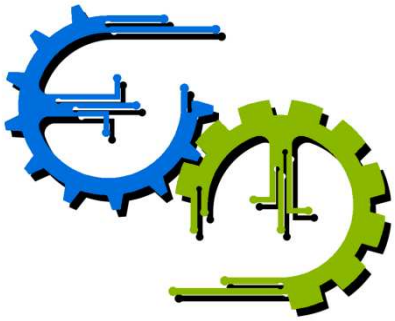
TRABALHO DE GRADUAÇÃO

INSTRUMENTAÇÃO E CONTROLE DE UM SISTEMA DE AR CONDICIONADO HÍBRIDO

Por
Alexandra Gonçalves de Ávila
Breno Holanda Saloio

Brasília, Agosto de 2009



ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

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

TRABALHO DE GRADUAÇÃO

INSTRUMENTAÇÃO E CONTROLE DE UM SISTEMA DE AR CONDICIONADO HÍBRIDO

POR

Alexandra Gonçalves de Ávila
Breno Holanda Saloio

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

Banca Examinadora

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

Prof. Geovany Araújo Borges, UnB/ ENE _____

Prof. Marco A. do Egito Coelho, UnB/ ENE _____

Brasília, Agosto de 2009

FICHA CATALOGRÁFICA

AVILA, ALEXANDRA GONÇALVES &
SALOIO, BRENO HOLANDA
Instrumentação e controle de um sistema de ar condicionado híbrido,

Distrito Federal 2009.

xiii, 107p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2009). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1.Automação 2.Controle
3.Conforto térmico 4.Ar condicionado híbrido

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

REFERÊNCIA BIBLIOGRÁFICA

AVILA, A.G. & SALOIO, B.H., (2009). Instrumentação e controle de um sistema de ar condicionado híbrido. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 29/2009, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 107p.

CESSÃO DE DIREITOS

AUTOR: Alexandra Gonçalves de Ávila e Breno Holanda Saloio.

TÍTULO DO TRABALHO DE GRADUAÇÃO: Instrumentação e controle de um sistema de ar condicionado híbrido.

GRAU: Engenheiro de Controle e Automação ANO: 2009

Alexandra Gonçalves de Ávila
SCLN 105 Bloco D sl 12 – Asa Norte.
70000-000 Brasília – DF – Brasil.

Breno Holanda Saloio
SQN 307 Bloco J ap 502 – Asa Norte.
70746-100 Brasília – DF – Brasil.

RESUMO

O sistema de refrigeração proposto para a planta piloto é um sistema híbrido que integra um sistema de refrigeração convencional, isto é, aparelhos de ar-condicionado comerciais com ciclos de expansão/compressão de gás e um sistema de refrigeração evaporativa direta. O sistema de refrigeração evaporativa proposto utiliza um novo conceito onde uma célula evaporativa de tamanho reduzido e baixa resistência ao fluxo de ar é utilizada com um ventilador, montados em uma caixa com um pequeno reservatório de água e uma pequena bomba elétrica que periodicamente umedece a célula evaporativa. A vantagem deste novo sistema é a possibilidade da utilização do refrigerador evaporativo onde se pode umedecer o ambiente e ao mesmo tempo resfriá-lo, sem a necessidade de utilizar o compressor em determinados horários do dia ou da noite. O sistema de ar condicionado evaporativo é muito mais eficiente que o ar condicionado convencional (gás + compressor), porém, isoladamente não garante conforto térmico. À medida que a umidade do ar aumenta, reduz-se o potencial de resfriamento. O projeto descrito neste trabalho faz parte de um projeto maior, onde o objetivo é o controle do ambiente de um edifício visando o conforto térmico humano e economia de energia para sistemas de refrigeração. Neste projeto a proposta é automatizar um sistema de ar condicionado híbrido já construído e instalado na sala de reuniões do LAVSI (Laboratório de Automação, Visão e sistemas Inteligentes). O projeto envolve a instalação de uma rede sem fio de sensores e atuadores na sala de reuniões que deverá ser integrada a um software supervisório para observação, gravação e controle das variáveis do ambiente. Os sensores e atuadores foram divididos em módulos que representam uma área relativa à sala ou a uma função exercida pelo módulo. Cada módulo contém um módulo ZigBit de radio freqüência sob o protocolo de comunicação ZigBee, ou seja, o padrão IEEE 802.15.4. Estes dispositivos são responsáveis pela comunicação sem fio da rede. A programação dos módulos deve ser feita de maneira a garantir o conforto térmico, minimizar o consumo de energia e evitar que a condensação danifique os aparelhos eletrônicos. A atuação no sistema será sobre o acionamento do: 1) compressor, 2) damper, 3) ventilador e 4) umidificador. E deverá medir a temperatura, a umidade, a velocidade do vento, a radiação térmica e a radiação solar.

Palavras Chave: Ar condicionado evaporativo, sistema híbrido, rede *wireless*, conforto térmico.

ABSTRACT

The proposed refrigeration system for the pilot plant is an hybrid system which integrates a conventional refrigeration system, i.e., commercial air-conditioning equipment with expansion/compressing cycle of refrigerant gas and a direct evaporative refrigeration system. The evaporative refrigerator system proposed utilizes a new concept where an evaporative cell of reduced size and low resistance to the air flux is used with a ventilator, arranged in a box with a small reservoir of water and a small electric pump that periodically humidify the evaporative cell. The advantage of this new system is the possibility of utilize an evaporative refrigerator system to humidify the environment at the same time it is refrigerated, without the necessity of use the compressor in certain hours of day or night. The evaporative air-conditioning is much more efficient than the convectional air-conditioning (gas + compressor), however, working alone it is not capable to offer thermal comfort. As the humidity increases, it reduces the potential of cooling. The project presented in this work is part of a larger project, where the objective is the control of a building's ambient, pretending to reach the human thermal comfort and economy of energy for refrigeration systems. In this project the proposal is to automate a hybrid air-conditioning system already built and installed in the meeting room of LAVSI (Laboratório de Automação, Visão e sistemas Inteligentes).The project involves the installation of a wireless network of sensors and actuators in the meeting room that must be integrated with a supervisory software for observation, record and control of the variables of the environment. The sensors and the actuators were divided into modules that represent the area relative to the room or by a function exercised by the module. Each module has a ZigBit module of radio frequency over the ZigBee protocol, i.e. the IEEE 802.15.4 pattern. The programming of the modules must be in a way to guarantee the thermal comfort, reduce the energy consumption and to avoid condensation capable to damage electronics equipments. The system actuation will act on the: 1) compressor, 2) damper, 3) ventilator and 4) humidification. It will measure the temperature, the humidity, the air velocity, the thermal radiation and solar radiation.

Keywords: Evaporative air-conditioning; hybrid system; ambient comfort; wireless network.

