima sessão apresenta o BitCloud, um conjunto de Bibliotecas e API fornecidas pelo fabricante do ZigBit para facilitar o desenvolvimento de aplicações para os módulos.

4 BITCLOUD

Apresenta uma noção geral da estrutura de software utilizada para proporcionar uma rede de sensores sem fio com múltiplas funcionalidades embarcadas em um único MCU.

4.1 INTRODUÇÃO

O BitCloud é um sistema de tempo real cooperativo multitarefa criado para facilitar o desenvolvimento de aplicações de redes sem fio embarcadas em hardware da Meshnetics. O BitCloud possibilitou a utilização de um único MCU nos módulos sensores, atuadores e coordenador. Por isso além de reduzir a complexidade do hardware, tornou-o flexível, uma vez que, por meio da aplicação, o desenvolvedor pode controlar as funcionalidades do módulo ZigBee.

4.2 ARQUITETURA DE PILHA DO SOFTWARE

A arquitetura do BitCloud segue as orientações para a separação de camadas conforme IEEE 802.15.4 e ZigBee. Além do núcleo da pilha, que implementa o protocolo, o BitCloud contém camadas adicionais para facilitar o desenvolvimento de aplicações do usuário, dentre elas, *task manager*, *security*,*power manager* e também camadas de abstração de hardware, *hardware abstraction layer* (HAL) e *board support package* (BSP) [11].

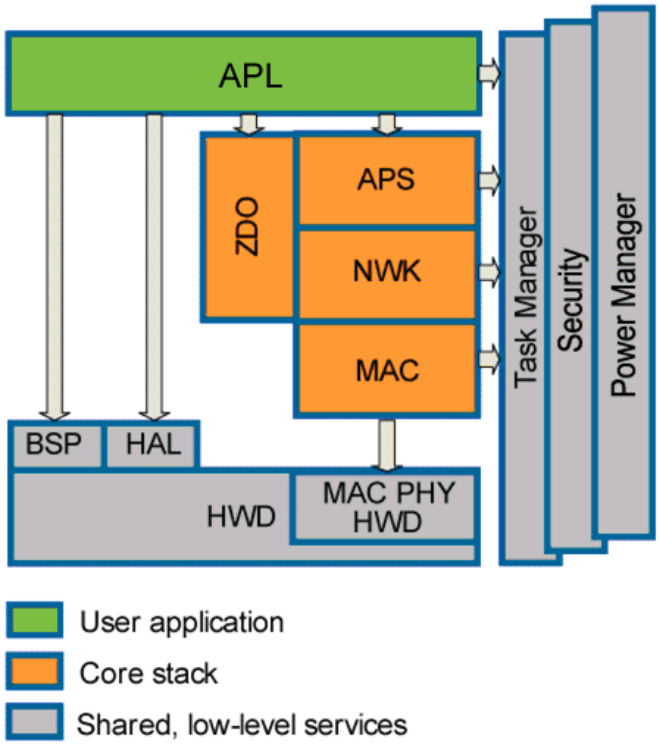

Figura 7- arquitetura da pilha do BitCloud [11].

Segundo *Bitcloud Stack Documentation* [12], segue a descrição das principais camadas da pilha do BitCloud.

No topo da pilha central, a *Application Support Sublayer* (APS) visível para a aplicação proporciona o mais alto nível das APIs relacionadas com a rede.

ZigBee Device Object (ZDO) são APIs que implementam a gestão da rede e a gestão de energia. ZDO também define o tipo do dispositivo, os comandos *device* and *services discovery* que permitem o dispositivo responder a requisições remotas e disponibilizam APIs para transmissão e aquisição de dados ponto a ponto, multiponto e por radiodifusão.

Existem três camadas de serviços *task maneger*, *security* e *power manager*. Estas funcionalidades estão disponíveis para aplicações do usuário e também podem ser utilizados por camadas de nível mais baixo.

O *Task manager* realiza o agendamento de tarefas que intermedeia o uso do MCU entre as camadas internas e a aplicação do usuário. O *Task manager* utiliza um algoritmo baseado em prioridade de fila (*priority queue-based),* especificamente desenvolvido para arquitetura de pilha multicamada e demandas de tempo crítico dos protocolos de rede.

Power management é responsável por desligar o dispositivo, salvar o estado do sistema quando prepara o dispositivo para dormir e restaurar o sistema quando o dispositivo acorda.

Hardware Abstraction Layer (HAL) contém APIs para o uso dos recursos de hardware (EEPROM, *app, sleep e watchdog timers*), bem como os *drivers* para facilitar a integração com uma série de periféricos externos (IRQ, I^2C, SPI, UART, 1-*wire*).

Board Support Package (BSP*)* contém um conjunto de *drivers* para gerenciar os periféricos (sensores, UID chip, chaves e botões) do kit de desenvolvimento MeshBean 2.

Configuration server é o componente da pilha que guarda os parâmetros de configuração chave e permite ao usuário reconfigurar esses parâmetros sem recompilar o núcleo das camadas da pilha. O usuário pode configurar parâmetros sem lidar com diversas versões de bibliotecas da pilha, por exemplo, pode-se configurar topologia de rede, *network fan-in*, o alcance, PAN ID, máscara de canal, etc.

4.3 NOMENCLATURA DO BITCLOUD

Para facilitar a o entendimento das funções pelo usuário, estas seguem uma nomenclatura descrita nas seis regras abaixo. O código desenvolvido para este trabalho segue a mesma convenção.

1. Cada nome de função é precedido pelo nome da camada onde a função reside, por exemplo, ZDO_GetLqiRssi.
2. Cada prefixo é seguido de um underline separando o prefixo do nome da função.
3. O nome descritivo da função deve ter o prefixo Get ou Set indicando a solicitação de um parâmetro retornado ou a modificação de um parâmetro da camada subjacente, por exemplo, HAL_GetSystemTime.
4. O nome descritivo da função deve ter o sufixo Req, Request, Ind, ou Conf indicando o seguinte:

    a. Req e Request correspondem a requisições assíncronas do usuário à pilha, por exemplo, APS_DataReq.
    b. Ind corresponde à indicações assíncronas ou eventos propagados da pilha para a aplicação do usuário, por exemplo, ZDO_NetworkLostInd.
    c. Por convenção, nomes de funções terminados em Conf são as interrupções definidas pelo usuário para requisições assíncronas que confirmam a execução da requisição.

5. Estruturas e nomes de tipos têm um sufixo _t.

6. Enumerações e macro variáveis são escritos com letras maiúsculas.

4.4 SISTEMAS DE TEMPO REAL.

Sistemas de tempo real são aqueles que devem satisfazer a limitações de resposta no tempo bem delimitadas, caso contrário correrão o risco de conseqüências graves, incluindo falhar [13]. Neste contexto a pilha do BitCloud e a aplicação deste trabalho caracterizam-se como sistemas de tempo real, pois o sistema tem de executar tarefas em períodos determinados e também lidar com eventos assíncronos.

Eventos síncronos são aqueles que ocorrem em determinadas posições no fluxo do programa. Já os eventos assíncronos são aqueles que ocorrem a qualquer instante independente do fluxo do programa. Mesmo uma interrupção de tempo que ocorre com um período preciso de tempo é assíncrona, pois pode ocorrer a independente da posição no fluxo do programa [13].

Existem diversas arquiteturas para sistemas de tempo real, entretanto é suficiente detalhar apenas algumas estratégias utilizadas pelo BitCloud para compreensão da dinâmica e das limitações das aplicações rodando sobre a pilha.

Phase-driven ou *State-driven* ou *Automata-driven* (FSA-driven) trabalha da seguinte forma: divide-se o código em blocos por meio de declarações *if them*, declaração *switch case* ou como uma máquina de estados finitos. Em cada bloco do código atualiza-se uma variável (*flag*) para marcar a posição atual do processo. Repete-se a chamada da função usando recursão e a flag indica o fluxo do programa. Assim o programa poderá ser interrompido e retornar à posição onde parou sem perder informações críticas [13].

Nos sistemas *round-robin* os processos são executados seqüencialmente. Cada processo tem um tamanho definido em termos de um período de tempo. Uma interrupção é iniciada com uma taxa correspondente ao tamanho dos processos a serem executados [13].

Um sistema cooperativo multitarefa exige uma programação disciplinada e uma aplicação apropriada. Estes tipos de Kernel são empregados em conjunto com um código guiado por uma máquina de estados finitos. Dois ou mais processos podem ser codificados conforme o modelo guiado por estados (*state-driven*). Quando cada fase é concluída, uma chamada é feita para o expedidor central. O expedidor seleciona o contador de programa para uma lista de processos que são executados no modelo *round-robin*, ou seja, ele seleciona o próximo processo a ser executado. Então antes de cada nova fase ser executada o expedidor é chamado novamente. Todas as informações que precisam ser armazenadas entre uma fase e outra são armazenadas em variáveis globais, assim é possível a comunicação entre processos [13].

4.5 CONTROLE DE FLUXO DO PROGRAMA

Para concentrar, em um único MCU, as atividades características de uma WSN, precisa-se de uma estratégia de programação que garanta o funcionamento adequado de todas as tarefas críticas. As limitações da plataforma de hardware utilizada não permitem um verdadeiro sistema operacional, portanto, o BitCloud utiliza um sistema conhecido como *Event-driven systems* [11].

Cada solicitação de função da API é relacionada a uma notificação assíncrona. O resultado da solicitação é entregue por meio de uma função de resposta (*callback*) chamada pela camada solicitada. O que a aplicação do usuário fornece para camadas inferiores é um ponteiro da função que é chamado quando a requisição é atendida.

O desacoplamento da requisição com a resposta é extremamente importante para esta aplicação, pois a execução do programa não precisa parar até que a resposta esteja pronta. Assim que o resultado da requisição for concluído será gerada uma interrupção que chamará a função de callback definida pelo usuário.

Todas as camadas do BitCloud são definidas em função de chamadas e suas correspondentes *callbacks* .Cada camada define um número de callbacks para as camadas inferiores invocarem e, por sua vez, invocam *callbacks* das camadas superiores [11].

4.6 MECANISMO DE COMFIMAÇÃO REQUISIÇÃO E INDICAÇÃO

A requisição é a chamada assíncrona para a camada inferior para executar alguma ação paralelamente à aplicação do usuário. A confirmação é realizada por meio de uma *callback* que é executada quando a ação está completa e o resultado da ação está disponível. A indicação também é uma *callback* só que executada em outro dispositivo. Como exemplo segue o caso de envio de uma mensagem entre dois nós da rede, inicialmente, o nó que envia a mensagem faz a requisição:

```
APS_DataReq(&apsDataReq);
```

O nó destinatário ao receber a mensagem tem a seguinte indicação executada:

```
void appCoordinatorDataInd(APS_DataInd_t* ind){
//código executado quando a mensagem chega.
}
```

O nó remetente executa a seguinte confirmação quando a mensagem é recebida pelo nó destinatário:

```
static void APS_DataConf(APS_DataConf_t *conf){
//código executado quando a mensagem chega.
}
```

Na confirmação, caso a mensagem não for enviada com sucesso, é possível tomar a decisão de enviar a mensagem novamente. Este fato pode ocorrer se, por exemplo, dois sensores mandarem mensagens em um intervalo de tempo no qual o coordenador não seja capaz de processá-las, logo um dos sensores deverá reenviar a mensagem para efetivar a comunicação. Maiores detalhes sobre aspectos da programação serão detalhados em capítulo específico.

Diante do exposto é possível compreender duas regras de programação impostas pelo fabricante.

Regra 1: As aplicações do usuário são organizadas como um conjunto de funções de *callback* executadas quando é completada a sua requisição à camada inferior [11].

Regra 2: A aplicação do usuário é responsável por declarar as funções de *callback* para lidar com eventos de seu interesse não solicitados pelo sistema [11].

4.7 ESCALAMENTO DE TAREFAS E PRIORIDADES

No sistema BitCloud existe apenas uma aplicação rodando no topo da pilha, assim as demandas por recursos do sistema não acontecem entre aplicações, mas entre uma única aplicação e as camadas da pilha. Ambas, pilha e aplicação executam seu código em um mesmo MCU [11].

Como o hardware em questão apresenta limitações quanto à memória não é possível armazenar todo o estado do sistema para poder interrompê-lo. Por isso longas rotinas não devem ser utilizadas nas funções de *callback.* Como exemplo se a *callback* ZDO_StartNetworkConf, que inicializa a rede, levar muito tempo para ser executada, o resto da pilha será efetivamente bloqueada esperando pelo término da *callback* para retornar o controle para a camada inferior. Note que as callbacks rodam com a prioridade da camada que as invocou, então ZDO_StartNetworkConf roda com a prioridade do nível ZDO. Logo, esta função de *callback* não pode demorar mais do que o tempo permitido ao custo de outras funções de prioridade maior ficarem bloqueadas e até mesmo serem perdidas. Conseqüentemente tem-se as regras 3 e 4 [11]:

Regra 3: Todas as funções de *callback* definidas pelo usuário devem ser executadas em 10ms ou menos.

Regra 4: A função de *callback* tem a prioridade da camada que a invocou.

Para executar funções de *callback* maiores do que 10ms é necessário utilizar o *task handler* usando a API *task manager*. Para tanto, preserva-se o estado atual da aplicação e posta-se uma tarefa na fila de tarefas.

O *task handler* também é uma função de *callback* que roda com a prioridade da camada da aplicação do usuário. Logo, as instruções executadas pelo *task handler* só serão realizadas quando todas as instruções das outras camadas prioritárias terminarem. Por isso são permitidas instruções maiores no task handler. No entanto, deve-se observar as regras 5 e 6 [11]:

Regra 5: Aplicações do *task handler* serão executadas somente quando todas as tarefas de prioridades superiores forem completadas.

Regra 6: Toda aplicação do *task handler* deve ser executada em até 50ms.

Basicamente pode-se fazer o controle de fluxo da aplicação de três formas: através do task handler seguindo as invocações do SYS_PostTask; confirmando callbacks invocadas pelas camadas inferiores da pilha em atendimento a alguma requisição e por meio de eventos de notificação assíncrona invocados pela pilha.

Existem outras regras para a programação utilizando a pilha do BitCloud são elas [11]:

- Partes críticas da aplicação não podem levar mais do que 50 μs;
- Uma estrutura inteira nunca deve ser passada para a pilha, ou seja, usada como argumento de uma função. Deve-se passar o ponteiro da estrutura. O mesmo vale para funções definidas pelo usuário que recebam estruturas como parâmetro;
- Preferencialmente deve-se usar variáveis globais ao invés de variáveis locais quando possível;
- A aplicação do usuário não pode usar funções recursivas. Além disso não é permitido mais do que 10 chamadas em seqüência, sendo uma função chamando outra. Essas funções não podem ter mais do que dois parâmetros como ponteiro de estrutura cada uma.
- Alocação dinâmica de memória não é permitida. O usuário deve utilizar as funções padrão malloc, calloc, e realloc da biblioteca C;
- Os recursos do hardware reservados para o uso da pilha não devem ser acessados pela aplicação do usuário;

5 DESENVOLVIMENTO

Apresenta o sistema desenvolvido e a aplicação embarcada. A interface gráfica do software supervisório também é apresentada neste capítulo.

5.1 SISTEMA PROPOSTO

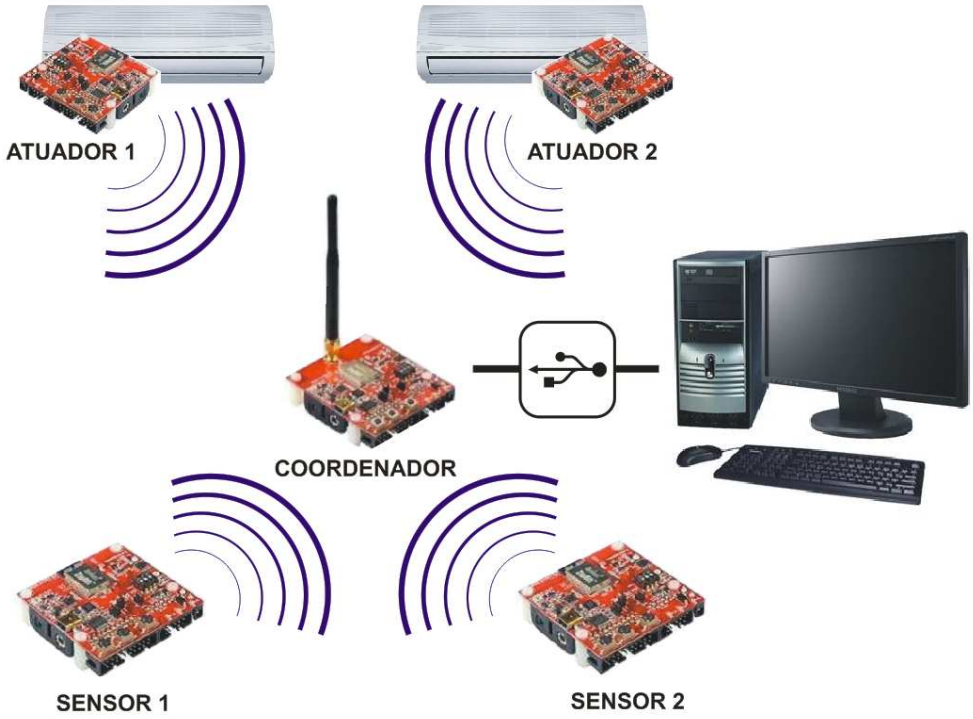

Figura 8- ilustração da rede de sensores proposta.