SUMÁRIO

1 Introdução 1

1.1 Objetivo do projeto 2

2 Fundamentos Teóricos 3

2.1 Índice de Conforto Térmico - PMV. 3

2.2 ZigBit – MeshNetics/ATMEL. 7

2.2.1 Padrão ZigBee – IEEE 802.15.4 8

2.2.2 BitCloud - ZigBeeNet 9

3 Desenvolvimento 12

3.1 Possibilidades de implementação utilizando projetos concluídos 13

3.1.1 Opção A – Sitrad (Banco de dados) 13

3.1.2 Opção B – Sitrad (RS-485) 13

3.1.3 Opção C – Banco de Dados (Replicação UDP) 14

3.1.4 Opção D – ZigBit e novos sensores 14

3.2 Implementação Final 15

3.3 Sensores e Atuadores do sistema 17

3.3.1 Anemômetro Dwyer 641-12-LED 17

3.3.2 Sensor de Umidade e Temperatura SHT71 19

3.3.3 Sensor de Radiação Térmica Média TY321A1009 21

3.3.4 Piranômetro Silicon Pyranometer SLIB – M003 22

3.3.5 Relé de Estado Sólido T2405Z - M 24

3.4 Módulos sensores/atuador de medição de dados wireless 25

3.4.1 Módulo Coordenador 25

3.4.2 Módulo Atuador 27

3.4.3 Módulo Interno 29

3.4.4 Módulo Móvel 30

3.4.5 Módulo Externo 32

3.5 Estratégia de controle 33

3.5.1 Histerese 33

3.5.2 Controle implementado 34

3.6 Software Supervisório 34

4 Dados Experimentais e Análise 40

5 Conclusão e Perspecitivas Futuras 46

Referências Bibliograficas 47

Anexos 49

AI Cáculo do PMV 50

AII Especificações do Projeto do Ar Condicionado Híbrido.....54

AIII Código do Aplicativo ZigBit/Coordenador e ZigBit/EndDevices 58

LISTA DE FIGURAS

2.1 Manutenção da temperatura interna do corpo. 3
2.2 Dissipação de calor do corpo humano....5
2.3 Índice PMV relacionado ao nível de conforto térmico 6
2.4 Percentagem previsível de pessoas insatisfeitas (PPD) em função do voto médio previsível (PMV) 7
2.5 Esquemático do módulo ZigBit 8
2.6 Arquitetura da pilha do softwareBitCloud....10
3.1 Ar Condicionado Híbrido do LAVSI 12
3.2 Localização dos módulos no LAVSI 16
3.3 Anemômetro Dwyer....17
3.4 Ligação Elétrica do Anemômetro....18
3.5 SHT-71....19
3.6 Condição de Operação e Memória....19
3.7 Esquema Elétrico do SHT-71 20
3.8 Exemplo de seqüência de transmissão e coleta dos dados....20
3.9 Sensor de Radiação Térmica TY7321A1009....21
3.10 Ligação elétrica do Sensor de Radiação Térmica TY321A1009 22
3.11 Piranômetro Silicon Pyranometer SLIB-M003 23
3.12 Relé de Estado Sólido - Teletronic 24
3.13 Esquema Elétrico – SSR Teletronic 25
3.14 Módulo Coordenador....26
3.15 Esquema Elétrico do Módulo Coordenador 27
3.16 Módulo Atuador....27
3.17 Acionamento do Damper....28
3.18 Esquema Elétrico do Módulo Atuador 29
3.19 Módulo Interno 29
3.20 Esquema Elétrico do Módulo Interno....30
3.21 Módulo Móvel....31
3.22 Esquema Elétrico do Módulo Móvel 31
3.23 Módulo Externo 32
3.24 Esquema Elétrico do Módulo Externo 33
3.25 Resposta pro Histerese....34
3.27 Tela principal do Software Supervisório 35
3.28 Botões do Supervisório 36
3.29 Configuração da Comunicação 36
3.30 Parâmetros o Conforto Térmico 37

3.31 Comandos e Dados.....38
4.1 Gráfico dos dados de temperatura coletados, sem filtragem.....41
4.2 Gráfico referente às temperaturas mensuradas.....42
4.3 Gráfico referente às umidades mensuradas.....42
4.4 Gráfico dos dados referentes à Temperatura Radiante.....43
4.5 Gráfico dos dados referentes à Radiação Solar.....43
4.6 Gráfico dos dados referentes à Velocidade do Vento.....44

LISTA DE TABELAS

2.1 Variáveis do PMV 4
3.1 Especificações do Anemômetro 18
3.2 Especificações do SHT-71 21
3.3 Especificações do Sensor de Radiação Térmica TY7321A1009 22
3.4 Especificações do Piranômetro Pyranometer SLIB-M003 24
3.5 Modos de operação do Sistema Híbrido 39

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	

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
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
LAVSI	Laboratório de Visão e Sistemas Inteligentes
mA	Mili Amperes
MCU	Microcontroller Unit
MHz	Mega Hertz
mim	Minuto
MS-DOS	Microsoft Disk Operation System

PMV	Predicted Mean Vote
PPD	Predicted Percentage of Dissatisfied
RF	Radio Freqüência
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
V	Volts
VAC	Corrente Alternada
VDC	Corrente Continua
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Network
Zigbee	Padrão IEEE 802.15.4 de comunicação sem fio

1 Introdução

Na atualidade, o assunto desenvolvimento sustentável está em foco, pois a visão das pessoas em todo o mundo sobre como é “preciso garantir as necessidades do presente sem comprometer o futuro” vem mudando. O desenvolvimento de ambientes inteligentes proporcionaria, além de uma economia energética, uma mínima degradação ambiental, conciliando o desenvolvimento econômico com a preservação ambiental.

Baseada nessa percepção sobre a evolução humana, a tecnologia vêm expandindo de forma a garantir conforto e máxima economia, diminuindo o abismo que há entre o consumo e a renovação. Charles Darwin já citava “Não é o mais forte nem o mais inteligente que sobrevive. É o mais adaptado à mudanças”, mas com a voracidade em que o homem devora o meio ambiente, chegará o dia em que a Terra, não mais se adaptará ao homem e este perecerá.

Do mesmo jeito em que a tecnologia foi vilã no século passado, tem-se agora a oportunidade de se transformar a tecnologia na solução para preservação e sustentabilidade do futuro.

A necessidade de se controlar sistemas e processos físicos existe desde tempos remotos. O controle manual, primeira forma de controle utilizada pelo homem, e ainda presente em muitos processos, apresenta a necessidade de um operador humano que deve conhecer o sistema e ter razoável experiência e habilidade. Com o crescente aumento no grau de sofisticação das atividades humanas surgiu o interesse e a necessidade de automatizar ou semi-automatizar determinados processos, isso foi possível a partir do desenvolvimento cientifico e tecnológico, que dentre os diversos conhecimentos nos trouxe as teorias clássicas de controle. Contudo, com o avanço da tecnologia, os sistemas e processos ficaram ainda mais complexos, tornando ineficaz ou até mesmo impossível a utilização dos controladores convencionais obtidos a partir da teoria clássica. Isso desencadeou uma busca por novos métodos e estratégias de controle tais como: controle adaptativo, controle preditivo, e sistemas de controle inteligente.

A utilização de sistemas inteligentes em controle tem despertado grande interesse nos últimos anos. Dentre as técnicas mais utilizadas estão a Lógica Nebulosa (“fuzzy”) e as Redes Neurais Artificiais (RNA).

Neste projeto, abordar-se-á a lógica de controle clássica, controle liga-desliga com uso de histerese, porém este projeto possui uma estrutura que suporta outras implementações de controle.

1.1 Objetivo do projeto

O objetivo do projeto é a instrumentação, controle e automação de um ambiente, em que este é monitorado por uma rede de módulos de sensores e atuadores. Os módulos se comunicam através da tecnologia *wireless*, ZigBee, ou IEEE 802.15.4, em que a existência de um módulo coordenador faz a coleta das informações dos módulos sensores e atuadores e as enviam a um software supervisório. Este software processa as informações coletadas e devolve ao módulo coordenador o modo de operação do sistema, para, posteriormente, enviá-las ao módulo atuador. O software foi implementado em linguagem Visual Basic, onde as variáveis são mostradas e analisadas, além de ter a possibilidade de mudança do *set point* da temperatura de controle. Este projeto foi dividido nas seguintes etapas:

- Implementação das telas de supervisão em linguagem Visual Basic;
- Medição da temperatura, umidade, temperatura média radiante, velocidade do vento, radiação solar;
- Coleta dos dados dos sensores, via protocolo ZigBee, através de módulos ZigBit – MeshNetics/ATMEL e envio para o software supervisório;
- Definição da estratégia de controle;
- Acionamento dos atuadores do sistema:
  - *Damper* com acionamento contínuo;
  - Split (compressor);
  - Ventilador;
  - Umidificador (bomba de água);
- Formulação matemática dos critérios de otimização do modelo de conforto térmico PMV (Predicted Mean Vote) e consumo energético.

2 Fundamentos Teóricos

2.1 Índice de conforto Térmico – PMV

De acordo com a *American Society of Heating Refrigeration and Air Conditions* (ASHRAE), conforto térmico pode ser definido como "o estado de espírito em que o indivíduo expressa satisfação em relação ao ambiente térmico". Este estado é obtido quando um indivíduo está numa condição de equilíbrio com o ambiente que o rodeia, o que significa que é possível a manutenção da temperatura dos tecidos constituintes do corpo, num domínio de variação estrito, sem que haja um esforço sensível [12].

Devido aos seres humanos serem homeotérmicos, ou seja, tendência a manutenção da temperatura interna do corpo, estes tendem ao equilíbrio entre a produção interna de calor devido ao metabolismo e à perda de calor para o meio ambiente através de processos fisiológicos, Figura 2.1.

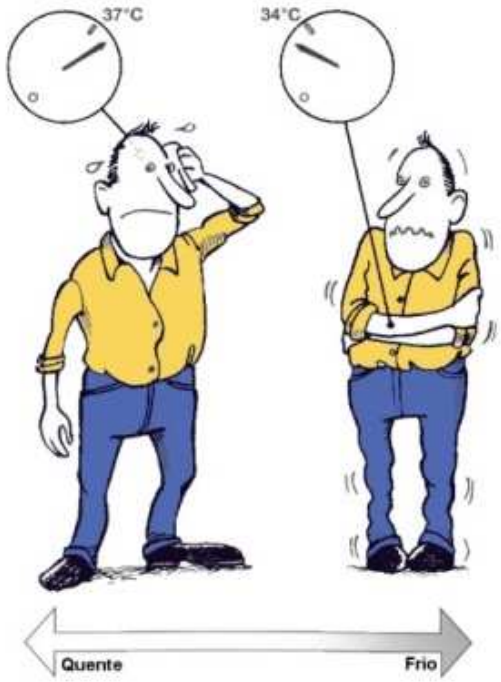


Figura 2.1 - Manutenção da temperatura interna do corpo.

Assim, a avaliação da sensação térmica e grau de desconforto de indivíduos expostos a ambientes térmicos moderados devem ser validados através de critérios e normas.

Existem várias normas e especificações com o objetivo de avaliar a sensação térmica, como exemplo, pode-se citar a norma ASHRAE 55, na qual visa especificar as características de

ambientes internos que, associadas a fatores pessoais, geram condições térmicas aceitáveis para no mínimo 80% dos ocupantes do local. Neste projeto foi escolhido a ISO 7730, proposto por Fanger em 1970, no qual utiliza o PMV, apresentando um método para previsão da sensação térmica e do grau de desconforto de pessoas expostas a ambientes de temperatura moderada. Além disso, especifica as condições térmicas aceitáveis para o conforto [11].

Desse modo, a avaliação da sensação térmica e grau de desconforto de indivíduos expostos a ambientes térmicos moderados deverá ser feita, neste projeto, com base em critérios que se baseiam na determinação dos índices PMV (Voto Médio Previsível ou *Predicted Mean Vote*) e PPD (Percentagem previsível de insatisfeitos ou *Predicted Percentage of Dissatisfied*) e aplicam-se a ambientes interiores onde se pretenda avaliar as condições ambientais em termos de conforto térmico [12]. Estes critérios seguem, em linhas gerais, as Normas ISO - 7730 (Conforto - PMV e PPD para ambientes moderados) e ISO - 7726 (Parâmetros físicos - *Thermal environments - Instruments and methods for measuring physical quantities*).

O índice de PMV é uma norma que define matematicamente o índice de conforto térmico considerando seis variáveis para o seu cálculo. Onde quatro variáveis são físicas e podem ser inferidas através de sensores e outras duas pessoais. A seguir a tabela 1 mostra as variáveis para o cálculo do índice PMV:

Tabela 2.1. Variáveis do PMV.

Variáveis pessoais	Variáveis físicas
Nível de atividade física realizada pelos ocupantes do ambiente	Temperatura do ar
	Temperatura média radiante
Tipo de roupa utilizado pelos ocupantes do ambiente	Velocidade do vento
	Umidade do ar

Cada variável possui especificações das quais se deve fazer uma análise individual, a fim de mensurá-las corretamente.

O corpo humano necessita de calor para exercer as suas atividades, age, portanto, como uma fonte de calor, oriundo do metabolismo dos alimentos que ingerimos. Assim, existem mecanismos que promovem a troca de calor com o meio ambiente, dissipando a energia gerada através das atividades físicas que exercemos [6]. A dissipação desse calor pode ser efetuada, basicamente, através da pele e da respiração. E estas se dividem (figura 2.2):

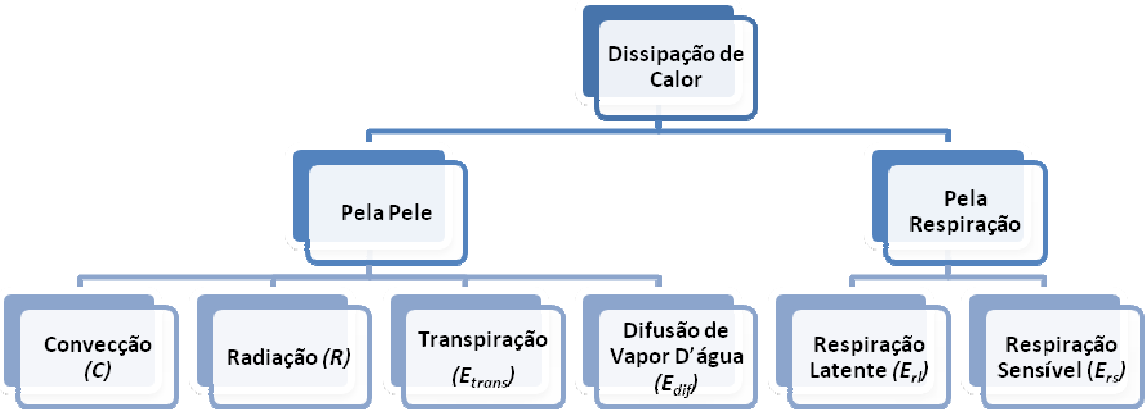


Figura 2.2 - Dissipação de calor do corpo humano

Assim, em reação às condições ambientais, a produção de calor pelo corpo humano (Taxa Metabólica - M) será igual à dissipação, fazendo com que não haja armazenamento significativo de calor por parte do mesmo. Nestas condições, o equilíbrio térmico corporal é dado por [14]:

$$M - E_{dif} - E_{trans} - E_{rl} - E_{rs} = R + C \quad (1)$$

Nesse caso, a sensação térmica relacionada ao nível de atividade física é função da produção do calor interno, e o calor perdido ocorreria sob valores confortáveis de temperatura média da pele e da taxa de suor para cada nível de atividade em análise.

A carga térmica, na qual um corpo é submetido, é dada por:

$$L = M - E_{dif} - E_{trans} - E_{rl} - E_{rs} - R - C \quad (2)$$

Em que a alteração da temperatura média da pele e da taxa de suor para manter a temperatura interna do corpo, dependerá da carga térmica submetida sobre o corpo.

Podemos, então, concluir que existe uma relação entre a sensação térmica demonstrada por uma pessoa e a carga térmica à qual está submetida. A carga térmica, por sua vez, é influenciada pela produção de calor corporal interna. Logo, podemos dizer que a sensação térmica tem relação estreita com a produção de calor corporal interna [14].
O índice PMV utiliza uma escala de sensação térmica proposta por Fanger e pode ser observada na figura 2.3, abaixo.