Com o objetivo de operar o sistema de ar condicionado do LAVSI de modo automático e manter o conforto térmico, foi proposto o sistema ilustrado na Figura 8.

Inicialmente, os sensores que estão dormindo acordam para realizar medidas de temperatura e bateria feitas a cada 10 segundos. Após a leitura os dados são enviados para o coordenador e os sensores voltam a dormir. A leitura de bateria possibilita ao software supervisório indicar o momento da troca.

Os atuadores posicionados acima dos equipamentos de ar condicionado também enviam medidas de temperatura a cada 10 segundos. Essas medidas foram incluídas para comparar o sistema proposto com o sistema de controle do fabricante, uma vez que é essa a temperatura que o ar condicionado considera para acionar o compressor.

O coordenador é responsável por enviar os dados via USB para um software em um computador. O software avalia se os equipamentos devem ser ligados ou não e manda esta informação de volta ao coordenador. Os atuadores são os destinatários desta informação e atuam sobre um relé que liga ou desliga o equipamento.

5.2 AMBIENTE CONTROLADO

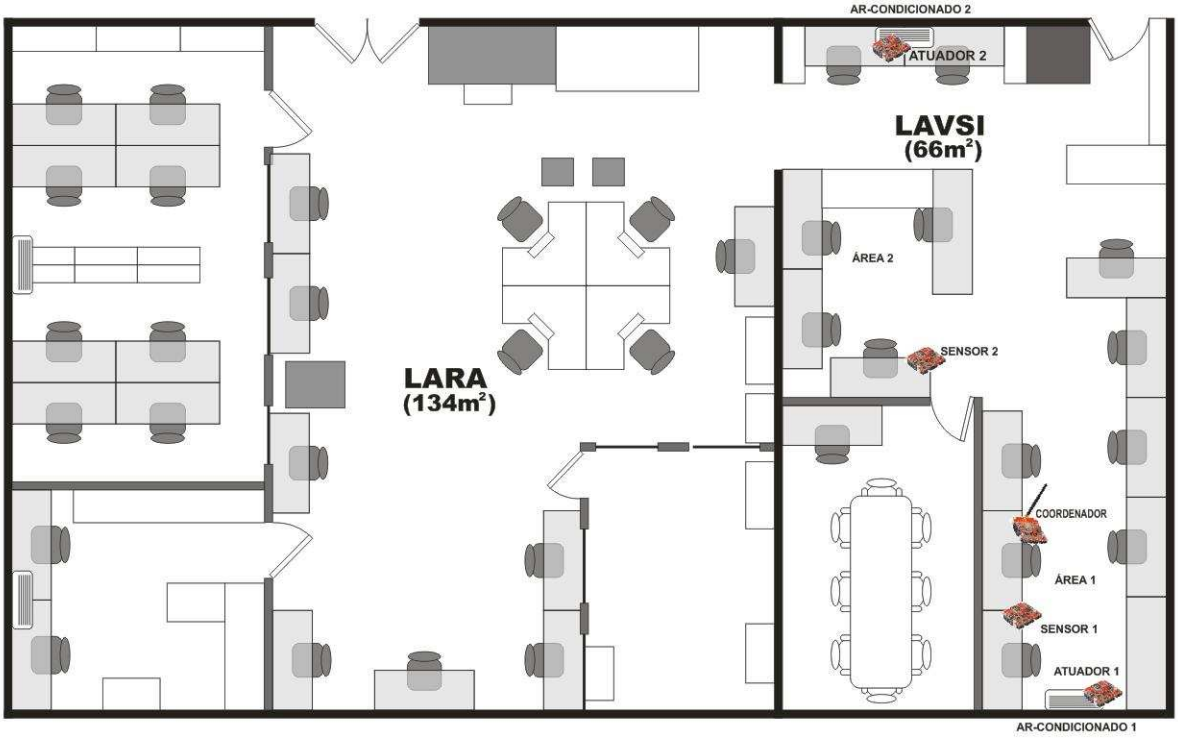

Figura 9 - laboratório de pesquisa LAVSI.

O ambiente onde o sistema foi implantado está representado pela Figura 9. O laboratório dispõe de dois equipamentos de ar condicionado tipo split, da marca Springer, modelo MAXIFLEX, com capacidade de 22000 BTUs/h cada. O laboratório LAVSI apresenta uma passagem para o laboratório LARA. O LARA não possui aparelhos de ar condicionado e conseqüentemente interfere no sistema de forma significativa, pois existe troca de calor entre os ambientes.

Os equipamentos de ar condicionado só possibilitam um nível de controle, compressor ligado ou compressor desligado. Não é possível selecionar um nível de operação para manter a temperatura do ambiente, então o controle é feito ligando e desligando o compressor ao longo do tempo de operação.

O objetivo da rede de sensores sem fio é medir o nível de conforto do ambiente e possibilitar a automação do sistema para economizar energia. Com a automação, o sistema pode evitar os excessos dos usuários e fazer o uso racional da energia. É nesse aspecto que o sistema proposto diferencia-se do convencional.

5.3 REQUISITOS BÁSICOS

Com o intuito de aprimorar o projeto descrito no item 1.3, o novo sistema deve possuir maior memória flash para suportar o protocolo BACnet e a aplicação do usuário embarcados. Assim, o módulo ZigBit foi escolhido como módulo transmissor por possuir um microcontrolador com 128k de memória flash.

Os requisitos de projeto são os seguintes:

- Mesmo hardware para sensores, atuadores e coordenador;
- Utilização de um único MCU por hardware;
- Os sensores devem entrar em modo *sleep* após o ciclo de leitura e envio de dados para economia de bateria;
- Medidas de temperatura feitas de 10 em 10 segundos;
- Os nós atuadores e coordenador serão alimentados por fonte de alimentação e não serão desligados;
- O software supervisório instalado em um computador deve receber os dados;

5.4 APLICAÇÃO EMBARCADA

A aplicação lowpower.srec foi desenvolvida para todos os módulos Meshbean não dependendo da função do módulo na rede. O endereçamento e a definição da função do módulo como coordenador, sensor ou atuador é feita por meio do valor atribuído aos DIP switches.

A aplicação foi dividida em três arquivos principais:

- lowpower.c é a parte inicial comum para todos os módulos, responsável por configurar a rede e determinar o funcionamento de cada módulo;
- coordinator.c é a parte do aplicativo específica para o módulo coordenador da rede;
- enddevice.c é a parte do aplicativo específico para os módulos sensores e atuadores;

A programação utilizando o BitCloud tem o fluxo de programa orientado por estados. Os estados da aplicação deste trabalho estão representados na Figura 10.

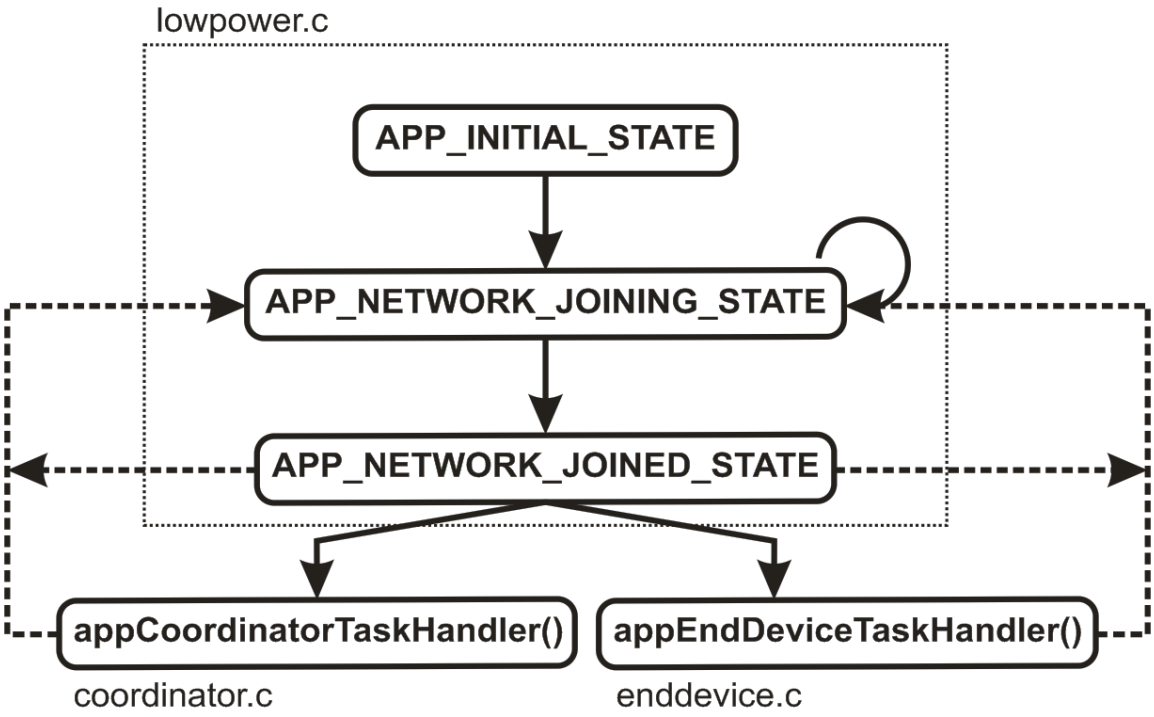

Figura 10- diagrama de estados do aplicativo embarcado.

Os estados possíveis para o módulo são determinados a partir dos itens de uma estrutura denominada AppState_t. Os itens desta estrutura podem ser observados no trecho de código abaixo:

```
typedef enum
{  APP_INITIAL_STATE,                    // Estado inicial da aplicação (após ligar ou resetar)
   APP_START_WAIT_STATE,                 // Aguardando pressionar o botão 0 do módulo
   APP_NETWORK_JOINING_STATE,            // Estado em que o módulo entra para a rede
   APP_NETWORK_JOINED_STATE,             // Rede criada e modulo inserido na rede
 } AppState_t;
```

Estes estados são comuns a todos os módulos e são utilizados apenas na inicialização dos módulos. Após a determinação do tipo de dispositivo na rede para cada MeshBean os estados passam a seguir a seguinte estrutura:

```
typedef enum
{
   DEVICE_ACTIVE_IDLE_STATE,             // Dispositivo ativo (acordado)
   DEVICE_MEASURING_STATE,               // Adquirindo dados dos sensores
   DEVICE_MESSAGE_SENDING_STATE,         // Estado de envio de dados ao coordenador
   DEVICE_SLEEP_PREPARE_STATE,           // Estado de preparação para dormir (enddevice)
   DEVICE_SLEEP_STATE,                   // Estado em que o enddevice dorme
   DEVICE_AWAKENING_STATE                // Módulo enddevice saindo do modo sleep
} AppDeviceState_t;
```

O aplicativo inicia a estrutura indexada no estado inicial, utilizando o seguinte código:

```
AppState_t appState = APP_INITIAL_STATE;
```

O estado futuro é atualizado durante o fluxo do estado atual a partir da atualização da variável appState. Ao invocar a função SYS_PostTask(APL_TASK_ID), posta-se uma nova tarefa no APL_TaskHandler() e o fluxo do programa segue para o próximo estado.

Quando os dispositivos estiverem conectados a rede, o estado atual será APP_NETWORK_JOINED_STATE. O aplicativo passa a chamar as funções específicas de cada tipo de módulo seguindo a estrutura de estados denominada AppDeviceState_t.

O código utilizado pode ser observado no ANEXO I. As funções de cada estado e os diagramas de estado dos módulos coordenadores e enddevice serão explicados detalhadamente.

5.4.1 LOWPOWER.C

O estado inicial (APP_INITIAL_STATE) do módulo, possui uma única função, a initApp(), que realiza a leitura dos *DIP switches* e atribui este valor ao endereço do módulo na rede. Se este endereço for zero o módulo irá funcionar como coordenador invocando a função appCoordinatorInit() do arquivo coordinator.c, caso contrário o módulo irá funcionar como dispositivo final invocando a função appEndDeviceInit () do arquivo enddevice.c . O papel do dispositivo na rede é atribuido da seguinte forma:

```
appDeviceType = DEVICE_TYPE_COORDINATOR;      //Se endereço igual a zero
appDeviceType = DEVICE_TYPE_END_DEVICE;       //Se endereço diferente de zero
```

Estes parâmetros são repassados para a configuração da rede de acordo com o trecho de código:

```
CS_WriteParameter(CS_NWK_ADDR_ID, &nwkAddr);              //Atribui o endereço na rede
CS_WriteParameter(CS_DEVICE_TYPE_ID, &appDeviceType);     //Atribui o tipo de dispositivo
```

Após estas configurações, a função initApp() invoca a função appOpenButtons(NULL, buttonReleased) que aguarda um dos botões do módulo MeshBean ser pressionado e liberado para atualizar o estado da estrutura.

Se o botão pressionado for o botão 1, o estado é atualizado para APP_NETWORK_JOINING_STATE, se o botão pressionado for o 2 e o dispositivo estiver trabalhando como *end device*, o estado do dispositivo passa a ser DEVICE_AWAKENING_STATE.

No APP_NETWORK_JOINING_STATE, a função responsável pela requisição de criação da rede é invocada (startNetwork()). A função ZDO_StartNetworkReq() do BitCloud faz a requisição da rede conforme as configurações passadas por um ponteiro de uma estrutura. Uma destas configurações é a função de *callback* executada quando o procedimento é finalizado. A *callback* definida permite o tratamento da falha ou a execução do próximo passo em caso de sucesso.

Quando a função definida como *callback* é chamada ela recebe o status da conexão. Se o status for ZDO_SUCCESS_STATUS, o estado do dispositivo é atualizado para APP_NETWORK_JOINED_STATE e posteriormente DEVICE_ACTIVE_IDLE_STATE. Caso o status seja ZDO_FAIL_STATUS , o estado da rede se mantém em APP_NETWORK_JOINING_STATE realizando nova tentativa de criação da rede.

A qualquer momento durante o funcionamento da rede, se por algum motivo ocorrer uma falha na rede, a função ZDO_MgmtNwkUpdateNotf () é invocada fazendo com que o dispositivo restabeleça a comunicação.

No estado APP_NETWORK_JOINED_STATE, o modulo é direcionado para os arquivos correspondentes ao tipo de dispositivo configurado, se configurado como coordenador a *Task Handler* appCoordinatorTaskHandler() passará a administrar os estados do módulo, caso contrario será chamada a *Task Handler* appEndDeviceTaskHandler.

5.4.2 COORDINATOR.C

No caso do coordenador, a função appCoordinatorInit(void) foi chamada pelo estado inicial do arquivo lowpower.c. Esta função tem como objetivo configurar os parâmetros de comunicação serial com o computador e de menssagens enviadas pela rede ZigBee, utilizando respectivamente as funções usartInit() e messageInit().

A *Task Handler* do dispositivo configurado como coordenado (endereço zero adquirido nos *DIP switches*) possui um único estado o DEVICE_ACTIVE_IDLE_STATE. Este estado apenas inicia um *timer* para que a cada 2 segundos, os dados coletados pela rede sejam enviados para um computador. O coordenador foi configurado para enviar dados a essa taxa somente para manter a conexão com o software MATLAB. Todo o fluxo de funções deste dispositivo é determinado pelas interrupções e funções de *callback* configuradas.

Os dados recebidos pela rede são atualizados dentro da função appCoordinatorDataInd(), que é uma função de *callback*, esta função indica o recebimento de um pacote pela rede. Devido ao fato da utilização de módulos funcionando como coordenador e *end device*, as funções de indicação de mensagem foram subdivididas respectivamente em appCoordinatorDataInd() e appEndDeviceDataInd(), determinadas no arquivo lowpower.c na função específica da pilha do BitCloud, a APS_DataInd().

A mensagem é enviada para a rede através da seguinte estrutura:

```
typedef struct                    // Mensagem da aplicação
{
 int16_t temperature;             // Variável para temperature dos módulos
 int16_t battery;                 // Variável para as baterias dos sensores
 int8_t control;                  // Parâmetro para atuação do compressor
 int8_t porta;                    // Estado da porta de atuação nos atuadores
} PACK AppMessage_t;
```

Esta estrutura é utilizada por todos os módulos. Quando ocorre o recebimento deste pacote pela rede, o coordenador armazena as variáveis de temperatura, bateria e porta de acordo com o endereço de cada módulo responsável pelo envio.

Os dados enviados pelo aplicativo supervisório provocam interrupções *byte* a *byte* no módulo coordenador invocando a função de *callback* definida nos parâmetros de inicialização da UART, a appReadByteEvent(). O software supervisório envia dois sinais de controle, atribuídos pelo coordenador às variáveis atuador_1 e atuador_2. A **Tabela 4** descreve os sinais de controle que podem ser recebidos pelo coordenador através da UART.