Figura 2.3 - Índice PMV relacionado ao nível de conforto térmico.

Para o cálculo do PMV, a seguinte fórmula é utilizada:

$$PMV=(0{,}303e^{-0{,}036M}+0{,}028)L \quad (3)$$

Onde:

M – Taxa de metabolismo em W/m^2 ;

L – Carga térmica atuante sobre um corpo em W/m^2 .

Os cálculos dos termos individuais da equação da carga térmica (L) podem ser visto no Anexo 1.

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 [15].

$$PPD=100-95e^{(-0{,}03353PMV^4-0{,}2179PMV^2)} \quad (4)$$

E, assim, a relação do índice do PMV e do PPD é tal, como pode ser observada no gráfico da figura 2.4..

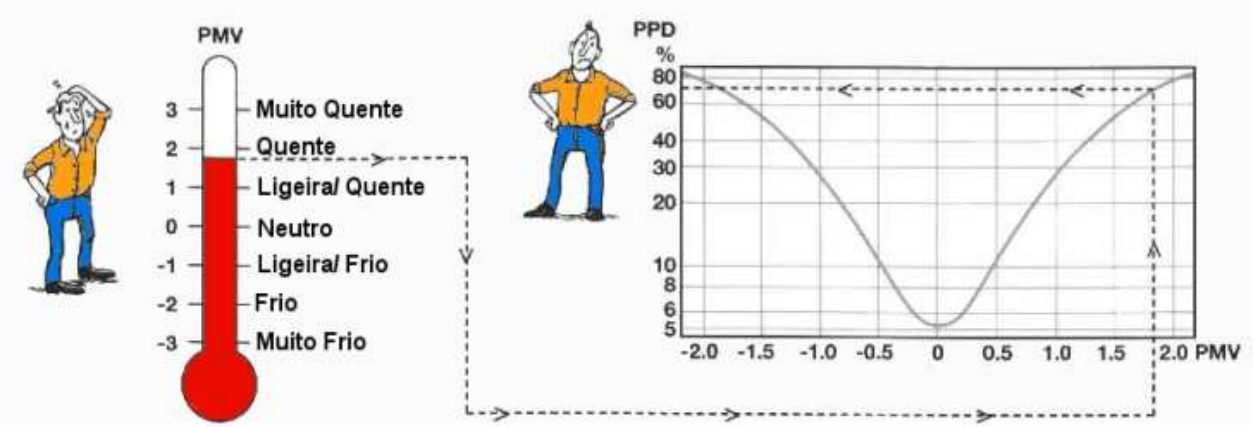


Figura 2.4 - Percentagem previsível de pessoas insatisfeitas (PPD) em função do voto médio previsível (PMV).

Deve-se notar, ainda, que os índices PMV e PPD exprimem sensações de desconforto em relação à totalidade do corpo. Nesse sentido, recomenda-se que o PPD seja inferior a 10% correspondendo este valor ao seguinte critério para o PMV [12]:

$$-0,5 < PMV < +0,5 \quad (5)$$

Na medição do índice do PMV existem três maneiras:

- Uso direto da equação mostrada na equação (3);
- Consulta à tabelas contendo valores do PMV para diversas combinações entre o nível de atividade, a temperatura do ambiente e a velocidade do vento;
- Através da medição direta dos parâmetros de interesse com o auxílio de sensores integrados.

Neste projeto será utilizada a terceira maneira para inferir o PMV.

2.2 ZigBit – MeshNetics

A implementação da rede de sensores sem fio tem como principal componente o módulo ZigBit. Ele foi, originalmente, comercializado pela MeshNetics, empresa que criou a integração de módulos com padrão de comunicação ZigBee e software usado em OEMs e sistemas integradores de produtos e soluções de conectividade sem fio. Posteriormente o ZigBit passou a ser comercializado pela ATMEL. Nessa mudança, alguns componentes do módulo tiveram seus nomes alterados, principalmente de software.

O módulo ZigBit possui um encapsulamento que contém um microcontrolador ATmega128 e o *transceiver* de radiofreqüência AT 86RF230. Pode-se, também, escolher a versão com

antena integrada ao módulo ou sem antena, figura 2.5. O módulo é um sistema cooperativo multitarefa e permite o carregamento de aplicações, diferente do último módulo comercializado pela empresa MeshNetics, o módulo XBee.

A configuração da pilha deste módulo se dá pelo BitCloud, antigamente chamado de ZigBeeNet.

BitCloud é uma plataforma de desenvolvimento de software ZigBee PRO certificado para aplicações *wireless* de tempo real com agendamento de tarefas [16], o que o tornou apropriado para a este trabalho, juntamente com o protocolo de comunicação utilizado, o padrão IEEE 802.15.4, ZigBee.

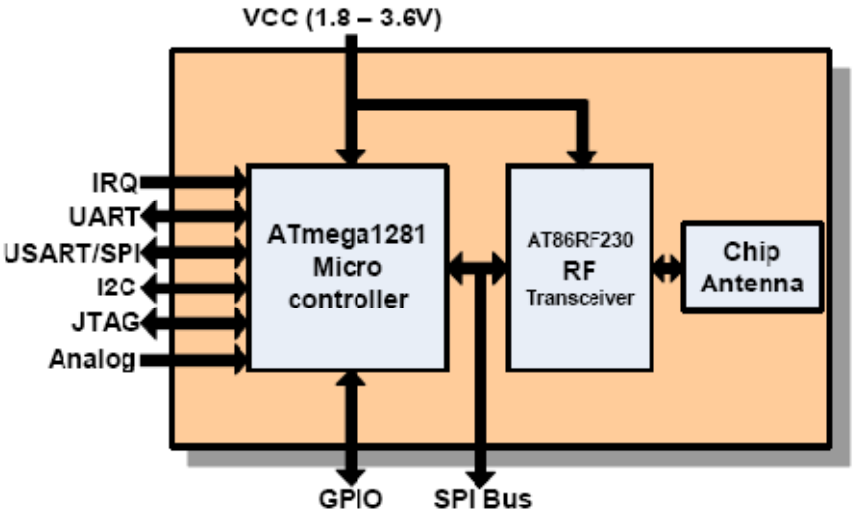


Figura 2.5 – Esquemático do módulo ZigBit

A comunicação entre o módulo e o computador pode ser feita pelo formato padrão para comunicação de dados serial, protocolo USART ou UART. Em que se utiliza a USART nos modos síncronos e assíncronos, obtendo maior versatilidade do dispositivo. E a UART para comunicações de maiores distâncias e a sincronização feita via software.

Outra vantagem do módulo ZigBit para implementação da rede de sensores sem fio é a quantidade de GPIOs e ADC no módulo, 25 e 4, respectivamente, garantindo uma grande quantidade de aplicações de diferentes implementações.

2.2.1 Padrão ZigBee – IEEE 802.15.4

Visando o desenvolvimento de um protocolo padrão para redes de sensores wireless, um consórcio de empresas (*ZigBee Alliance*), juntamente com o IEEE (*Institute of Electrical and Electronics Engineers*) desenvolveu o padrão ZigBee, que designa um conjunto de especificações para a comunicação sem-fio entre dispositivos eletrônicos, com ênfase na

baixa potência de operação, na baixa taxa de transmissão de dados e no baixo custo de implantação, sendo, também, um padrão aberto e global.

A automação predial vem crescendo em importância e em particular redes sem fio ZigBee, que foram desenvolvidas especificamente para este fim, têm sido empregadas. O ZigBee é um padrão alternativo para redes de comunicação que não necessitam de muito controle e é projetado para oferecer flexibilidade quanto aos dispositivos que pode controlar.