Tabela 4- sinais de controle recebidos pela UART no coordenador.

Atuador_1	Atuador_2	Comando para split 1	Comando para split 2
0	0	Desligar	Desligar
0	1	Desligar	Ligar

1	0	Ligar	Desligar
1	1	Ligar	ligar

Caso ocorra algum erro no envio do sinal de controle pelo software supervisório, os dados são descartados, considerando apenas o envio de bytes como zero ou um.

Ao ocorrer a interrupção do timer inicializado, a função sendUsartMessage(), determinada como função de *callback* do timer, é invocada enviando os dados armazenados de temperatura e bateria para a UART. Nesta mesma função, ocorre o envio do sinal de atuação para os módulos atuadores. Este envio ocorre apenas se o estado da porta estiver diferente do sinal recebido pela UART.

A atuação dos módulos ocorre de forma intercalada, ou seja, a cada interrupção muda-se o destinatário da mensagem do coordenador. Ora o atuador 1 recebe a mensagem, ora o Atuador 2 recebe a mensagem.

As funções sendControlMessageAt1() e sendControlMessageAt2() enviam o sinal de controle para os módulos atuadores 1 e 2 respectivamente. O envio de mensagem ocorre apenas se a rede estiver liberada (os módulos não podem enviar mensagens simultaneamente). O mecanismos de liberação e ocupação da rede são definidos pela seguinte estrutura:

```
typedef enum                          // Estado de transmissão da rede
{
  APP_DATA_TRANSMISSION_IDLE_STATE,   // Rede desocupada
  APP_DATA_TRANSMISSION_BUSY_STATE    // Rede ocupada
} AppDataTransmissionState_t;
```

O envio de mensagem é realizado por meio do mecanismo de requisição, indicação e confirmação. Quando a mensagem é enviada, é executada no coordenador a *callback* de confirmação APS_DataConf(APS_DataConf_t *conf). Esta recebe o status do envio e repete a ação se o status for negativo.

5.4.3 ENDDEVICE.C

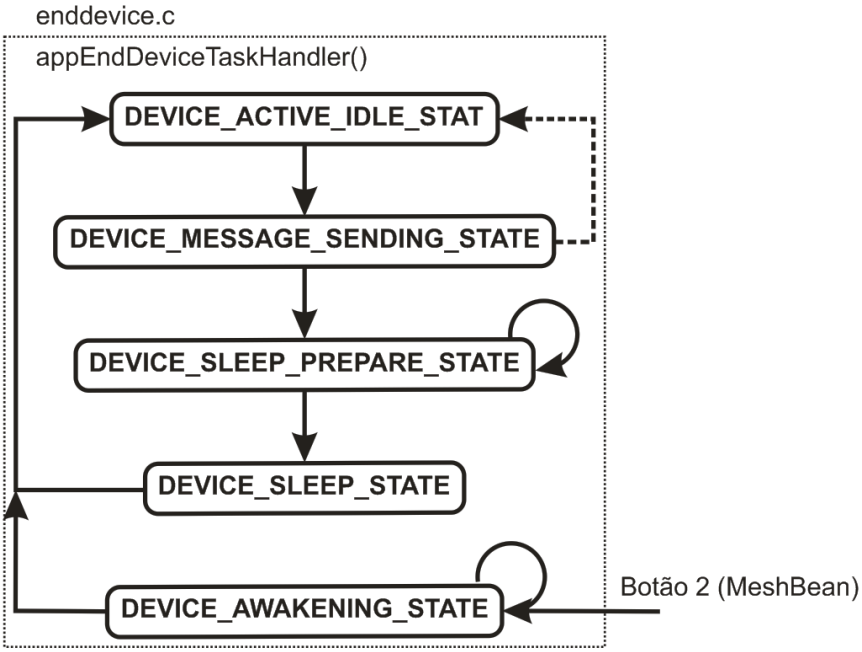

Figura 11- diagrama de estados do dispositivo configurado como *end device*.

Análogo ao funcionamento do dispositivo configurado como coordenador, inicialmente a função appEndDeviceInit () é chamada pelo estado inicial comum a todos os módulos. Esta função observa o valor do endereço na rede atribuído pelos *DIP switches*. Se este endereço for maior do que 2, o módulo será um dispositivo final atuador e se for menor do que 2, o módulo será um dispositivo final sensor. Em seguida ocorre também a configuração da comunicação com a rede, e inicialização dos sensores para a realização das medidas.

O diagrama de estados do funcionamento deste tipo de dispositivo pode ser observado na Figura 11. O estado inicial do dispositivo definido como APP_NETWORK_JOINING_STATE no arquivo lowpower.c passa ser DEVICE_ACTIVE_IDLE_STATE no arquivo enddevice.c. Neste estado ocorre a realização da leitura das medidas de temperatura e de bateria, que são originadas do sensor LM73 disponível no kit de desenvolvimento e do conversor analógico digital do ZigBit respectivamente.

Para a leitura da temperatura ambiente, Utilizou-se a seguinte função do BitCloud:

```
BSP_ReadTemperatureData(temperaturesSensorHandler);
```

Esta função recebe como parâmetro uma função do tipo *void*, que por sua vez deve possuir como parâmetros uma variável booleana e uma do tipo inteiro. A variável booleana informa se a leitura do valor de temperatura foi realizada com sucesso ou se houve uma falha. A variável do tipo inteiro recebe o valor da temperatura ambiente medida pelo módulo sensor. Por se tratar de números inteiros, a resolução dos dados foi de 1ºC.

Se a leitura tiver sido realizada com sucesso, o valor da temperatura será atribuído à variável correspondente na estrutura de mensagem da rede, caso contrário esse valor será atribuído como nulo.

De forma análoga, a leitura do nível de bateria pode ser realizada a partir da seguinte função:

```
BSP_ReadBatteryData(batterySensorHandler);
```

Que recebe como parâmetro novamente uma função do tipo *void*, que por sua vez possui como parâmetro uma variável do tipo inteiro à qual será atribuído o valor lido pelo conversor analógico-digital.

Após a leitura do nível de bateria e atribuição ao parâmetro correspondente da estrutura de mensagem da rede, ocorre a atualização do estado do dispositivo passando o estado para DEVICE_MESSAGE_SENDING_STATE. Neste estado ocorre primeiramente a leitura do estado da porta de atuação (se o dispositivo estiver configurado como atuador) e a atualização desta informação na variável da estrutura de mensagem. Em seguida, ocorre o envio dos dados para o coordenador.

Se houver falha no envio dos dados ao coordenador, o dispositivo retorna para o estado _ACTIVE_IDLE_STATE realizando novamente as leituras dos dados nos sensores e uma nova tentativa de envio. Se a mensagem for enviada com sucesso o estado do dispositivo é atualizado para DEVICE_SLEEP_PREPARE_STATE, onde a função prepareToSleep() é chamada para preparar o sistema para entrar em modo *sleep*.

Os módulos atuadores não entram no modo *sleep*, pois precisam manter o valor da porta de saída para manter o sistema de ar condicionado ligado ou desligado. Então, no estado DEVICE_SLEEP_PREPARE_STATE, os atuadores não chamam as funções que colocam o dispositivo em modo *sleep.* Para manter a operação de leitura e envio de dados é iniciada uma interrupção de tempo responsável por manter o fluxo do programa. Esta interrupção tem o mesmo período de tempo do modo sleep.

Para os módulos sensores, no estado DEVICE_SLEEP_PREPARE_STATE, ocorre a requisição para a camada ZDO do BitCloud para entrar em modo de economia de energia (modo *sleep*). Se houver um retorno de aceitação da requisição, o dispositivo executa o fechamento de todos os seus periféricos e tem o estado atualizado para DEVICE_SLEEP_STATE, caso contrário, o estado permanece em DEVICE_SLEEP_PREPARE_STATE ocorrendo nova requisição para o modo sleep.

O dispositivo deixa o estado DEVICE_SLEEP_STATE apenas quando o timer configurado para manter o dispositivo em modo *sleep* gera uma interrupção invocando a função de *callback* ZDO_WakeUpInd(), que por sua vez, chama a função wakeUpHandler(), que executa a abertura dos periféricos do microcontrolador nos módulos sensores. Após este processo o dispositivo volta para o estado DEVICE_ACTIVE_IDLE_STATE fechando o ciclo.

Em qualquer momento do funcionamento do módulo end-device, se ocorrer o pressionamento e liberação do botão 2 do MeshBean, o estado do dispositivo é atualizado para DEVICE_AWAKENING_STATE onde ocorre a solicitação para que o módulo deixe o modo *sleep*. O trecho de código abaixo mostra as funções responsáveis por acordar o dispositivo.

```
zdoWakeUpReq.ZDO_WakeUpConf = ZDO_WakeUpConf;   //Callback para a requisição
ZDO_WakeUpReq(&zdoWakeUpReq);                   //Enviando requisição
```

Se houver a confirmação da camada ZDO do BitCloud, a função wakeUpHandler() é invocada. Se a requisição for negada, o estado se mantém em DEVICE_AWAKENING_STATE.

5.5 SOFTWARE DE SUPERVISÃO

Figura 12- tela do supervisório.

Para a administração da rede, monitoramento dos parâmetros enviados pelo coordenador e controle dos aparelhos de ar-condicionado foi desenvolvido um software em linguagem Visual Basic cuja interface gráfica pode ser observada na Figura 12.

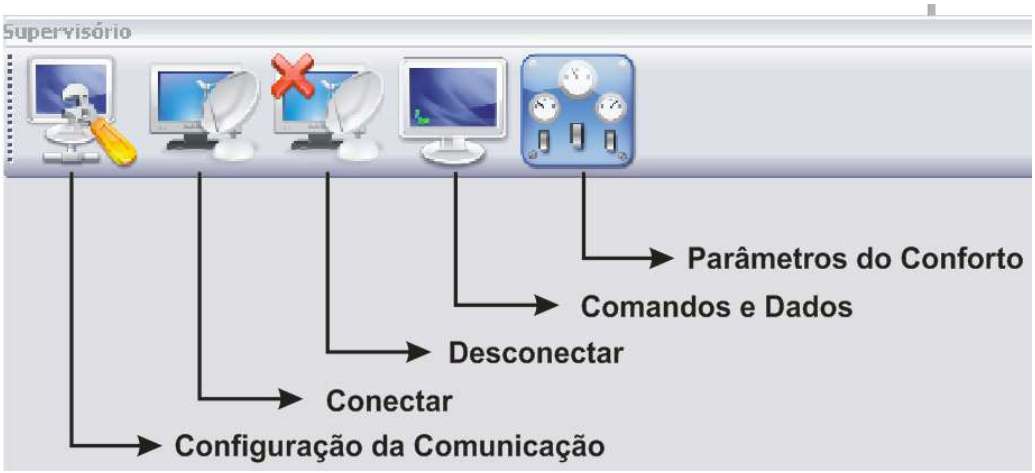

Figura 13- funções dos botões do software supervisório.

Para a inicialização da comunicação entre o computador e o módulo coordenador é necessário definir os parâmetros na seção de “configuração da comunicação” no supervisório. Os valores padrão utilizados pelo coordenador podem ser vistos na Figura 14. Para estabelecer a comunicação é necessário definir somente o número da porta COM, que varia de computador para computador.

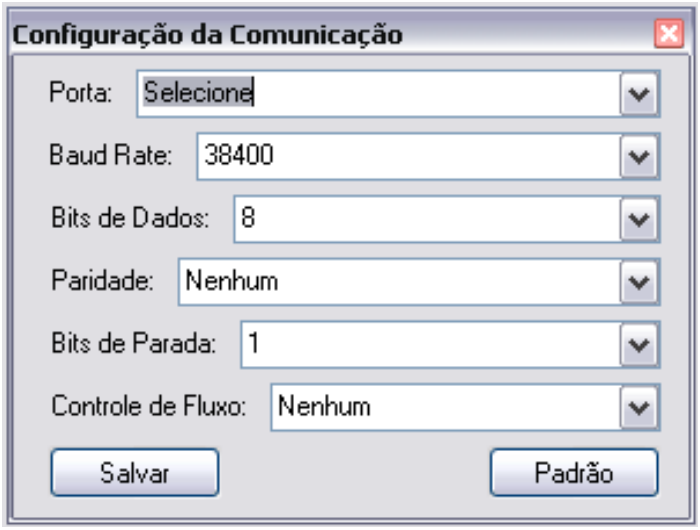

Figura 14- tela de configuração da comunicação do supervisório.

Após a configuração destes parâmetros, podem-se configurar os parâmetros necessários para o cálculo do conforto térmico na seção "parâmetros do conforto" do software. Como o objetivo do presente projeto é controlar a temperatura do laboratório LAVSI os valores para velocidade do vento, temperatura média radiante e umidade do ar foram fixadas de acordo com os dados medidos no laboratório. Desta forma estes dados não são atualizados em tempo real. Os valores padrões definidos no software podem ser observados na Figura 15.

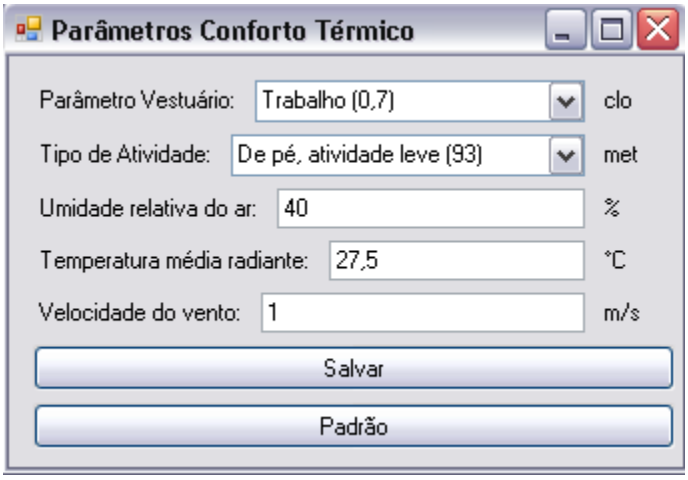

Figura 15- tela de configuração dos parâmetros do conforto.

Após as configurações prévias do software supervisório, é preciso clicar no botão referente a "conectar" na Figura 13 para inicializar a comunicação com o módulo coordenador através de uma porta COM. Os dados enviados para a porta USB em intervalos de 2 segundos geram uma interrupção no supervisório para o tratamento dos valores recebidos. Um histórico dos dados recebidos e enviados é exibido em tempo real na seção "comandos e dados" do software, como pode ser observado na Figura 16.

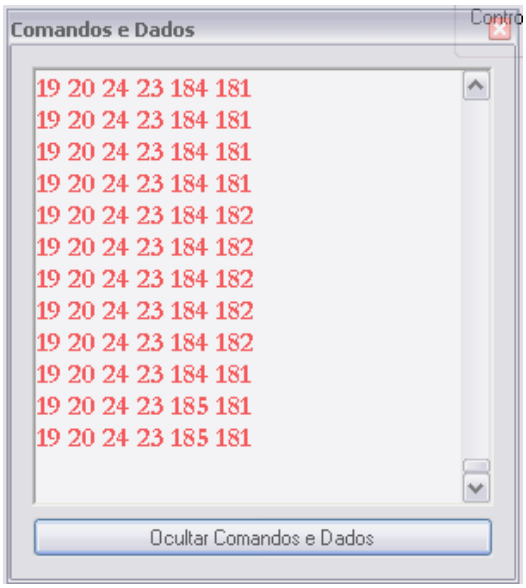

Figura 16- tela de dados e comandos do software supervisório.

O coordenador envia os valores recebidos da rede de temperatura e nível de bateria a cada dois segundos. Os valores recebidos são:

- temperatura do sensor 1;
- temperatura do sensor 2;
- temperatura do atuador 1;
- temperatura do atuador 2;
- nível de bateria do sensor 1;
- nível de bateria do sensor 2;

Após o armazenamento destes valores em variáveis específicas, ocorre o cálculo do nível de bateria em volts, assim como o cálculo do percentual em relação à tensão nominal das baterias utilizadas nos sensores. Ocorre também o cálculo da equação de conforto e dos índices PMV e PPD. Estes valores são, em seguida, armazenados em uma planilha de dados.

Os valores de temperatura e set-point são exibidos de forma gráfica, como pode ser observado na Figura 17. Todos os parâmetros calculados são também exibidos em mostradores específicos como evidenciado na Figura 18.

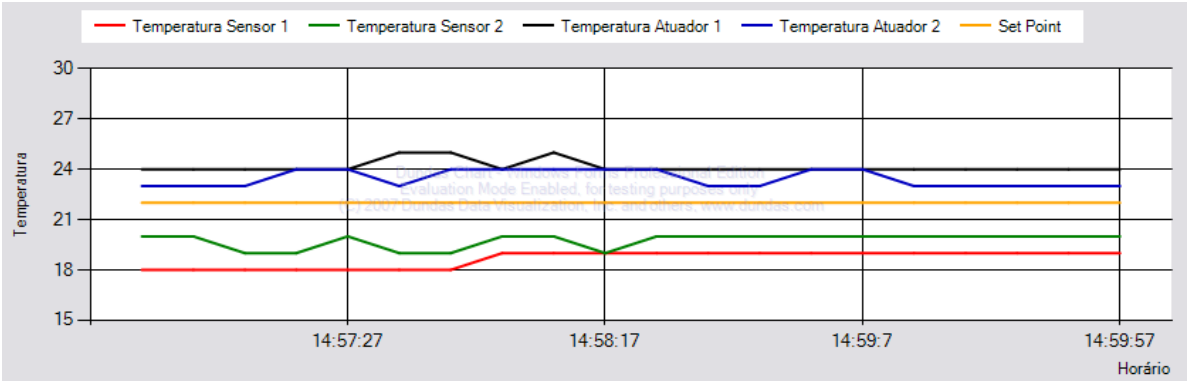

Figura 17- gráfico de temperaturas exibido em tempo real no software supervisório.