As aplicações mais comuns são, basicamente, controle de dispositivos como relês, trancas eletromagnéticas, ventilação, aquecimento, motores e eletrodomésticos, e aquisição de dados de sensores, como de temperatura, umidade, luminosidade, pressão, velocidade do vento, etc.

As principais vantagens da utilização do protocolo ZigBee - IEEE 802.15.4 são:

- Possibilidade de configuração automática da rede;
- Endereçamento dinâmico de módulos escravos;
- Controle por *handshaking* nas transferências de pacotes, garantindo maior integridade dos dados;
- Modos de endereçamento 16-bit short e 64-bit IEEE *addressing*;
- Freqüência de 2,4GHz ISM (*Industrial, Scientific, and Medical*), com exceção dos Estados Unidos e Europa, que trabalham com a freqüência de 915MHz e 868MHz, respectivamente;
- Taxa máxima de transmissão de 250Kbps;
- Gerenciamento de energia para assegurar baixo consumo.

A tecnologia utilizada é comparável às redes *Wi-Fi* e *Bluetooth* e diferencia-se destas por desenvolver menor consumo, por um alcance reduzido (cerca de 10 metros) e as comunicações entre duas unidades poder ser repetida sucessivamente pelas unidades existentes na rede até atingir o destino final. Funcionando todos os pontos da rede como retransmissores de informação, uma malha de unidades ZigBee pode realizar-se numa extensão doméstica ou industrial sem necessidade de utilizar ligações elétricas entre elas [10].

2.2.2 BitCloud – ZigBeeNet

A plataforma de desenvolvimento de software ZibBee , o BitCloud, caracteriza-se por tempo real, já que é um software que gerencia os recursos computacionais do microprocessador,

com o objetivo de garantir que todos os eventos sejam atendidos dentro de suas restrições de tempo, e que sejam gerenciados da forma mais eficiente possível. Softwares assim, responsáveis pelo gerenciamento dos recursos computacionais, também são chamados de Kernel (ou núcleo) do Sistema de Tempo Real, e conhecido no mercado como RTOS (*Real-Time Operation System*). E assim, a pilha do BitCloud se caracteriza como um sistema de tempo real, executando tarefas em determinados períodos, juntamente a eventos assíncronos.

A arquitetura do BitCloud segue as orientações para a separação de camadas conforme IEEE 802.15.4 - 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 com hardware *Abstraction Layer* (HAL) [17].

As camadas da pilha do BitCloud podem ser observadas na figura 2.6 abaixo e serão descritas, as principais camadas logo em seguida de acordo com [18].

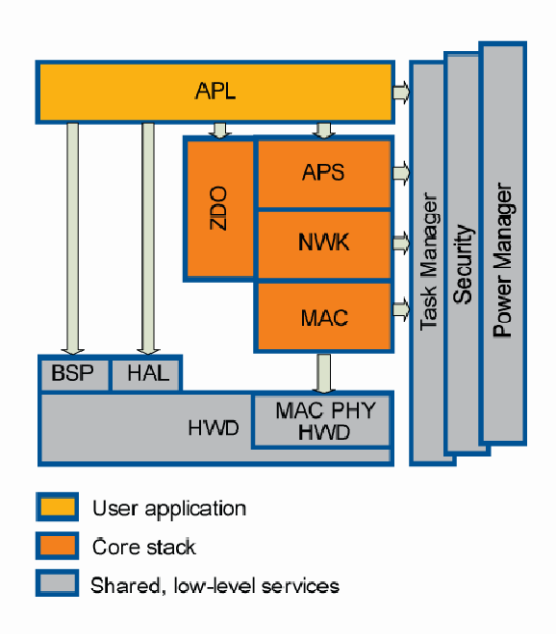


Figura 2.6 – Arquitetura da pilha do software BitCloud

- *Application Support Sublayer* (APS) – Localizada no topo da pilha central, proporciona o mais alto nível das APIs (*Application Programming Interface*) relacionadas com a rede;
- ZigBee *Device Object* (ZDO) – APIs que implementam a gestão da rede e da energia, juntamente com a definição do tipo do dispositivo, módulo coordenador ou módulo *enddevice*;

- *Hardware Abstraction Layer* (HAL) – APIs que utilizam dos recursos de hardware (EEPROM, App, *Sleep*, e *Watchdog timers*) e *drivers* que facilitam a integração com periféricos externos (IRQ, I^2C, SPI, UART, 1-wire);
- *Task manager, Security* e *Power Manager* – Camadas de serviço que estão disponíveis ao usuário, que podem ser como exemplo, agendamento de tarefas.

A concentração das atividades características de uma WSN (*Wireless Sensor Network*) em um único microcontrolador necessita de uma estratégia de programação que garanta o funcionamento adequado de todas as tarefas críticas. Sendo assim, as tarefas críticas não podem levar mais tempo do que 50µs. E na camada de aplicações, APS, um tempo máximo de 50ms. As limitações da plataforma de hardware utilizada não permitem um verdadeiro sistema operacional, portanto, o BitCloud utiliza um sistema conhecido como *Event-Driven Systems* [17]. Este paradigma de programação encaminha o fluxo do programa de acordo com a determinação de eventos, outputs de sensores, ações do usuário, ou mesmo *threads*.

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 microcontrolador [17].

O hardware em questão apresenta limitações quanto à memória, portanto, não é possível armazenar todo o estado do sistema para poder interrompê-lo, justificando a não utilização de rotinas muito extensas.

3 Desenvolvimento

O projeto deve fazer a integração dos módulos de aquisição de dados *wireless*, o sistema de atuadores e o software supervisório, relativo ao funcionamento do ar condicionado híbrido. O sistema híbrido de climatização utilizado no projeto foi construído por José Luís Olmos Flores, em sua tese de mestrado, na sala de reunião do GRAV [2], figura 3.1.

Algumas informações relevantes à construção do ar condicionado híbrido se encontram no Anexo II.

Figura 3.1 – Ar Condicionado Híbrido do LAVSI

Com a parte sistêmica funcionando deve-se então implementar técnicas de estratégia de controle visando o conforto térmico e a economia de energia. Sendo assim, várias possibilidades de implementação foram discutidas numa tentativa de se melhor aproveitar a parte já implementada por outros projetos concluídos.

3.1 Possibilidades de implementação utilizando projetos concluídos

Para validar os dados e fazer a defesa, José Luís Olmos Flores [2] utilizou a solução do software de gerenciamento remoto de sistemas de refrigeração e climatização Sitrad da empresa Full Gauge, juntamente com CLP's, sensores e atuadores da própria empresa. Os sensores utilizados fazem a medição das variáveis de temperatura e umidade. Estes foram dispostos dentro do sistema de refrigeração dificultando a manutenção, já que se deve abrir todo o sistema híbrido para se realizar tal procedimento. Na tentativa de se aproveitar os sensores já instalados inicialmente no sistema, desenvolveram-se possíveis alternativas de projetos. Assim, as opções de projetos serão descritas abaixo.

3.1.1 Opção A – Sitrad (Banco de dados)

Através da comunicação USB entre a estação de trabalho, que contém o sistema Sitrad e as controladoras Full Gauge, possibilita-se avaliar, configurar e armazenar dados de temperatura e umidade, utilizando como ferramenta um Banco de Dados MySQL, em que os dados podem ser amostrados no formato arquivo de texto (".txt") ou banco de dados (.SQL). Dado que a transmissão dos dados das controladoras para o sistema Sitrad são criptografados, utilizando, portanto, a mesma estação de trabalho do software Sitrad, ao novo software supervisório implementado, uma opção para se aproveitar a instalação dos sensores de umidade e temperatura seria habilitar uma replicação UDP dos dados do software supervisório Sitrad para o novo software supervisório. Os dados devem ser replicados em tempos sincronizados e com menor tempo de atraso possível, pois as variáveis devem ser lidas em tempo real. As informações coletadas pelo Sitrad seriam, portanto, transferidas para o supervisório, aproveitando todo o sistema de coleta de dados implementado inicialmente. Porém, como as controladoras instaladas possuem um sistema fechado, não há a possibilidade de se implementar e/ou modificar algoritmos de controle, além do tipo liga-desliga, que compõe o sistema Sitrad, emitindo sinais aos atuadores, o que resultou investigar uma nova metodologia.

3.1.2 Opção B – Sitrad (RS-485)

Ainda no intuito de conservar a instalação dos sensores de temperatura e umidade, uma nova opção para se fazer a aquisição desses dados seria utilizar a comunicação RS-485 utilizada no sistema Sitrad. Neste caso a opção seria interpretar o protocolo do sistema a partir de tentativa e erro, já que a interpretação dos dados se daria a partir do envio de cada leitura de dado e/ou envio de comando emitido por comunicação RS-485 do Sitrad e assim,

investigar o que cada *bitfield* de todos os *frames* significam. Após a investigação deste protocolo, um programa deveria ser implementado para que as informações fossem "montadas e desmontadas" em *frames* a fim de utilizar essa comunicação. Pois este programa deveria ler os dados em RS-485, mas também deveria replicar esses dados no Banco de Dados do sistema do novo supervisório. Porém, nesta opção, o problema da programação fechada dos métodos de controle e dos atuadores, não foi solucionado, assim como na "opção A", além da dificuldade de inserção de novas variáveis na rede de sensores. Sendo assim, nessa opção também deveria ser construído um novo módulo para o acionamento e módulos sensores, o que gerou em uma nova proposta.

3.1.3 Opção C – Banco de Dados (Replicação UDP)

A nova opção desenvolvida também manteria a tentativa de se "aproveitar" os sensores de temperatura e umidade já instalados. Assim, a nova opção de se fazer a instrumentação e controle do sistema híbrido seria englobar a opção A, juntamente com a implementação de módulos atuadores utilizando o kit de desenvolvimento ZIGBIT da empresa MeshNetics/ATMEL, no qual possui padrão IEEE 802.15.4 – ZigBee e um microcontrolador integrado ATMEGA 128. A comunicação entre o ZigBit e o supervisório se daria a partir do USBCON, em que o programa CP-2102 simula uma porta serial RS-232 e a aquisição dos dados dos sensores se faria pela replicação do banco de dados do sistema Sitrad para o banco de dados do software supervisório implementado (Opção A). O acionamento se daria a partir de dispositivos de controle relés de estado sólido, dos quais seriam controlados por sinais enviados pelo supervisório através do USBCON. Nessa implementação, o sistema contaria com comunicação parcialmente *wireless* e a outra parte seria dada a partir da replicação do Banco de Dados.

3.1.4 Opção D – ZigBit e novos sensores

Outra opção seria a implementação do sistema como um todo, aproveitando, somente, a parte mecânica e elétrica do projeto do Ar Condicionado Híbrido do José Luis Olmos Flores [2]. A nova rede de sensores seria implementada por totalidade e atuaria concomitantemente à rede de sensores/controladora da Full Gauge, implementada no projeto anterior, sendo somente o acionamento não compartilhado. Dessa maneira, também se implementaria um novo módulo atuador, em que este teria a ação de ligar e desligar o sistema evaporativo e de refrigeração, através da comunicação *wireless* do software supervisório/coordenador com o módulo atuador.

O sensor SHT-71 ficou responsável pela aquisição dos dados de temperatura e umidade, o sensor TY321A1009 pela temperatura média radiante, o anemômetro Dwyer 641-12-LED pela velocidade do vento e o sensor Silicon Pyronometer SLIB M003 pela radiação solar. Os relés adotados foram o produto T2405Z-M. E assim, através de toda essa rede de sensores, os módulos estariam aptos a fazer todo o controle e a instrumentação do ar condicionado híbrido.

3.2 Implementação Final

Para melhor validação dos resultados, maior flexibilidade e continuidade deste projeto em desenvolvimentos posteriores, optou-se pela opção D, implementando todo um novo sistema de automação, concomitantemente ao que já havia no ambiente.

Para isso, foram elaborados cinco módulos diferentes para o monitoramento dos sensores e acionamento dos atuadores do sistema. Os módulos foram criados para garantir a alimentação adequada e a leitura correta dos valores assim como a proteção dos dispositivos eletrônicos e circuitos impressos. De acordo com a estratégia utilizada para o monitoramento e controle da sala foram criadas quatro placas. Sendo três placas de monitoramento contendo apenas sensores e um módulo ZigBit, e uma placa de acionamento para todos os atuadores (bomba, compressor, ventilador e *damper*). Ainda foi criada uma placa gravadora com o objetivo apenas de alimentar o circuito ZigBit para a gravação de programas e testes de portas.

O sistema híbrido, o ambiente a ser controlado (Sala de reunião do Laboratório LAVSI) e a localização dos módulos podem ser averiguados na figura 3.2.

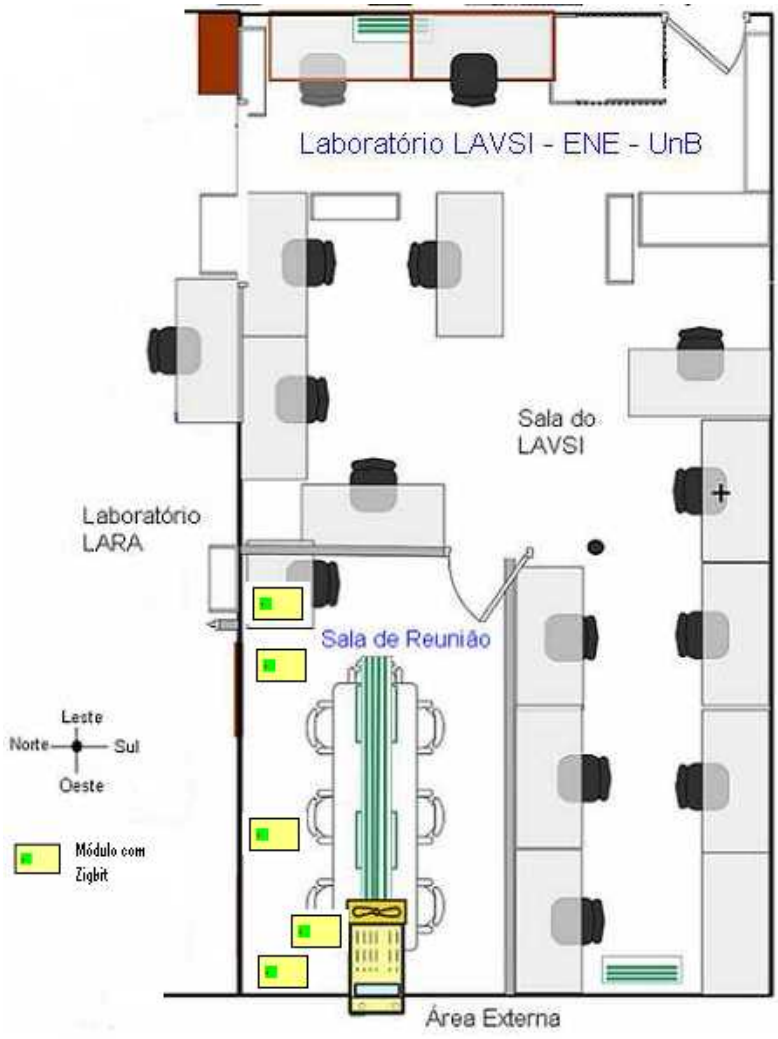


Figura 3.2 – Localização dos módulos no LAVSI

Os módulos desenvolvidos e seus principais componentes são:

- Módulo Externo:

  - Módulo ZigBit (01 unidade);

  - Sensor de Temperatura e Umidade SHT-71 (01 unidade);

  - Piranômetro Silicon Pyronometer SLIB-M003 (01 unidade).