Figura 18- exibição dos dados dos ambientes do LAVSI.

O controle utilizado pelo aplicativo é do tipo *liga-desliga* com 2º C de histerese. Para um set-point de 22º C, o aplicativo envia o comando para desligar um aparelho de ar-condicionado, quando o sensor envia um valor de temperatura para o aplicativo de 21º C. Após desligado, o aplicativo apenas envia o comando para ligar novamente o compressor do aparelho, quando a temperatura do ambiente chegar a 23º C. Todos os procedimentos de atuação dos compressores são precedidos de alteração da imagem de status do compressor e inserção na planilha de dados do estado atual (ligado ou desligado).

Para exemplificar o funcionamento do software, podemos observar um conjunto de dados enviados pelo módulo coordenador. Os dados após o tratamento do aplicativo podem ser observados na Tabela 5.

Tabela 5- exemplo de dados do software supervisório após tratamento.

Parâmetros	**Setor 1**	**Setor 2**
Temperatura do sensor ºC	19	20
Temperatura do atuador ºC	24	23
Nível de bateria %	87,4	85,45
Índice PMV	-0,44	-0,26
Índice PPD	8,97	6,4
Status do compressor	desligado	desligado

Para a determinação do nível de bateria em volts, a seguinte função deve ser utilizada:

$$nível\ de\ bateria = \frac{3.1,25.(valor\ da\ leitura\ de\ bateria)}{256}\ V$$

$$nível\ de\ bateria = \frac{3.1,25.(valor\ da\ leitura\ de\ bateria)}{3.256}.100\ \%$$

Trechos relevantes do código do aplicativo supervisório podem ser vistos no Anexo II.

6 ANÁLISE DOS RESULTADOS

Os resultados com a rede de sensores desenvolvida são apresentados. Este capítulo contém também a conclusão e as perspectivas para os próximos trabalhos.

6.1 DETALHES DO EXPERIMENTO

O BitCloud possui a camada BSP para manipulação dos sensores e outros periféricos da placa de desenvolvimento MeshBean 2. A função que implementa a leitura do sensor de temperatura permite uma resolução de 1 °C. Apesar de a resolução ser pequena, ela é suficiente do ponto de vista do sistema. Isso porque, o compressor do ar condicionado utilizado, após ser desligado, tem que esperar 3 minutos para ser ligado novamente segundo orientações fabricante [14]. Com um valor de histerese bem dimensionado pode-se garantir esse tempo de segurança para um ambiente específico. Contudo, o valor de 2 °C foi escolhido para histerese, não pela restrição do compressor, mas pela resolução do sensor. Se a histerese fosse de 1°, quando o sensor estivesse no limite entre os dois valores, esse realizaria medidas oscilando entre os dois valores e provocaria o acionamento indesejado do compressor.

As placas foram posicionadas para os testes de forma que o ar frio do ar condicionado não incidisse diretamente no sensor. Evitou-se assim a realização de medidas que não representassem a temperatura média da sala.

Caso o sensor fosse posicionado na direção do ar frio o ar condicionado operaria por períodos muito curtos e a temperatura média da sala permaneceria alta. Quanto maior o numero de sensores colocados na sala mais precisa seria a média da temperatura da sala. No entanto, é desnecessário em um mesmo ambiente realizar varias medidas de temperatura, uma vez que, não é possível atuar sobre vários pontos simultaneamente. A melhor conduta é posicionar o sensor adequadamente evitando a influência do vento frio do ar condicionado e, eventualmente, outras fontes de calor.

Devido à diferença de densidade entre o ar frio e o ar quente há diferença de temperatura entre o ar próximo ao piso e o ar próximo ao teto. Como os aparelhos de ar condicionado estão posicionados próximos ao teto, o módulo atuador foi posicionado acima do aparelho. Assim, o atuador mede a temperatura do ar de retorno, ou seja, a temperatura do ar que retorna ao ar condicionado para ser refrigerado e serve de parâmetro para o controle do fabricante acionar o equipamento.

Foi incluída nos gráficos a medida de um sensor mais preciso para analisar de forma mais minuciosa a oscilação da temperatura entre os níveis escolhidos para ligar e desligar os equipamentos. Esta medida esta em amarelo nas figuras que mostram as temperaturas na sala. Este sensor está localizado próximo ao teto no limite do ponto médio entre os equipamentos de ar condicionado. Em azul escuro tem-se a temperatura do atuador, em azul claro a temperatura do sensor e em vermelho a temperatura externa.

Os testes realizados têm o objetivo de observar a operação do sistema desenvolvido e compará-lo com o sistema original de fábrica. Pretende-se também verificar se o sistema possibilita economia de energia mantendo o conforto térmico.

6.2 CONTROLE PROPOSTO

O teste iniciado às 14 horas 28 minutos do dia 18 de junho de 2009 e finalizado às 14 horas 25 minutos do dia seguinte está ilustrado na Figura 19 e na Figura 20 A temperatura externa média durante as 24h do teste foi de 22,86 °C com máxima de 29,4 e mínima de 16,8. O desvio padrão foi de 3,875. Este teste foi realizado com acionamento feito pelo software supervisório deste trabalho com *set point* de 22 °C e histerese de 2°C.

Observando os dados da Figura 19 nota-se uma diferença relativamente grande entre a temperatura no atuador e a temperatura no sensor. Essa diferença não apresentou tendência de queda durante o teste. O equipamento da área 1 passou a noite e madrugada desligado exceto por um acionamento por volta das 22 horas. Isso porque a temperatura interna da sala estava oscilando e o sensor fez uma medida de 23 °C que é suficiente para acionar o equipamento.

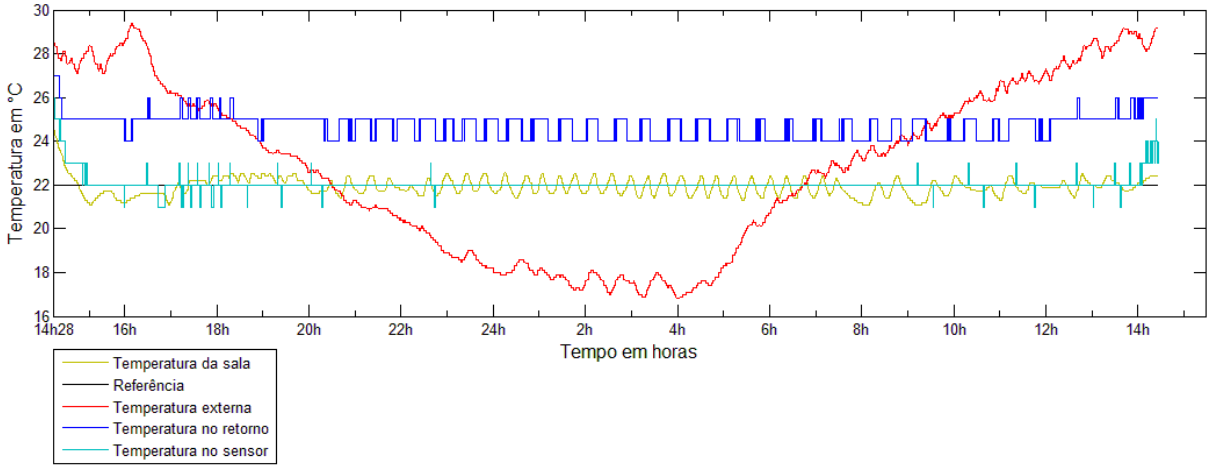

Figura 19- teste com o controle proposto área 1.

Na área 2 o sistema foi acionado durante toda noite e madrugada em intervalos aproximadamente regulares. Esse fato parece não fazer sentido, uma vez que a temperatura externa estava muito abaixo do *set point*. No entanto, a temperatura externa não é a única fonte de calor. Como explicado anteriormente há uma abertura para o laboratório LARA na área 2. A sala do LARA é a principal fonte de calor para a área 2 e essa área acumulou, durante o dia, calor suficiente para provocar o acionamento o ar condicionado durante a noite.

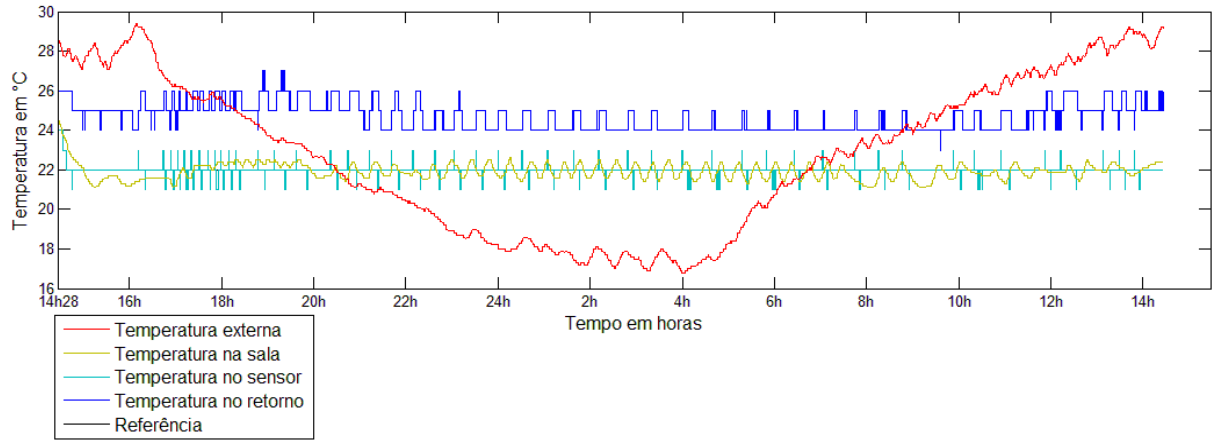

Figura 20- teste com o controle proposto área 2.

A Figura 21 ilustra melhor o fato explicado. O ar condicionado 2 foi acionado durante todo o teste já o ar condicionado 1 não sofreu tanta interferência da abertura do LARA. Isso por que o ar condicionado 2 manteve a temperatura entre os valores estipulados.

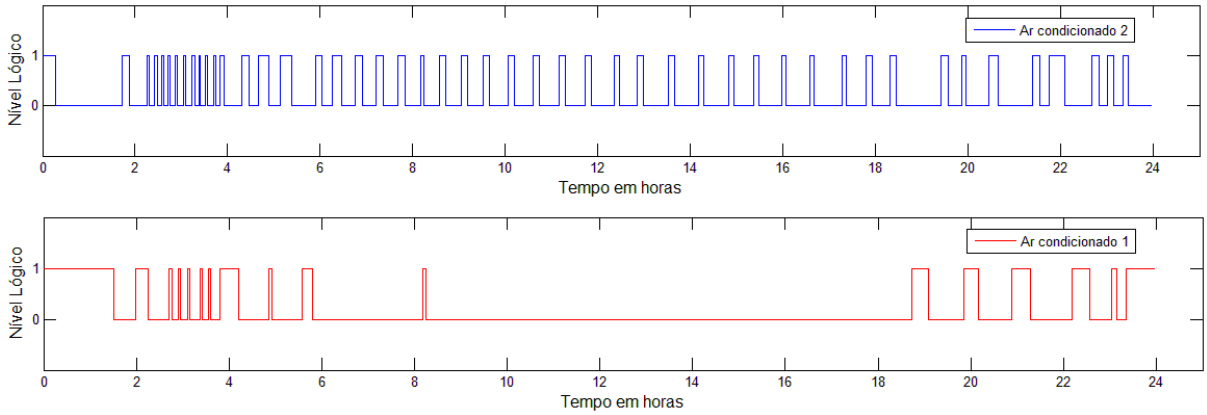

Figura 21- acionamento dos aparelhos durante o teste com o controle proposto.

6.3 CONFORTO TÉRMICO

Com o controle do sistema proposto, dados os parâmetros fixados conforme a Figura 15, a temperatura oscilando em torno de 22° manteve o valor do PMV oscilando em torno de zero como ilustrado na Figura 22. Na escala do PMV o ambiente estava neutro.

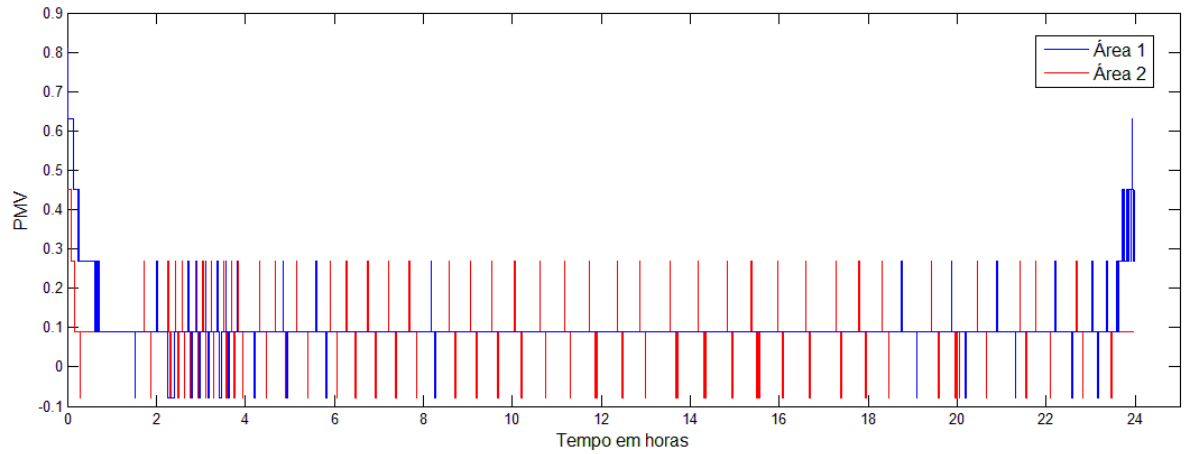

Figura 22- índice PMV calculado para o teste com o controle proposto.

Na Figura 23 têm-se os valores calculados do índice PPD durante o teste. Temos um percentual de pessoas descontentes com o ambiente oscilando entre 5% e 7% segundo este índice. Para a área 2 este valor oscilou mais do que para a área 1. Isso por que a variação de temperatura foi maior na área 2.

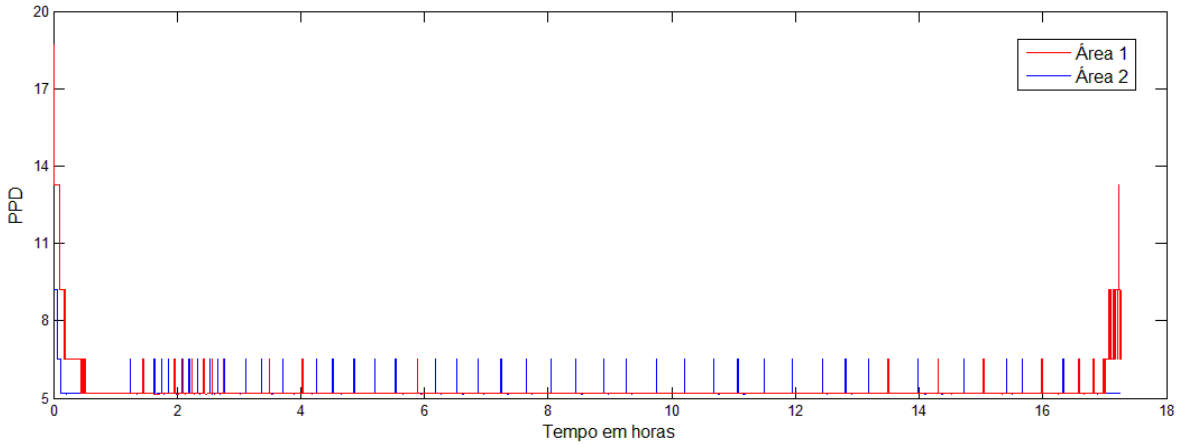

Figura 23- índice PPD calculado para o teste com o controle proposto.

6.4 CONTROLE DO FABRICANTE

O teste iniciado às 17 horas 41 minutos do dia 15 de junho de 2009 e finalizado às 16 horas 31 minutos do dia seguinte está ilustrado na Figura 24 e na Figura 25. A temperatura externa média durante as 23h do teste foi de 23,04 °C com máxima de 29,6 e mínima de 16,9 e desvio padrão de 3,736. Este teste foi realizado com acionamento do fabricante e *set point* de 22 °C. Apesar de o início do teste ser um pouco diferente e a duração ser 1 hora menor, as condições dos testes são semelhantes, e como a duração foi relativamente longa, essas diferenças não produzem efeitos significativos do ponto de vista qualitativo.

O teste com atuação do fabricante mostrou um funcionamento insatisfatório. A sala apresentou temperatura muito baixa durante os testes ficando entre 20 e 21 °C. A razão é bastante simples: o sistema do fabricante é projetado para operar em ambientes fechados com condições mínimas de isolamento térmico. Como nosso ambiente não satisfaz essas condições, a temperatura de retorno do ar não cai na proporção que deveria, enquanto que a temperatura na altura dos sensores chega a valores muito baixos. O erro médio fica entre 1 e 2 graus. Esse erro justifica um gasto maior de energia para manter a sala a essa temperatura. Não se sabe exatamente como funciona o sistema do fabricante, mas, analisando o consumo de energia, pode-se concluir que o equipamento da área 2 passou 2,66 vezes mais tempo ligado em comparação ao teste com o sistema proposto.

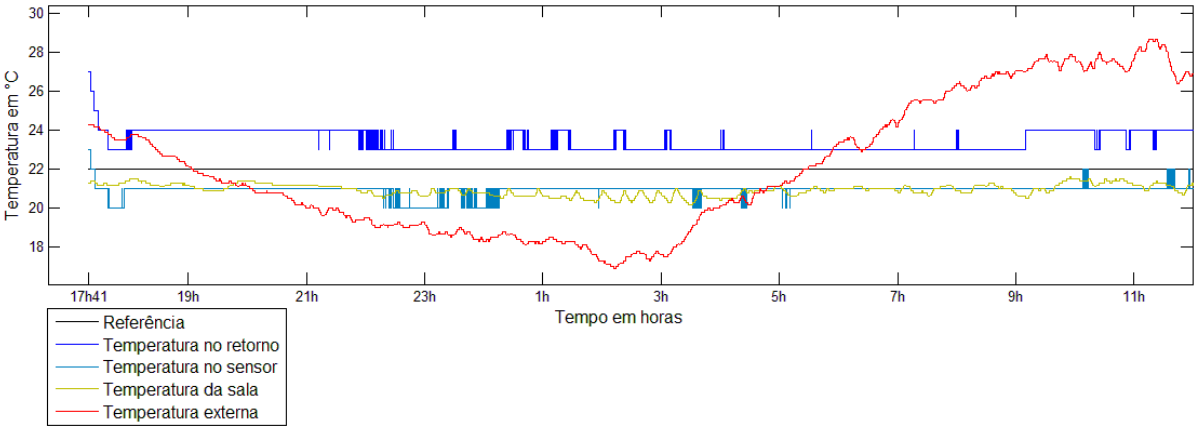

Figura 24- teste com controle do fabricante área 1.

A Figura 25 mostra o resultado da área 1. Nota-se que a temperatura de retorno ficou boa parte do teste a 23°C e chegou a 22°C em alguns momentos. O ar condicionado da área 2 ficou muito tempo ligado possibilitando que o aparelho da área 1 trabalhasse menos. A temperatura nesta área também ficou abaixo do *set point.*

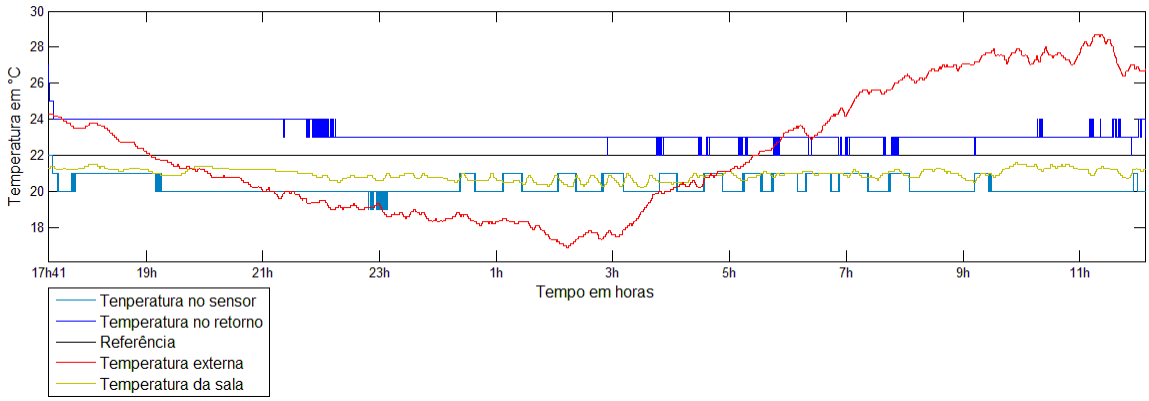

Figura 25- teste com controle do fabricante área 2.

6.5 CONSUMO DE ENERGIA

Como explicitado na Tabela 6 os consumos dos aparelhos foi bastante diferente nos dois testes realizados. As pequenas diferenças entre as condições dos testes não podem ser a causa de uma diferença desta magnitude. A grande diferença de consumo deve-se, sobretudo, ao *layout* da sala que favorece o sistema proposto devido à passagem existente para o laboratório LARA.

O consumo da área 1 foi menor no teste com o sistema do fabricante porque o ar condicionado da área 2 funcionou por um período mais longo interferindo na área1.

Tabela 6- consumo de energia durante os testes.

	Área 1	Área 2	Total	Unidade
Controle do fabricante	8,42	38,41	46,83	KWh
Controle proposto	12,35	14,43	26,78	KWh

Observando o percentual de energia economizada tem-se uma redução de 42,81%, contudo esse resultado deve ser visto com ressalvas. As condições do ambiente não favoreceram o sistema do fabricante. Isso porque o sensor do fabricante está na unidade condensadora [14], conseqüentemente, próximo ao teto. Como o LAVSI apresenta uma passagem para o laboratório LARA cuja área é duas vezes maior do que a área do LAVSI, os equipamentos de ar condicionado têm maior dificuldade em resfriar o ar próximo ao sensor do aparelho. Assim a temperatura da sala fica muito baixa, mas a temperatura próxima ao sensor do aparelho permanece acima do valor de *set point.*

Em condições adequadas o sistema do fabricante pode ser tão eficiente quanto o proposto. Isso não significa que a automação do sistema usando rede de sensores sem fio não possa reduzir gastos. A redução ocorre com a operação mais precisa do sistema mantendo a temperatura de conforto térmico durante o período de utilização do ambiente. Com uma rede de sensores, medidas de temperatura e outras variáveis relevantes ao conforto térmico pode-se reduzir o consumo de energia mantendo o ambiente confortável e evitando que a operação do usuário use mais do que o necessário para manter o conforto.

Em redes de sensores sem fio o consumo de bateria é um fator crítico. Por isso os módulos sensores deste trabalho entram em modo *sleep.* Apesar desta estratégia o consumo de bateria foi relativamente elevado. Este fator pode ser melhorado aumentando o intervalo entre o envio de dados, uma vez que, os dados de temperatura enviados pelos módulos foram muito redundantes.

7 CONCLUSÃO

Apresenta a conclusão e as perspectivas futuras para os próximos trabalhos.

7.1 CONCLUSÃO

Em edifícios já ocupados e sem instalações de sistemas de ar condicionado central a alternativa para refrigeração de ambientes é a utilização de equipamentos de ar condicionado de janela e split. Contudo, esses equipamentos são operados por usuários que muitas vezes não fazem o uso eficiente destes aparelhos. A automação de aparelhos de ar condicionado do tipo janela e do tipo split pode ser implementada por uma rede de sensores e atuadores sem fio. O controle desses equipamentos pode economizar energia mantendo o conforto térmico para os ocupantes das instalações.

A rede de sensores utilizada neste trabalho apresentou-se flexível e eficaz para automação predial. Funcionalidades podem ser acrescentadas pelo simples carregamento de uma nova aplicação no ZigBit. As possibilidades são amplas, entre elas, podem-se conectar sensores ao barramento I2P, sensores ao conversor AD, display as GPIOs e outros equipamentos podem ser controlados. Além disso, a topologia de rede pode ser ampliada acrescentando-se roteadores para aumentar a área controlada.

O *bootloader* fornecido pela MeshNetics permite fácil carregamento de aplicações nos módulos, no entanto, não possibilita ainda o carregamento sem fio. Dessa forma para atualização da aplicação dos módulos é necessário o uso de um computador conectado ao hardware via USB ou por JTAG.

As aplicações desenvolvidas utilizando o BitCloud (API e bibliotecas) necessitam seguir uma série de regras apresentadas neste trabalho. Isso porque a aplicação divide recursos do hardware com camadas da pilha responsáveis pelo padrão de comunicação IEEE 804.15.4 e ZigBee. Considerando a complexidade da aplicação que implementa o protocolo BACNet será necessário um grande desenvolvimento de software para utilizar este protocolo de comunicação.

A baixa resolução da medida de temperatura no ambiente controlado, cerca de 1 °C, resultou em uma resolução do PMV de 0,18 na faixa de valores mostrados na Figura 22. Esse resultado é insatisfatório, pois permite somente 5 valores na faixa de conforto térmico. Por isso para efetuar o controle do PMV deve-se utilizar uma resolução maior.

7.2 PERSPECTIVAS FUTURAS

Serão medidas outras variáveis necessárias para o cálculo do conforto térmico, são elas: velocidade do vento, umidade, radiação térmica, radiação solar e nível de atividade. Com esses dados será possível determinar a temperatura ótima para os ambientes controlados podendo assim economizar energia. Estas medidas permitirão também a atuação sobre outras variáveis fundamentais ao conforto térmico como umidade e velocidade do vento.

Outro aprimoramento desejado é a utilização do protocolo BACNet que deverá ser implementado na plataforma ZigBit. Tornar os dispositivos da rede de sensores e atuadores dispositivos BACNet possibilitará fácil integração entre os equipamentos e softwares de outros fabricantes.

Devido à limitação de alcance dos módulos ZigBee deve-se aprimorar a rede de sensores acrescentando-se módulos roteadores. A função destes módulos é repassar as mensagens recebidas para o próximo dispositivo efetuando um enlace de comunicação entre dispositivos fora do alcance do coordenador.

Nos testes realizados, observou-se a necessidade da padronização dos testes de consumo de energia no laboratório LAVSI. Portanto, sugere-se a padronização das características mais relevantes para possibilitar a comparação dos sistemas.

Outras técnicas de controle podem ser utilizadas, em especial o controle por lógica Fuzzi, que baseado em regras, um controlador Fuzzi pode considerar fatores desconsiderados pelas técnicas convencionais. Dessa forma regras operacionais que poupam energia podem ser implementadas, como por exemplo, desligar o equipamento se a sala estiver vazia.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SOHRABY, K., MINOLI, D.,ZNATI, T. *Wireless Sensor Networks.* Hoboken, John Wiley & Sons, 2007.

[2] FILHO, P.R.M. & DIAS, Y.F.G., (2008). *Acionamento de potência para rede de automação wireless*. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 012, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 67 pp.

[3] MARKOV, D. *Standards in Thermal Comfort.* In: ANNUAL INTERNATIONAL COURSE: VENTILATION AND INDOOR CLIMATE, Sofia, 2002. P. Stankov, 2002 (Ed) pp. 147-157.

[4] LAMBERTS, R. e XAVIER, A.A.P. *Conforto Térmico e Stress Térmico.* Laboratório de Eficiência Energética em Edificações, Universidade Federal de Santa Catarina, Florianópolis, SC, 2002. 111 pp.

[5] ATALAIA, M.A.R. *O conforto humano e as alterações ambientais, um estudo de caso em ambientes quentes*. Universidade de Aveiro, Portugal 10pp.

[6] ÁGUAS, M.P.N. *Conforto térmico*: módulo da disciplina de mestrado "Métodos instrumentais em energia e ambiente". Instituto Superior Técnico. Lisboa, 2000/2001 25 pp.

[7] MESHNETICS, *ZigBit™ Development Kit 2.0 User's Guide*, manual fornecido em CD pelo fabricante, 2008.

[8] MESHNETICS, *ZigBit™ OEM Modules ZDM-A1281-**, manual fornecido em CD pelo fabricante, 2008.

[9] Newman, H. M., *ASHRAE SSPC 135. BACnet*, disponível em < http://www.bacnet.org/Tutorial/HMN-Overview/sld001.htm. > acesso em: 06 de junho de 2009.

[10] FARAHANI, S., *Zigbee Wireless Networks and Tranceiver*. Burlington, MA, : Elsevier, 2008. [11]. ATMEL CORPORATION, *BitCloud User Guide* 2009, disponível em < http://www.atmel.com/dyn/resources/prod_documents/doc8199.pdf.> , acesso 27 de maio de 2009.

[12] ATMEL Corporation. *BitCloud Stack Documentation*, disponível em < http://www.atmel.com/forms/bitcloud_rzraven.asp?fn=dl_BitCloud_ZDK_1_5_0.zip.> acesso em 27 de maio de 2009.

[13] LAPLANTE, P., *Real-Time Systems Design and Analysis*. New York , IEEE, 1992.

[14] SPRINGER CARRIER, *Manual do Proprietário*, disponível em < http://www.springer.com.br/springer/site/biblioteca/biblioteca_interna.asp?Produto_id=676> acesso em 05 de julho de 2009.

[15] GALLO, E. A., RIBEIRO, F. N., (2007). *Índice de conforto térmico ISO 7730 em Automação Predial*. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-n°, Faculdade de Tecnologia, Universidade de Brasília, DF, 98pp.

ANEXO I

Código do aplicativo utilizado nos módulos MeshBean da MeshNetics, baseado no sample aplication lowpower.