- Módulo Interno:

  - Módulo ZigBit (01 unidade);

  - Sensor de Temperatura e Umidade SHT-71 (03 unidades);

  - Sensor de Radiação Térmica TY321A1009 (01 unidade).

- Módulo Móvel:

  - Módulo ZigBit (01 unidade);

- Sensor de Temperatura e Umidade SHT-71 (01 unidade);

- Anemômetro Dwyer 641-12-LED (01 unidade).

- Módulo Atuador:

  - Módulo ZigBit (01 unidade)

  - Atuador do compressor;

  - Atuador do *damper*;

  - Atuador do ventilador;

  - Atuador da bomba hidráulica;

- Módulo Gravador:

  - Alimentação do circuito de entrada e saídas RS-232, e principais bornes do ZigBit disponibilizados.

3.3 Sensores e Atuadores do Sistema

Os sensores e atuadores utilizados no sistema serão descritos abaixo, a fim de se englobar o resultado final alcançado neste projeto, ou mesmo a justificativa dos esquemas elétricos implementados dos módulos.

3.3.1 Anemômetro Dwyer 641-12-LED

Anemômetros são instrumentos que servem para medir a direção e indicar a velocidade do vento. O anemômetro utilizado neste projeto foi o anemômetro da série 641, marca Dwyer, mostrado na figura 3.3. A transmissão da velocidade do ar dessa série utiliza a tecnologia de fluxo de massa quente. Ele possui oito seleções diferentes de intervalos de medição, que variam de 1,25m/s a 75m/s. E pode atingir uma resolução de 0,01m/s.

Figura 3.3 - Anemômetro Dwyer

Outras especificações do sensor podem ser vistas na tabela abaixo de acordo com [20].

Tabela 3.1 – Especificações do Anemômetro

SPECIFICATIONS
Service: Clean air and compatible, non-combustible gases.
Accuracy:
3% FS Process gas: 32 to 122°F (0 to 50°C).
4% FS Process gas: -40 to 32°F & 122 to 212°F
(-40 to 0°C & 50 to 100°C).
Response Time: Flow: 1.5 seconds to 95% of final value (Output filter set to minimum).
Temperature Limits: Process: -40 to 212°F (-40 to 100°C).
Ambient: 32 to 140°F (0 to 60°C).
Pressure Limit: 100 psi (6.89 bar) maximum.
Humidity Limit: Non-Condensing.
Power Requirements: 12–35 VDC, 10–16 VAC.
Output Signal: 4-20 mA, isolated 24V source, 3 or 4-wire connection.
Output Filter: Selectable 0.5 –15 (seconds).
Loop Resistance: 600 ohms max.
Current Consumption: 300 mA max.
Electrical Connections: Screw terminal.
Process Connections: 1/2˝ male NPT.
Enclosure Rating: Designed to meet NEMA 4X (IP66) for non LED models only.
Mounting Orientation: Unit not position sensitive. Probe must be aligned with airflow.
Weight: 12.6 oz (357.2 g).
Agency Approval: CE.

OPTIONAL DISPLAY VERSION:
Display: 4-1/2 digit 1/2˝ red LED.
Resolution: 1 FPM, 0.01 MPS
(10 FPM @ 10,000 and 15,000 FPM ranges).
Weight: 13.3 oz (377 g).

A conexão elétrica deste sensor é fácil e flexível. Pode ser alimentado com uma fonte AC ou DC. A figura 3.4 mostra a ligação elétrica utilizada neste projeto para a alimentação e recepção do sinal de output de 4 - 20mA.

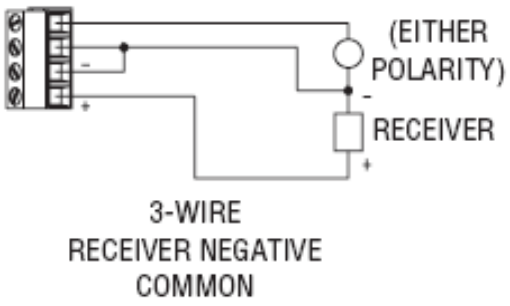


Figura 3.4 - Ligação elétrica do anemômetro

A seleção da variação da faixa de atuação configurada neste sensor para validação dos resultados foi de 0 – 1,25m/s, com resolução de 0,01m/s.

3.3.2 Sensor de Temperatura e Umidade SHT-71

O SHT-71, sensor de temperatura e umidade (figura 3.5), apresenta a vantagem de ser digital, evitando a construção de um circuito auxiliar para a regulagem de tensão e amplificação do sinal de saída. Além de poder ser implementado usando portas I/O's ao invés de portas analógicas.

Apresenta uma acurácia de ± 0,4℃ para a temperatur a e de ± 3% para umidade relativa e uma resolução de 0,01℃ para a temperatura e 0,05 U R para a umidade, com 14 e 12 bits de resolução, respectivamente.

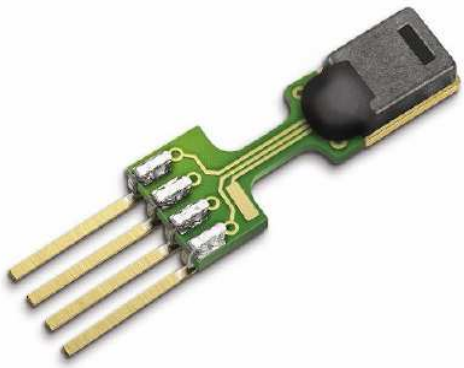


Figura 3.5 – SHT-71

O sensor SHT-71 pode ser alimentado com uma tensão de 2,4 – 5,5V DC e possui um capacitor de 100nF integrado que funciona como um filtro, o qual não continha nas versões anteriores. A condição de operação depende da temperatura em que o sensor está exposto, como pode ser observado na figura 3.6.

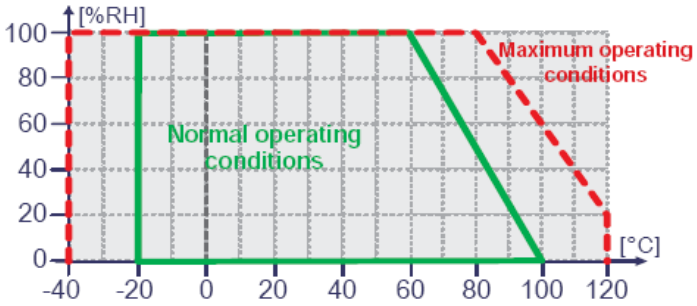


Figura 3.6 – Condição de Operação e Memória

Abaixo, tem-se um circuito de aplicação típico utilizando um microcontrolador incluindo um resistor pull up (Figura 3.7).

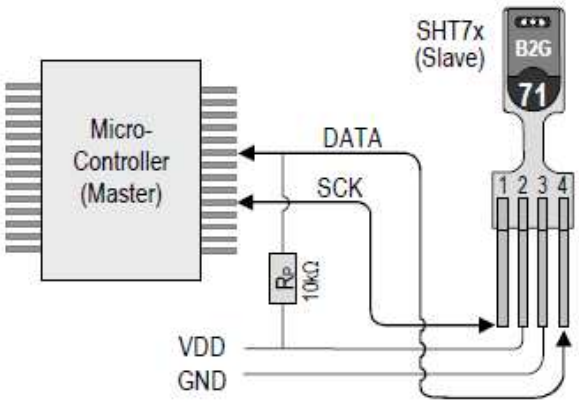


Figura 3.7 – Esquema Elétrico do SHT-71

A figura 3.8 mostra o funcionamento interno de comunicação do sensor. O SHT apresenta um conversor A/D integrado, disponibilizando suas medições de forma digital. A comunicação é feita por dois fios, um de dados e outro para o *Clock,* utilizados para a leitura das informações. O sensor utiliza protocolo proprietário semelhante ao I^2C e uma comunicação *half-duplex*, ou seja, o pino do microcontrolador Atmega128 ligado a linha de dados hora deve ser configurado como *output* hora como *input*.