AI.1 lowpower.c

```
/**************************************************************************
 \arquivo   lowpower.c
 \autores    Raphael Carvalho de Almeida Azevedo
             Rodrigo Bertuol de Queiroz
 **************************************************************************/
#include <lowpower.h>
/**************************************************************************
 Variáveis Globais
**************************************************************************/
//Variável para determinar o estado do módulo
AppState_t appState = APP_INITIAL_STATE;
//Variável para determinar o estado do dispositivo
AppDeviceState_t appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
//Parâmetro da rede para o tipo de dispositivo
DeviceType_t appDeviceType;
// Endpoint simple descriptor (ZDO endpoint descriptor)
SimpleDescriptor_t simpleDescriptor = {APP_ENDPOINT, APP_PROFILE_ID, 1, 1, 0, 0 , NULL, 0, NULL};
/**************************************************************************
 Variáveis Locais
**************************************************************************/
//Variável para configuração do timer da rede
static HAL_AppTimer_t networkTimer;
//Parâmetro para propriedades da rede (endpoint e indicação de mensagem)
static APS_RegisterEndpointReq_t apsRegisterEndpointReq;
//Parâmetro para a requisição de criação da rede
static ZDO_StartNetworkReq_t  zdoStartNetworkReq;
/**************************************************************************
 Static functions
**************************************************************************/
#ifdef _BUTTONS_
static void buttonReleased(uint8_t button);
#endif
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *conf);
static void initApp(void);
static void startNetwork(void);
static void startingNetworkTimerFired(void);
static void APS_DataInd(APS_DataInd_t *ind);
/**************************************************************************
 Implementação
**************************************************************************/
/*****************************************************************
 Description: Task handler da aplicação (principal)
 Parameters:  nenhum.
```

```
 Returns:    nenhum.
*****************************************************************/
void APL_TaskHandler()
{
        //Estados comuns a todos os módulos
        switch (appState)
        {
                //Estado inicial após reset
                case APP_INITIAL_STATE:
                {
                        initApp();                      // Função para iniciar os módulos
                        break;
                }
                //Estado para iniciar a rede
                case APP_NETWORK_JOINING_STATE:
                {
                        startNetwork();                 // Função para iniciar a rede
                        break;
                }
                //Estado em que a rede foi criada
                case APP_NETWORK_JOINED_STATE:
                {
                        //Se estiver configurado para coordenador
                        #ifdef _COORDINATOR_
                                if (DEVICE_TYPE_COORDINATOR == appDeviceType)
                                {
                                        //Task handler do coordenador
                                        appCoordinatorTaskHandler();
                                }
                        #endif
                        //Se estiver configurado para end-device
                        #ifdef _ENDDEVICE_
                                if (DEVICE_TYPE_END_DEVICE == appDeviceType)
                                {
                                        //Task handler do end-device
                                        appEndDeviceTaskHandler();
                                }
                        #endif
                        break;
                }
                default:
                break;
        }
}
/*****************************************************************
 Description:  Função para iniciar os módulos e configurar parâmetros  na rede
 Parameters:   nenhum.
 Returns:     nenhum.
*****************************************************************/
static void initApp(void)
{
        //Variável para endereço do módulo na rede
        ShortAddr_t nwkAddr;
        //Variável para habilitar recebimento em estado IDLE
```

```
        bool rxOnWhenIdle;
        //Próximo estado do módulo
        appState = APP_INITIAL_STATE;
        //Leitura dos DIP switches para endereço na rede
        nwkAddr = appReadSliders();
        if (0 == nwkAddr)
        {
                //Se DIP switch igual a zero
                #ifdef _COORDINATOR_
                        appDeviceType = DEVICE_TYPE_COORDINATOR;
                        rxOnWhenIdle = true;
                        //Função para inicializar o coordenador (coordinator.c)
                        appCoordinatorInit();
                #else
                        return; // This device can not be coordinator
                #endif // _COORDINATOR_
        }
        else
        {
                //Se DIP switch diferente de zero
                #ifdef _ENDDEVICE_
                        appDeviceType = DEVICE_TYPE_END_DEVICE;
                        rxOnWhenIdle = false;
                        //Função para inicializar o end-device (enddevice.c)
                        appEndDeviceInit();
                #else
                        return; // This device can not be end device
                #endif // _ENDDEVICE_
        }
        //Parâmetros para configurar a rede
        CS_WriteParameter(CS_NWK_ADDR_ID, &nwkAddr); //endereço
        CS_WriteParameter(CS_DEVICE_TYPE_ID, &appDeviceType); //tipo de dispositivo
        //recebimento em estado IDLE
        CS_WriteParameter(CS_RX_ON_WHEN_IDLE_ID, &rxOnWhenIdle);
        //Função para liberar o funcionamento dos LEDs
        appOpenLeds();
        //Função para aguardar a liberção do botão para atualizar o estado
        appOpenButtons(NULL, buttonReleased);
        //Atualiza o estado retornando para o Task Handler da aplicação (principal)
        SYS_PostTask(APL_TASK_ID);
}
/************************************************************************
 Description: Função chamada quando um botão do MeshBean é liberado
 Parameters:  Número do botão pressionado
         (KEY1 as BSP_KEY0 - Inicializar a rede,
          KEY2 as BSK_KEY1 - Acordar e enviar dados - end device)
 Returns:    none
************************************************************************/
#ifdef _BUTTONS_
        static void buttonReleased(uint8_t button)
        {
                switch (button)
                {
                        //Se o botão 1 da MeshBean for pressionado e liberado
```

```
                        case BSP_KEY0:
                        {
                                //Se o módulo estiver no estado inicial
                                if (APP_INITIAL_STATE == appState)
                                {
                                        //Próximo estado do módulo
                                        appState = APP_NETWORK_JOINING_STATE;
                                        //Atualiza o estado retornando para o Task Handler da
                                        //aplicação (principal)
                                        SYS_PostTask(APL_TASK_ID);
                                }
                                break;
                        }
                        //Se o dispositivo estiver configurado como end-device
                        //E o botão 2 da MeshBean for pressionado e liberado
                        #ifdef _ENDDEVICE_
                        case BSP_KEY1:
                        {
                                //Se o módulo estiver em modo sleep
                                if (DEVICE_SLEEP_STATE == appDeviceState)
                                {
                                        //Próximo estado do dispositivo
                                        appDeviceState = DEVICE_AWAKENING_STATE;
                                        //Atualiza o estado retornando para o Task Handler da
                                      //aplicação (principal)
                                        SYS_PostTask(APL_TASK_ID);
                                }
                        break;
                        }
                        #endif
                        default:
                        break;
                }
        }
#endif
/*****************************************************************
 Description: Função de confirmação de criação da rede
 Parameters:  confirmInfo - informação de confirmação da camada ZDO
 Returns:    nenhum.
*****************************************************************/
void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confInfo)
{
        //Desabilita o timer iniciado na função startNetwork
        HAL_StopAppTimer(&networkTimer)
        //Se a camada retornar status de sucesso (rede criada)
        if (ZDO_SUCCESS_STATUS == confInfo->status)
        {
                //Próximo estado do módulo
                appState = APP_NETWORK_JOINED_STATE;
                //Próximo estado do dispositivo
                appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
                //Liga o LED correspondente da rede (verde)
                appOnLed(APP_NETWORK_STATUS_LED);
                //Registra propriedades da rede (endpoint e indicação de mensagem)
```

```
                apsRegisterEndpointReq.simpleDescriptor = &simpleDescriptor;
                apsRegisterEndpointReq.APS_DataInd = APS_DataInd;
                //Envia os parâmetros para a camada APS
                APS_RegisterEndpointReq(&apsRegisterEndpointReq);
        }
        //Atualiza o estado retornando para o Task Handler da aplicação (principal)
        SYS_PostTask(APL_TASK_ID);
}
/*************************************************************************
 Description: Requisição de inicialização da rede
 Parameters:  nenhum.
 Returns:    nenhum.
*************************************************************************/
static void startNetwork(void)
{
        //Configuração de um timer para piscar o LED da rede (verde)
        //Parâmetros do timer
        networkTimer.interval = APP_JOINING_INDICATION_PERIOD;
        networkTimer.mode     = TIMER_REPEAT_MODE;
        networkTimer.callback = startingNetworkTimerFired;
        //Solicitação para a camada HAL iniciar o timer
        HAL_StartAppTimer(&networkTimer);
        //Parâmetro para solicitação de criação da rede
        zdoStartNetworkReq.ZDO_StartNetworkConf = ZDO_StartNetworkConf;
        //Solicitação para a camada ZDO para inicializar a rede
        ZDO_StartNetworkReq(&zdoStartNetworkReq);
}

/*************************************************************************
 Description: Função de callback do timer configurado para a rede
 Parameters:  nenhum.
 Returns:    nenhum.
*************************************************************************/
void startingNetworkTimerFired(void)
{
        //Função de inversão do estado do LED
        appToggleLed(APP_NETWORK_STATUS_LED);
}

/*************************************************************************
 Description:  Função que indica que o módulo recebeu mensagem na rede
 Parameters:   ind - Indicação primitiva da camada APS
 Returns:     nenhum.
*************************************************************************/
void APS_DataInd(APS_DataInd_t* ind)
{
        //Se o módulo estiver configurado como coordenador
        #ifdef _COORDINATOR_
                if (DEVICE_TYPE_COORDINATOR == appDeviceType)
                {
                        //Chama a função correspondente do coordenador
                        //para indicação de mensagem na rede
                        appCoordinatorDataInd(ind);
                }
```

```
        #endif
        //Se o módulo estiver configurado como end-device
        #ifdef _ENDDEVICE_
                if (DEVICE_TYPE_END_DEVICE == appDeviceType)
                {
                        //Chama a função correspondente do end-device
                        //para indicação de mensagem na rede
                        appEndDeviceDataInd(ind);
                }
        #endif
}

/*************************************************************
 Description: Função que notifica alteração na rede
 Parameters:  ZDO_MgmtNwkUpdateNotf_t *nwkParams - Notificação
 Returns:    nenhum.
*************************************************************/
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams)
{
        ZDO_StartNetworkConf_t conf;
        if (ZDO_NETWORK_STARTED_STATUS == nwkParams->status)
        {
                conf.status = ZDO_SUCCESS_STATUS;
                ZDO_StartNetworkConf(&conf);
        }
        else if (ZDO_NETWORK_LEFT_STATUS == nwkParams->status)
        {
                appState = APP_NETWORK_JOINING_STATE;
                SYS_PostTask(APL_TASK_ID);
        }
}
// eof lowpower.c
```

AI.2 coordinator.c

```
/*************************************************************************//**
 \arquivo coordinator.c
 \autores
   Raphael Carvalho de Almeida Azevedo
        Rodrigo Bertuol de Queiroz
*************************************************************************/
#ifdef _COORDINATOR_
#include <lowpower.h>
#include <util/delay.h>
#include <halInterrupt.h>
/*************************************************************************
*
 Variáveis Globais

*************************************************************************
*/
/*************************************************************************
*
```

Variáveis Locais

```
******************************************************************************
*/
//Dados temporários recebidos pela rede
static uint8_t tmpDataBuffer[APP_TMP_DATA_BUFFER_SIZE];
static uint8_t tmpDataBufferActualLength = 0;
//Variáveis para parâmetros da usart
static HAL_UsartDescriptor_t appUsartDescriptor;
static bool usartTxBusyFlag = false;
static uint8_t usartTxBuffer[APP_USART_TX_BUFFER_SIZE];
static uint16_t readBytesCount=0;
//Parâmetros e variáveis para mensagem na rede
static APS_DataReq_t apsDataReq;
static AppMessageBuffer_t appMessageBuffer;
static AppDataTransmissionState_t appDataTtransmissionState =
APP_DATA_TRANSMISSION_IDLE_STATE;
//Variável para configuração do timer
static HAL_AppTimer_t sendMessageTimer;
//Variáveis para os dados recebidos pela rede
static uint16_t temp_sensor_1;
static uint16_t temp_sensor_2;
static uint16_t temp_atuador_1;
static uint16_t temp_atuador_2;
static uint16_t bat_sensor_1;
static uint16_t bat_sensor_2;
//Variáveis de controle recebidos do supervisório
static uint8_t atuador_1;
static uint8_t atuador_2;
//Variáveis de endereço na rede (destinatário)
static ShortAddr_t destAdress;
//Variáveis de endereço na rede (remetente)
static ShortAddr_t remetAdress;
//Flag para envio do controle na interrupção
static bool controle = false;
//Variável para status das portas dos atuadores
static uint8_t status_atuador_1;
static uint8_t status_atuador_2;
/******************************************************************************
*
  Local functions
******************************************************************************
*/
static void usartInit(void);
static void sendDataToUsart(uint8_t* data, uint8_t length);
static void usartWriteConf(void);
static void messageInit(void);
static void APS_DataConf(APS_DataConf_t *conf);
static void timerInit(void);
static void sendUsartMessage(void);
static void appReadByteEvent(uint8_t readBytesLen);
static void sendControlMessageAt1(void);
static void sendControlMessageAt2(void);
/*****************************************************************
```

```
 Description:  Rotina de inicialização do coordenador
 Parameters:   nenhum.
 Returns:     nenhum.
*****************************************************************/
void appCoordinatorInit(void)
{
        //Função para inicializar a USART
        usartInit();
        //Função para configurar os parâmetros de mensagem na rede
        messageInit();
}
/*****************************************************************
 Description:  Task handler do coordenador
 Parameters:   nenhum.
 Returns:     nenhum.
*****************************************************************/
void appCoordinatorTaskHandler(void)
{
        switch (appDeviceState)
        {
                //Estado único do coordenador
                case DEVICE_ACTIVE_IDLE_STATE:
                {
                        //Função para iniciar o timer
                        //para envio de dados para a USART
                        timerInit();
                        break;
                }
                default:
                break;
        }
}
/*****************************************************************
 Description:  Timer para envio de dados para a USART
 Parameters:   nenhum.
 Returns:     nenhum.
*****************************************************************/
void timerInit(void)
{
        //Parâmetros para configuração do timer
        sendMessageTimer.interval = 2000;
        sendMessageTimer.mode     = TIMER_REPEAT_MODE;
        sendMessageTimer.callback = sendUsartMessage;
        //Requisição para a camada HAL iniciar o timer
        HAL_StartAppTimer(&sendMessageTimer);
}
/*****************************************************************
 Description:  Função para enviar os dados recebidos para a USART
 Parameters:   nenhum.
 Returns:     nenhum.
*****************************************************************/
void sendUsartMessage(void)
{
        //Disabilita todas as interrupções
```

```
        HAL_DisableInterrupts();
        //Variável para a mensagem da USART
        uint8_t str[100];
        uint8_t length;
        //Confirmando o tipo de dispositivo (coordenador)
        if (DEVICE_TYPE_COORDINATOR == appDeviceType)
        {
                length = sprintf((char *) str, "%d %d %d %d %d %d\r\n",
                                                temp_sensor_1 , temp_sensor_2 , temp_atuador_1 ,
                                                temp_atuador_2, bat_sensor_1 , bat_sensor_2);
                                                //Função para enviar os dados para a USART
                                                sendDataToUsart(str, length);
        }
        //Função para inverter o status do LED da USART (amarelo)
        appToggleLed(APP_RECEIVING_STATUS_LED);
        //Parte para enviar o sinal de controle para os atuadores
        switch (controle)
        {
                case false:
                {
                        //Atualiza a flag
                        controle = true;
                        //Verifica se o status da porta está diferente do
                        //Sinal de controle recebido pela USART
                        if(status_atuador_1 != atuador_1)
                        {
                                //Atribui ao parâmetro de controle da estrutura da mensagem
                                //O valor recebido pela USART
                                appMessageBuffer.message.control = atuador_1;
                                //Função para enviar o sinal de controle para o atuador 1
                                sendControlMessageAt1();
                        }
                        break;
                }
                case true:
                {
                        //Atualiza a flag
                        controle = false;
                        //Verifica se o status da porta está diferente do
                        //Sinal de controle recebido pela USART
                        if(status_atuador_2 != atuador_2)
                        {
                                //Atribui ao parâmetro de controle da estrutura da mensagem
                                //O valor recebido pela USART
                                appMessageBuffer.message.control = atuador_2;
                                //Função para enviar o sinal de controle para o atuador 1
                                sendControlMessageAt2();
                        }
                        break;
                }
        }
        //Habilita todas as interrupções
        HAL_EnableInterrupts();
}
```

```
/*************************************************************************
 Description:  Função que indica que o módulo recebeu mensagem na rede
 Parameters:   ind - Indicação primitiva da camada APS
 Returns:      nenhum.
*************************************************************************/
void appCoordinatorDataInd(APS_DataInd_t* ind)
{
	//Parâmetros da mensagem
	AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;
	//Atribui a variável o endereço do remetente
	remetAdress = ind->srcAddress.shortAddress;
	//Verifica os endereços e atribuia as variáveis
	//Os dados recebidos na rede
	//Sensor 1
	if(ind->srcAddress.shortAddress == 1)
	{
		temp_sensor_1 = appMessage->temperature;
		bat_sensor_1 = appMessage->battery;
	}
	//Sensor 2
	if(ind->srcAddress.shortAddress == 2)
	{
		temp_sensor_2 = appMessage->temperature;
		bat_sensor_2 = appMessage->battery;
	}
	//Atuador 1
	if(ind->srcAddress.shortAddress == 3)
	{
		temp_atuador_1 = appMessage->temperature;
		status_atuador_1 = appMessage->porta;
	}
	//Atuador 2
	if(ind->srcAddress.shortAddress == 4)
	{
		temp_atuador_2 = appMessage->temperature;
		status_atuador_2 = appMessage->porta;
	}
}
/*************************************************************************
 Description: Função para determinar os parâmetros da USART e iniciar
 Parameters:  nenhum.
 Returns:     nenhum.
*************************************************************************/
void usartInit(void)
{
	//Inicia a flag da USART
	usartTxBusyFlag = false;
	//Parâmetros da USART
	appUsartDescriptor.tty          = APP_USART_CHANNEL;
	appUsartDescriptor.mode         = USART_MODE_ASYNC;
	appUsartDescriptor.baudrate     = USART_BAUDRATE_38400;
	appUsartDescriptor.dataLength   = USART_DATA8;
	appUsartDescriptor.parity       = USART_PARITY_NONE;
	appUsartDescriptor.stopbits     = USART_STOPBIT_1;
```

```
        appUsartDescriptor.rxBuffer       = rxBuffer;
        appUsartDescriptor.rxBufferLength  = USART_RX_BUFFER_LENGTH;
        appUsartDescriptor.txBuffer       = NULL;
        appUsartDescriptor.txBufferLength  = 0;
        appUsartDescriptor.rxCallback     = appReadByteEvent;
        appUsartDescriptor.txCallback     = usartWriteConf;
        appUsartDescriptor.flowControl    = USART_FLOW_CONTROL_NONE;
        //Função do BitCloud para iniciar a USART
        HAL_OpenUsart(&appUsartDescriptor);
}
/*****************************************************************
 Description: Parâmetros de envio de mensagem para a rede
 Parameters:  nenhum.
 Returns:    nenhum.
*****************************************************************/
static void messageInit(void)
{
        //Parâmetros dos dados na rede
        apsDataReq.dstAddrMode          = APS_SHORT_ADDRESS;
        apsDataReq.dstAddress.shortAddress = destAdress;
        apsDataReq.dstEndpoint          = APP_ENDPOINT;
        apsDataReq.profileId            = APP_PROFILE_ID;
        apsDataReq.clusterId            = APP_CLUSTER_ID;
        apsDataReq.srcEndpoint          = APP_ENDPOINT;
        apsDataReq.asduLength           = sizeof (AppMessage_t);
        apsDataReq.asdu                 = (uint8_t *) &appMessageBuffer.message;
        apsDataReq.txOptions.acknowledgedTransmission = 1;
        apsDataReq.radius               = 0;
        apsDataReq.APS_DataConf         = APS_DataConf;
}
/*****************************************************************
 Description: Função de Confirmação do envio para a rede pela camada APS
 Parameters:  nenhum.
 Returns:    nenhum.
*****************************************************************/
static void APS_DataConf(APS_DataConf_t *conf)
{
        //Se a mensagem tiver sido enviada com sucesso
        if (APS_SUCCESS_STATUS == conf->status)
        {
                //Liberação do estado da rede
                appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; // Data
transmission entity is idle
        }
        else
        {
                //Liberação do estado da rede
                appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
        }
        //Desliga o LED de envio para a rede (vermelho)
        BSP_OffLed(LED_RED);
}
/*****************************************************************
 Description: Função para enviar o sinal de controle para atuador 1
```

```
 Parameters:  nenhum.
 Returns:    nenhum.
*****************************************************************/
static void sendControlMessageAt1(void)
{
        //Definindo o endereço de destino
        destAdress = 3;
        //Chama a função para iniciar os parâmetros da mensagem
        messageInit();
        //Se a rede estiver desocupada
        if(appDataTtransmissionState == APP_DATA_TRANSMISSION_IDLE_STATE)
        {
                //Atualiza o estado da rede para ocupado
                appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;
                //Requisição para a camada APS enviar a mensagem
                APS_DataReq(&apsDataReq);
        }
}
/*****************************************************************
 Description: Função para enviar o sinal de controle para o atuador 2
 Parameters:  nenhum.
 Returns:    nenhum.
*****************************************************************/
static void sendControlMessageAt2(void)
{
        //Definindo o endereço de destino
        destAdress = 4;
        //Chama a função para iniciar os parâmetros da mensagem
        messageInit();
        //Se a rede estiver desocupada
        if(appDataTtransmissionState == APP_DATA_TRANSMISSION_IDLE_STATE)
        {
                //Atualiza o estado da rede para ocupado
                appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;
                //Requisição para a camada APS enviar a mensagem
                APS_DataReq(&apsDataReq);
        }
}
/*****************************************************************
 Description: Função para enviar os dados para a USART
 Parameters:  data - ponteiro para o endereço da mensagem
         length - comprimento da mensagem
 Returns:    nenhum.
*****************************************************************/
void sendDataToUsart(uint8_t* data, uint8_t length)
{
        if (APP_TMP_DATA_BUFFER_SIZE > tmpDataBufferActualLength + length)
        {
                memcpy(&tmpDataBuffer[tmpDataBufferActualLength], data, length);
                tmpDataBufferActualLength += length;
        }
        if (false == usartTxBusyFlag)
        {
                usartWriteConf();
```

```
        }
}
/*****************************************************************************
 Description: Função para leitura de dados recebidos pela USART
 Parameters: are not used.
 Returns: nothing.
*****************************************************************************/
static void appReadByteEvent(uint8_t readBytesLen)
{
        //Liga o LED de dados na USART (vermelho)
        BSP_OnLed(LED_RED);
        readBytesLen = readBytesLen;
        //Função do BitCloud para leitura do Byte na USART
        HAL_ReadUsart(&appUsartDescriptor,&read_msg,1);
        //Flag criada para atribuição das variáveis de controle
        readBytesCount++;
        //Atribui o primeiro Byte para a variável de controle
        //Do atuador 1
        if(readBytesCount == 1)
        {
                atuador_1 = (uint8_t)atoi(read_msg);
                read_msg[0] = "";
        }
        //Atribui o primeiro Byte para a variável de controle
        //Do atuador 2
        if(readBytesCount == 2)
        {
                atuador_2 = (uint8_t)atoi(read_msg);
                readBytesCount=0;
                read_msg[0] = "";
                BSP_OffLed(LED_RED);
        }
}
/*****************************************************************
 Description: Writing confirmation has been received. New message can be sent.
 Parameters: none.
 Returns: nothing.
*****************************************************************/
void usartWriteConf(void)
{
        int bytesWritten;
        if (0 < tmpDataBufferActualLength)  // data waiting to be written to USART
        {
                memcpy(usartTxBuffer, tmpDataBuffer, tmpDataBufferActualLength);
                bytesWritten = HAL_WriteUsart(&appUsartDescriptor, usartTxBuffer,
                tmpDataBufferActualLength);
                if (0 < bytesWritten)
                {
                        tmpDataBufferActualLength -= bytesWritten;
                        usartTxBusyFlag = true;
                }
        }
        else
        {
```

```
                usartTxBusyFlag = false;
        }
}
#endif // _COORDINATOR_
// eof coordinator.c
```

AI.3 enddevice.c

```
/*************************************************************************//**
  \arquivo enddevice.c
  \autores
    Raphael Carvalho de Almeida Azevedo
        Rodrigo Bertuol de Queiroz
*****************************************************************************/
#ifdef _ENDDEVICE_
#include <lowpower.h>
#include <avr/io.h>
#include <gpio.h>
#include <util/delay.h>
/*****************************************************************************
*
  Variáveis Globais
*****************************************************************************
*/
/*****************************************************************************
*
  Variáveis Locais
*****************************************************************************
*/
//Parâmetros e variáveis para mensagem na rede
static AppDataTransmissionState_t appDataTtransmissionState =
APP_DATA_TRANSMISSION_IDLE_STATE;
static APS_DataReq_t apsDataReq;
static AppMessageBuffer_t appMessageBuffer;
//Parâmetros para requisição para entrar no modo sleep
static ZDO_SleepReq_t zdoSleepReq;
//Parâmetros para requisição para sair no modo sleep
static ZDO_WakeUpReq_t zdoWakeUpReq;
//Variável para nova leitura dos sliders
static uint8_t sliders;
//Variável para determinar se o módulo vai ser atuador
static bool atuador;
//Variável para configuração do timer
static HAL_AppTimer_t sendDataTimer;
//Variável para leitura da porta de atuação
static uint8_t status_porta;
/*****************************************************************************
*
  Static functions
*****************************************************************************
*/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf);
static void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf);
```

```
static void APS_DataConf(APS_DataConf_t *conf);
static void temperaturesSensorHandler(bool result, int16_t temperature);
static void openPeriphery(void);
static void closePeriphery(void);
static void sendMessage(void);
static void prepareToSleep(void);
static void batterySensorHandler(int16_t battery);
static void timerInit(void);
static void sendData(void);
/************************************************************************
 Description:  Rotina de inicialização do enddevice
 Parameters:   nenhum.
 Returns:      nenhum.
************************************************************************/
void appEndDeviceInit(void)
{
	//Setando a porta ADC_INPUT_! como saída
	GPIO_ADC_INPUT_1_make_out();
	//Analisa para ver se é atuador
	sliders = appReadSliders();
	if (sliders > 2)
	{
		atuador = true;
	}
	if((sliders > 0)&(sliders < 3))
	{
		atuador = false;
	}
	//Parâmetros de envio de mensagem pela rede
	apsDataReq.dstAddrMode           = APS_SHORT_ADDRESS;
	apsDataReq.dstAddress.shortAddress = 0;
	apsDataReq.dstEndpoint           = APP_ENDPOINT;
	apsDataReq.profileId             = APP_PROFILE_ID;
	apsDataReq.clusterId             = APP_CLUSTER_ID;
	apsDataReq.srcEndpoint           = APP_ENDPOINT;
	apsDataReq.asduLength            = sizeof (AppMessage_t);
	apsDataReq.asdu                  = (uint8_t *) &appMessageBuffer.message;
	apsDataReq.txOptions.acknowledgedTransmission = 1;
	apsDataReq.radius                = 0;
	apsDataReq.APS_DataConf          = APS_DataConf;
	//Configuração dos sensores para futuras medições
	BSP_OpenTemperatureSensor();
	BSP_OpenBatterySensor();
}
/************************************************************************
 Description:  Task handler do end-device
 Parameters:   nenhum.
 Returns:      nenhum.
************************************************************************/
void appEndDeviceTaskHandler(void)
{
	//Estados do dispositivo end-device
	switch (appDeviceState)
	{
```

```
                //Estado inicial
                case DEVICE_ACTIVE_IDLE_STATE:
                {
                        //Leitura da temperatura
                        #ifdef _TEMPERATURE_SENSOR_
                        BSP_ReadTemperatureData(temperaturesSensorHandler);
                        #else
                        temperaturesSensorHandler(false, 0);
                        #endif
                        //Leitura da bateria
                        BSP_ReadBatteryData(batterySensorHandler);
                        break;
                }
                //Estado de envio de mensagem para a rede
                case DEVICE_MESSAGE_SENDING_STATE:
                        sendMessage();
                        break;
                //Estado de preparação par ao modo sleep (end-device)
                case DEVICE_SLEEP_PREPARE_STATE:
                {
                        if (appReadButtonsState() & BSP_KEY1)
                        SYS_PostTask(APL_TASK_ID);
                        else
                        prepareToSleep();
                        break;
                }
                //Estado do módulo forçado a sair do modo sleep
                case DEVICE_AWAKENING_STATE:
                {
                        zdoWakeUpReq.ZDO_WakeUpConf = ZDO_WakeUpConf;
                        ZDO_WakeUpReq(&zdoWakeUpReq);
                        break;
                }
                default:
                break;
        }
}
/*************************************************************************
 Description:  Função que indica que o módulo recebeu mensagem na rede
 Parameters:   ind - Indicação primitiva da camada APS
 Returns:      nenhum.
*************************************************************************/
void appEndDeviceDataInd(APS_DataInd_t* ind)
{
        //Variável de controle
        uint8_t control;
        AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;
        ind = ind; //Warning prevention
        //Atribuição para a variável de controle
        //Do dado recebido pela rede
        control = appMessage->control;
        //Se a variável de controle for 1
        //Liga o compressor
        if(control == 1)
```

```
        {
                //Liga o LED amarelo
                BSP_OnLed(LED_YELLOW);
                //Seta a porta para atuação
                GPIO_ADC_INPUT_1_set();
        }
        //Se a variável de controle for 0
        //Desliga o compressor
        if(control == 0)
        {
                //Desliga o LED amarelo
                BSP_OffLed(LED_YELLOW);
                //Reseta a porta para atuação
                GPIO_ADC_INPUT_1_clr();
        }
}
/*****************************************************************
 Description:  Função para abrir os periféricos ao sair do modo sleep
 Parameters:   nenhum.
 Returns:     nenhum.
*****************************************************************/
static void openPeriphery(void)
{
        //Abre novamente os LEDs
        appOpenLeds();
        //Abre novamente os sensores
        BSP_OpenBatterySensor();
        #ifdef _TEMPERATURE_SENSOR_
                BSP_OpenTemperatureSensor();
        #endif
}
/*****************************************************************
 Description:  Função para fechar os periféricos ao entrar do modo sleep
 Parameters:   nenhum.
 Returns:     nenhum.
*****************************************************************/
static void closePeriphery(void)
{
        //Desliga todos os LEDs
        appOffLed(APP_NETWORK_STATUS_LED);
        appOffLed(APP_SENDING_STATUS_LED);
        appOffLed(APP_RECEIVING_STATUS_LED);
        //Desabilita os registradores de saída
        appCloseLeds();
        //Desabilita o sensor de temperatura
        BSP_CloseTemperatureSensor();
}
/*****************************************************************
 Description: Função de Confirmação do envio para a rede pela camada APS
 Parameters:  conf - APS Data Confirm primitive
 Returns:    nenhum.
*****************************************************************/
static void APS_DataConf(APS_DataConf_t *conf)
{
```

```
        //Libera o status da rede
        appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
        //Desliga o LED de envio de dados pela rede (vermelho)
        appOffLed(APP_SENDING_STATUS_LED);
        //Se a mensagem tiver sido enviada com sucesso
        if (APS_SUCCESS_STATUS == conf->status)
        {
                //Próximo estado do dispositivo
                appDeviceState = DEVICE_SLEEP_PREPARE_STATE;
        }
        //Se a mensagem não tiver sido enviada
        else
        {
                //Próximo estado do dispositivo
                appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
        }
        //Atualiza o estado retornando para o Task Handler da aplicação (principal)
        SYS_PostTask(APL_TASK_ID);
}
/************************************************************************
 Description:  Função para leitura da temperatura
 Parameters:   result - status da medida (1 - success, 0 - fail)
          temperature - temperatura medida
 Returns:     none
************************************************************************/
static void temperaturesSensorHandler(bool result, int16_t temperature)
{
        //Se a medida tiver sido realizada com sucesso
        //Atribui a temperatura para a variável correspondente da rede
        //Caso contrário, esta variável será enviada como zero
        if (true == result)
                appMessageBuffer.message.temperature = temperature;
        else
                appMessageBuffer.message.temperature = 0;
}
/************************************************************************
 Description:  Função para leitura da bateria
 Parameters:   battery - nível de bateria medida
 Returns:     none
************************************************************************/
static void batterySensorHandler(int16_t battery)
{
        //Atribui o valor medido a variável correspondente da rede
        appMessageBuffer.message.battery = battery;
        //Próximo estado do dispositivo
        appDeviceState = DEVICE_MESSAGE_SENDING_STATE;
        //Atualiza o estado retornando para o Task Handler da aplicação (principal)
        SYS_PostTask(APL_TASK_ID);
}
/************************************************************************
 Description:  Send the application message
 Parameters:   none
 Returns:     none
************************************************************************/
```

```
static void sendMessage(void)
{
	//Se for atuador ler a porta
	if (atuador == true)
	{
		//Leitura do status da porta do módulo atuador
		status_porta = GPIO_ADC_INPUT_1_read();
		//Atribui o valor medido a variável correspondente da rede
		appMessageBuffer.message.porta = status_porta;
	}
	//Se a rede estier liberada enviar a mensagem
	if (APP_DATA_TRANSMISSION_IDLE_STATE == appDataTtransmissionState)
	{
		//Atualiza o status da rede para ocupado
		appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;
		//Requisição de envio da mensagem na rede
		APS_DataReq(&apsDataReq);
		//Liga o LED para sinalizar o envio
		appOnLed(APP_SENDING_STATUS_LED);
	}
}
/************************************************************************
 Description: Função de Confirmação da requisição para modo sleep
 Parameters:  conf - APS sleep Confirm primitive
 Returns:     nenhum.
************************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf)
{
	//Se a camada ZDO permitir a solicitação
	if (ZDO_SUCCESS_STATUS == conf->status)
	{
		//Se for módulo sensor
		if(atuador == false)
		{
			//Invoca a função para fechar os periféricos
			closePeriphery();
		}
		//Próximo estado do dispositivo
		appDeviceState = DEVICE_SLEEP_STATE;
	}
	else
		//Atualiza o estado retornando para o Task Handler da aplicação (principal)
		SYS_PostTask(APL_TASK_ID);

}
/************************************************************************
 Description:  Função que prepara o módulo para entrar no modo sleep
 Parameters:   nenhum.
 Returns:      nenhum.
************************************************************************/
static void prepareToSleep(void)
{
	//Se o módulo estiver funcionando como sensor
	if(atuador == false)
```

```
        {
                //Define a função de confirmação
                zdoSleepReq.ZDO_SleepConf = ZDO_SleepConf;
                //Envia soliscitação para a camada ZDO para entrar
                //Em modo sleeo
                ZDO_SleepReq(&zdoSleepReq);
        }
        //Se o módulo estiver funcionando como atuador
        else
        {
                //Próximo estado do dispositivo
                appDeviceState = DEVICE_SLEEP_STATE;
                //Chama uma função para configurar um timer
                //Para manter sincronismo
                timerInit();
        }
}
/*************************************************************************
 Description:  Função para iniciar o timer para o módulo atuador
 Parameters:   nenhum.
 Returns:     nenhum.
*************************************************************************/
static void timerInit(void)
{
        //Configura o timer para o atuadro sincronizar com o sensor
        sendDataTimer.interval = 10000;
        sendDataTimer.mode     = TIMER_REPEAT_MODE;
        sendDataTimer.callback = sendData;
        HAL_StartAppTimer(&sendDataTimer);
}
/*************************************************************************
 Description:  Função de callback do Timer
 Parameters:   nenhum.
 Returns:     nenhum.
*************************************************************************/
static void sendData(void)
{
        //Desliga o timer
        HAL_StopAppTimer(&sendDataTimer);
        //Próximo estado do dispositivo
        appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
        //Atualiza o estado retornando para o Task Handler da aplicação (principal)
        SYS_PostTask(APL_TASK_ID);
}
/*************************************************************************
 Description: Função para ser executada após sair do modo sleep
 Parameters:  nenhum.
 Returns:    nenhum.
*************************************************************************/
static void wakeUpHandler(void)
{
        //Próximo estado do dispositivo
        appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
        Função para abrir os periféricos
```

```
        openPeriphery();
        //Liga o LED indicativo da rede (verde)
        appOnLed(APP_NETWORK_STATUS_LED);
        //Atualiza o estado retornando para o Task Handler da aplicação (principal)
        SYS_PostTask(APL_TASK_ID);
}
/************************************************************************
 Description: Função de indicação que o módulo saiu do modo sleep
 Parameters: none.
 Returns: nothing.
************************************************************************/
void ZDO_WakeUpInd(void)
{
 if (DEVICE_SLEEP_STATE == appDeviceState)
        wakeUpHandler();
}
/************************************************************************
 Description: Confirmação da camada ZDO após requisição para acordar
 Parameters:  conf - parâmetro de confirmação
 Returns:    nenhum..
************************************************************************/
void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf)
{
        //Se a camada responder com sucesso o pedido
        //Chama a função para acordar
        if (ZDO_SUCCESS_STATUS == conf->status)
                wakeUpHandler();
        //Se a camada responder com falha o pedido
        //Atualiza o estado retornando para o Task Handler da aplicação (principal)
        //Nova tentativa para acordar
        else
                SYS_PostTask(APL_TASK_ID);
}
#endif // _ENDDEVICE_
// eof enddevice.c
```

ANEXO II

Trechos de código do aplicativo supervisório utilizado para comunicar com o módulo MeshBean configurado como dispositivo coordenador na rede.

AII.1 Comunicação do software com a porta COM

Todos os parâmetros para iniciar a comunicação com o módulo coordenador são definidos na seção "configuração da comunicação" no software supervisório, botão indicado na Figura 26.

Figura 26- botão do software supervisório para configurar os parâmetros de comunicação.

O trecho de código responsável pela tentativa de estabelecer a comunicação pode ser observado a seguir:

```
'SE A PORTA ESTIVER ABERTA, FECHA A COMUNICAÇÃO
If COM_port.IsOpen Then
        COM_port.Close()
End If
'PASSANDO OS PARÂMETROS PARA A COMUNICAÇÃO
With COM_port
        'PORTA SELECIONADA
        .PortName = porta_comunicação
        'BAUDRATE
        .BaudRate = Convert.ToInt64(porta_baudrate)
        'BIT DE DADOS
        .DataBits = Convert.ToInt64(porta_bit_dados)
        'BITS DE PARADA
        If porta_bit_parada = "1" Then
                .StopBits = StopBits.One
        End If
        If porta_bit_parada = "1,5" Then
                .StopBits = StopBits.OnePointFive
        End If
        If porta_bit_parada = "2" Then
                .StopBits = StopBits.Two
        End If
        If porta_bit_parada = "Nenhum" Then
                .StopBits = StopBits.None
        End If
        'PARIDADE
        If porta_paridade = "Par" Then
                .Parity = Parity.Even
        End If
        If porta_paridade = "Ímpar" Then
                .Parity = Parity.Odd
        End If
        If porta_paridade = "Nenhum" Then
                .Parity = Parity.None
```

```
        End If
        If porta_paridade = "Marca" Then
                .Parity = Parity.Mark
        End If
        If porta_paridade = "Espaço" Then
                .Parity = Parity.Space
        End If
End With
'TENTANDO ABRIR A COMUNICAÇÃO COM A PORTA ESCOLHIDA
Try
        'ABRINDO A CONEXÃO COM A PORTA
        COM_port.Open()
        'MUDANDO O STATUS DA CONEXÃO E A FONTE
        Label13.Text = "CONECTADO"
        Label13.Font = New Font("Microsoft Sans Serif",
        Label13.Font.Size, FontStyle.Bold)
        Label13.ForeColor = Color.Green
        'MUDANDO A IMAGEM DO STATUS DA CONEXÃO
        PictureBox1.Image = Image.FromFile(conectado)
        PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
Catch ex As Exception
        MessageBox.Show("Não foi possível iniciar a comunicação com a porta: " + porta_comunicação + "!",
        "Problema na comunicação")
        MsgBox(ex.ToString)
End Try
```

Os parâmetros padrões para a utilização dos módulos MeshBean, foram configurados da seguinte forma:

```
Form1.porta_comunicação = ComboBox1.Text
Form1.porta_baudrate = "38400"
Form1.porta_bit_dados = "8"
Form1.porta_bit_parada = "1"
Form1.porta_paridade = "Nenhum"
Form1.porta_controle = "Nenhum"
```

AII.2 Configuração dos parâmetros para o Conforto Térmico

Os parâmetros para o conforto térmico podem ser configurados na seção “parâmetros do conforto” no software supervisório, botão indicado na Figura 27.

Figura 27- botão do software supervisório para configurar os parâmetros de conforto térmico.

AII.2.1 Valores padrão para o Conforto Térmico

Entre todas as variáveis para o conforto térmico, segundo a norma ISO 7730, apenas foi possível realizar o controle da temperatura ambiente, com o liga/desliga dos compressores dos aparelhos de ar-condicionado.

O valor para a variável umidade relativa do ar foi determinado com o auxílio dos dados coletados pelo sistema de ar-condicionado evaporativo (da sala de reuniões do LAVSI). A umidade do ar para os dias e horários dos testes era em torno de 40%.

Para a variável de velocidade do vento, foi utilizado o sensor AVT 641 (Series 641 Air Velocity Transmitter) da fabricante Dwyer, levando em consideração uma vazão constante de ar na saída dos aparelhos controlados. Com o sensor posicionado na direção do vento, foram coletados valores de até 2 m/s para a velocidade do vento.

Na tentativa de evitar nichos de desconforto térmico nos locais ocupados do laboratório, a saída de ar do split foi direcionado para uma direção oposta à localização do módulo sensor, obtendo medidas de até 1m/s nas posições dos módulos sensores. Este valor elevado ocorre devido à utilização do modo "*swing*" do aparelho de ar-condicionado, em que altera a posição horizontal das pás do aparelho na horizontal.

Apesar de a norma ISO 7730 levar em consideração velocidades do ar inferiores a 0,15 m/s no inverno e 0,25 m/s no verão, o valor de 1m/s foi utilizado, com o objetivo de calcular a influência desta variável no conforto térmico do laboratório.

Para a temperatura média radiante, o sensor TY7321 da fabricante YAMATAKE (Figura 28), obtendo valores médios de 27,5 ºC no ambiente com quatro computadores ligados e quatro ocupantes.

Figura 28- sensor TY7321 (YAMATAKE)

Para o parâmetro de metabolismo em função das atividades executadas no ambiente, o valor padrão do software supervisório foi definido para atividades leves. Enquanto que para o parâmetro de vestuário, foi definido como padrão vestimenta de trabalho. Os cálculos referentes a estes parâmetros, em função da Equação 6 e Tabelas 1 e 2, podem ser observados no trecho de código a seguir:

```
'FUNÇÃO QUE CALCULA O FATOR DE VESTUÁRIO
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles ComboBox1.SelectedIndexChanged
        Select Case ComboBox1.Text
                Case "Nu"
                        i_vest = 0
                        i_r_vest = 0
                Case "Calções"
                        i_vest = 0.1
                        i_r_vest = 0.016
                Case "Tropical"
                        i_vest = 0.3
                        i_r_vest = 0.047
                Case "Leve de Verão"
                        i_vest = 0.5
                        i_r_vest = 0.078
                Case "Trabalho"
                        i_vest = 0.7
```

```
                    i_r_vest = 0.124
                Case "Inverno, interior"
                    i_vest = 1.0
                    i_r_vest = 0.155
                Case "Fato Completo"
                    i_vest = 1.5
                    i_r_vest = 0.233
        End Select
        calcula_fator_vestuário()
End Sub
'FUNÇÃO QUE CALCULA O FATOR DE VESTUÁRIO
Private Function calcula_fator_vestuário()
        If (i_r_vest < 0.078) Then
            f_vest = 1.0 + (1.29 * i_r_vest)
        End If
        If (i_r_vest >= 0.078) Then
            f_vest = 1.05 + (0.645 * i_r_vest)
        End If
        Return Nothing
End Function
'FUNÇÃO QUE CALCULA O PARÂMETRO METABOLISMO
Private Sub ComboBox2_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles ComboBox2.SelectedIndexChanged
        Select Case ComboBox2.Text
            Case "Deitado"
                metabolismo = 47
            Case "Sentado a descansar"
                metabolismo = 58
            Case "Atividade sedentária"
                metabolismo = 70
            Case "De pé, atividade leve"
                metabolismo = 93
            Case "De pé, atividade média"
                metabolismo = 117
            Case "Grande atividade"
                metabolismo = 175
        End Select
End Sub
```

AII.2.2 Cálculo da Equação do Conforto

Analisando a Equação (7), é possível notar a recursão da variável T_{vest}, necessitando então a realização da iterações no código para uma aproximação do valor desta variável (15). Para a iteração, a temperatura da vestimenta é iniciada com o valor da temperatura do ar.

O trecho de código responsável por calcular a equação do conforto pode ser analisado a seguir.

```
Dim acumulação_de_calor As Double = 0
Dim convecção As Double = 0
Dim radiação As Double = 0
Dim respiração_sensível As Double = 0
Dim respiração_latente As Double = 0
Dim transpiração As Double = 0
Dim difusão_de_vapor As Double = 0
Dim metabolismo_e_trabalho As Double = 0
```

```
Dim pressão_parcial_vapor As Double = 0
Dim p_sat As Double = 0
Dim t_vest As Double = 0
'parte referente à metabolismo e trabalho da equação
metabolismo_e_trabalho = metabolismo - trabalho
'calculando p_sat para pressão_parcial_vapor
p_sat = 1000 * Math.Pow(Math.E, (16.6536 - (4030.183 / (temp_área + 235))))
'calculando pressão_parcial_vapor
pressão_parcial_vapor = umidade_ar * p_sat
'parte referente à difusão de vapor da equação
difusão_de_vapor = -(3.05 * Math.Pow(10, -3) * (5733 - 6.99 * (metabolismo - trabalho) -
pressão_parcial_vapor))
'parte referente à transpiração
transpiração = -(0.42 * ((metabolismo - trabalho) - 58.15))
'parte referente à respiração_latente
respiração_latente = -(1.7 * Math.Pow(10, -5) * metabolismo * (5867 - pressão_parcial_vapor))
'parte referente à respiração sensível
respiração_sensível = -(0.0014 * metabolismo * (34 - temp_área))
'TENTATIVA DE FAZER A ITERAÇÃO PARA CONVERGIR OS VALORES
Dim t_vest_interação As Double = temp_área
Dim teste As String = "interação s/ interação t_pele convecção radiação" + vbCrLf
Dim teste1 As Double = 0
While (True)
'calculando t_vest
t_vest = calcular_t_vest(t_vest_interação, temp_área)
'calcula convecção
convecção = calcular_convecção(t_vest_interação, temp_área)
'calcula radiação
radiação = calcular_radiação(t_vest_interação)
teste += t_vest.ToString + " " + t_vest_interação.ToString + " " + convecção.ToString + " " + radiação.ToString +
vbCrLf
If t_vest < t_vest_interação Then
t_vest_interação -= 1
While (True)
interações += 1
'calculando t_vest
t_vest = calcular_t_vest(t_vest_interação, temp_área)
'calcula convecção
convecção = calcular_convecção(t_vest_interação, temp_área)
'calcula radiação
radiação = calcular_radiação(t_vest_interação)
teste += t_vest.ToString + " " + t_vest_interação.ToString + " " + convecção.ToString + " " + radiação.ToString +
vbCrLf
If t_vest < t_vest_interação Then
t_vest_interação -= 0.1
While (True)
interações += 1
'calculando t_vest
t_vest = calcular_t_vest(t_vest_interação, temp_área)
'calcula convecção
convecção = calcular_convecção(t_vest_interação, temp_área)
'calcula radiação
radiação = calcular_radiação(t_vest_interação)
teste += t_vest.ToString + " " + t_vest_interação.ToString + " " + convecção.ToString + " " + radiação.ToString +
vbCrLf
If t_vest < t_vest_interação Then
t_vest_interação -= 0.01
While (True)
interações += 1
```

```
'calculando t_vest
t_vest = calcular_t_vest(t_vest_interação, temp_área)
'calcula convecção
convecção = calcular_convecção(t_vest_interação, temp_área)
'calcula radiação
radiação = calcular_radiação(t_vest_interação)
teste += t_vest.ToString + " " + t_vest_interação.ToString + " " + convecção.ToString + " " + radiação.ToString +
vbCrLf
If t_vest < t_vest_interação Then
t_vest_interação -= 0.001
'calculando t_vest
t_vest = calcular_t_vest(t_vest_interação, temp_área)
'calcula convecção
convecção = calcular_convecção(t_vest_interação, temp_área)
'calcula radiação
radiação = calcular_radiação(t_vest_interação)
teste += t_vest.ToString + " " + t_vest_interação.ToString + " " + convecção.ToString + " " + radiação.ToString +
vbCrLf
Exit While
End If
t_vest_interação += 0.001
End While
Exit While
End If
t_vest_interação += 0.01
End While
Exit While
End If
t_vest_interação += 0.1
End While
Exit While
End If
t_vest_interação += 1
End While
'agora calculando a acumulação de calor
acumulação_de_calor = metabolismo_e_trabalho + difusão_de_vapor + transpiração + respiração_latente +
respiração_sensível + radiação + convecção

Private Function calcular_t_vest(ByVal iteração As Double, ByVal temp As Double)
    Dim t_pele As Double = 0
    Dim resultado As Double = 0
    'calculando t_pele para utilizar em t_vest
    t_pele = (35.7 - (0.0275 * (metabolismo - trabalho)))
    'calculando t_vest
    resultado = (t_pele - (i_r_vest * ((f_vest * (12.1 * (Math.Sqrt(velocidade_vento))) * (iteração - temp)) +
(3.96 * Math.Pow(10, -8) * f_vest * ((Math.Pow((iteração + 273), 4)) - (Math.Pow((temp_média_radiante +
273), 4)))))))
    Return resultado
 End Function

Private Function calcular_radiação(ByVal iteração As Double)
    Dim resultado As Double = 0
    'parte referente à radiação da equação do pmv
    resultado = -(3.96 * Math.Pow(10, -8) * f_vest * ((Math.Pow((iteração + 273), 4)) -
(Math.Pow((temp_média_radiante + 273), 4))))
    Return resultado
End Function

Private Function calcular_convecção(ByVal iteração As Double, ByVal temp As Double)
```

```
    Dim resultado As Double = 0
    'calculando parte referente a convecção da equação do pmv
    resultado = -(f_vest * (12.1 * (Math.Sqrt(velocidade_vento))) * (iteração - temp))

    Return resultado
End Function
```

AII.2.3 Cálculo do PMV

O trecho de código responsável pelo cálculo do índice PMV pode ser observado a seguir:

```
'calculando o pmv
pmv = acumulação_de_calor * ((0.303 * Math.Pow(Math.E, -(0.036 * metabolismo))) + 0.028)
```

AII.2.4 Cálculo do PPD

O trecho de código responsável pelo cálculo do índice PPD pode ser observado a seguir:

```
Private Function calcular_PPD(ByVal PMV As Double)
    Dim ppd As Double = 0
    ppd = 100 - (95 * Math.Pow(Math.E, ((-0.03353 * Math.Pow(PMV, 4)) - (0.2179 * Math.Pow(PMV, 2)))))
    Return ppd
End Function
```

AII.2.5 Envio do sinal de controle

O trecho de código responsável por verificar se existe a necessidade de atuação dos compressores dos aparelhos de ar-condicionado, e conseqüente o envio do sinal de controle pode ser observado a seguir:

```
Private Function verificar_atuação()
    Dim atuar As Boolean = False
    If ((temp_sensor_1 >= (set_point + 1)) And (split_1 = "desligado")) Then
      atuar = True
      split_1 = "ligado"
      PictureBox2.Image = Image.FromFile(ar_ligado)
      PictureBox2.SizeMode = PictureBoxSizeMode.StretchImage
      With Form3.dados
        .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
        .SelectionColor = Color.Blue
        .AppendText("Comando ligar split 1" + vbCrLf)
        .ScrollToCaret()
      End With
    End If
    If (temp_sensor_1 <= (set_point - 1) And (split_1 = "ligado")) Then
      atuar = True
      split_1 = "desligado"
      PictureBox2.Image = Image.FromFile(ar_desligado)
      PictureBox2.SizeMode = PictureBoxSizeMode.StretchImage
      With Form3.dados
        .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
        .SelectionColor = Color.Blue
        .AppendText("Comando desligar split 1" + vbCrLf)
```

```
            .ScrollToCaret()
        End With
    End If
    If ((temp_sensor_2 >= (set_point + 1)) = True And (split_2 = "desligado")) Then
        atuar = True
        split_2 = "ligado"
        PictureBox3.Image = Image.FromFile(ar_ligado)
        PictureBox3.SizeMode = PictureBoxSizeMode.StretchImage
        With Form3.dados
            .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
            .SelectionColor = Color.Blue
            .AppendText("Comando ligar split 2" + vbCrLf)
            .ScrollToCaret()
        End With
    End If
    If ((temp_sensor_2 <= (set_point - 1)) = True And (split_2 = "ligado")) Then
        atuar = True
        split_2 = "desligado"
        PictureBox3.Image = Image.FromFile(ar_desligado)
        PictureBox3.SizeMode = PictureBoxSizeMode.StretchImage
        With Form3.dados
            .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
            .SelectionColor = Color.Blue
            .AppendText("Comando desligar split 2" + vbCrLf)
            .ScrollToCaret()
        End With
    End If
    If atuar = True Then
        'MessageBox.Show("entrou")
        atuar_split()
    End If
    Return Nothing
End Function

Private Function atuar_split()
    auto_save_excel()
    If split_1 = "ligado" And split_2 = "ligado" Then
        COM_port.Write("1")
        COM_port.Write("1")
    End If
    If split_1 = "ligado" And split_2 = "desligado" Then
        COM_port.Write("1")
        COM_port.Write("0")
    End If
    If split_1 = "desligado" And split_2 = "ligado" Then
        COM_port.Write("0")
        COM_port.Write("1")
    End If
    If split_1 = "desligado" And split_2 = "desligado" Then
        COM_port.Write("0")
        COM_port.Write("0")
    End If
    Return Nothing
End Function
```