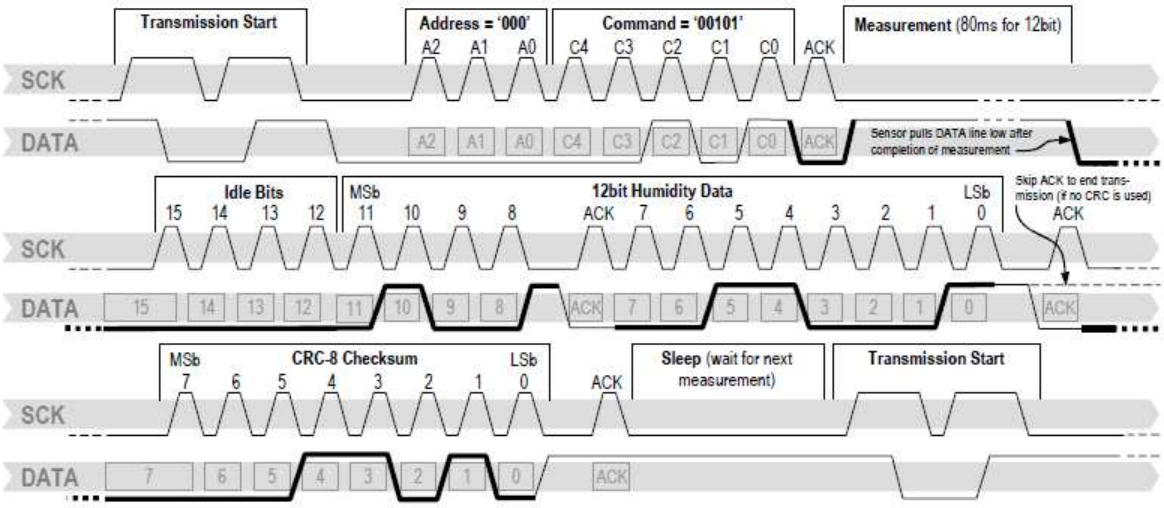


Figura 3.8 – Exemplo de seqüência de transmissão e coleta dos dados

A tabela 3.2 mostra os parâmetros gerais e elétricos do funcionamento do SHT.

Tabela 3.2 – Especificações do SHT-71

Parameter	Condition	min	typ	max	Units
Source Voltage		2.4	3.3	5.5	V
Power Consumption[5]	sleep		2	5	µW
	measuring		3		mW
	average		150		µW
Communication	digital 2-wire interface, see Communication				
Storage	10 – 50°C (0 – 80°C peak), 20 – 60%RH				

3.3.3 Sensor de Radiação Térmica TY7321A1009

O sensor de temperatura média radiante mensura a intensidade relativa da radiação térmica irradiada dos objetos de uma determinada área. A radiação infravermelha é um elemento crítico que determina o que é inferido como calor [19].

Este sensor, da marca YAMATAKE e modelo TY7321A1009 (Figura 3.9), possui como elemento sensor uma termopilha em miniatura, que produz uma voltagem proporcional à intensidade da radiação e um sinal de output via um circuito interno de processamento de sinal. A voltagem de output varia de 1 - 5V DC, de acordo com a temperatura radiante de 0 - 50℃.

Figura 3.9 – Sensor de Radiação Térmica TY7321A1009

A ligação correta do sensor deve seguir a figura 3.10, onde se deve usar um transformador de 24VAC separado de outros componentes e individualizado para cada sensor caso estejam ligados em paralelo, para que não venha a ocorrer nenhum dano. Caso a alimentação seja compartilhada com outro equipamento, o sensor pode vir a queimar se exposto a esta alimentação compartilhada.

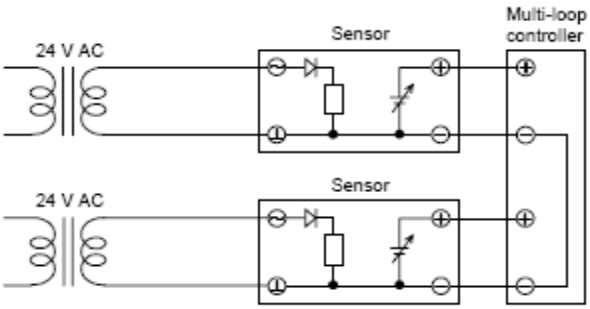


Figura 3.10 – Ligação elétrica do Sensor de Radiação Térmica TY7321A1009

Na tabela abaixo podem ser observadas outras especificações deste sensor.

Tabela 3.3 – Especificações do Sensor de Radiação Térmica TY7321A1009

Specifications

Item	Specification	
Sensing range	Radiant temperature:	5 °C to 50 °C
Sensing accuracy	Radiant temperature: Room temperature:	+/-2 °C (ambient temperature 25 °C) ±0.35 °C (TY7321B), ±0.7 °C (TY7321C)
Power supply	24 V AC +/-15 %	
Frequency	50/60 Hz +/-4 %	
Power consumption	Max. 0.3 VA	
Output signal	Radiant temperature: Room temperature:	1 to 5 V DC (corresponds to radiant temperature 0 to 50 °C) Pt3K• •(TY7321B), Pt100 (TY7321C)
Time constant	Radiant temperature: Room temperature:	Max. 10 seconds Max. 3 minutes
Rated conditions	Ambient temperature: Ambient humidity: Vibration:	15 to 35 °C 10 to 90 % RH (non-condensing) 2 m/s² (10 to 55 Hz)
Transport/storage conditions	Ambient temperature: Ambient humidity: Vibration:	-20 to + 60 °C 5 to 95 % RH (non-condensing) 9.8 m/s² (packed)
Effective range	52 °	
Sensor coverage	344 ° (horizontal) 65 ° (vertical, in 5 ° steps)	
Weight	Approx. 200 g	
Color	Base, housing, terminal cover: Cover: Case, housing, cover of sensor :	DIC 546 1/2 White White
Materials	Base, housing, terminal cover : Cover : Case, housing, sensor cover :	Molded polycarbonate resin, equivalent to UL V-O Molded fire-resistant ABS resin, equivalent to UL V-O Molded polycarbonate resin, equivalent to UL V-O

3.3.4 Piranômetro Silicon Pyronometer SLIB-M003

A radiação solar incidente no topo da atmosfera terrestre varia basicamente com a latitude e o tempo, no qual, ao atravessar a atmosfera, interage com seus constituintes e parte dessa radiação que é espalhada em outras direções é especificada de radiação solar difusa e a

outra parte que chega diretamente à superfície do solo é denominada de radiação solar direta. Somando a radiação difusa com a direta obtém-se a radiação solar global. A radiação global é medida por um radiômetro específico, denominado piranômetro [21]. Assim, o piranômetro é um instrumento usado para medir a radiação solar com uma varredura de 180°, justificando a sua instalação.

O sensor de radiação solar da empresa Global Water's, figura 3.11, utiliza a alta estabilidade do detector fotovoltáico de silicone para obter uma leitura com boa acurácia [21]. O sinal de output deste sensor é de 4 – 20mA e possui uma alimentação de 10 – 36VDC, que o torna bastante flexível em relação a sua implementação junto a um módulo receptor do seu sinal de saída.

Figura 3.11 – Piranômetro Silicon Pyranometer SLIB-M003

As principais especificações deste produto podem ser vistas na tabela abaixo.

Tabela 3.4 – Especificações do Piranômetro Silicon Pyranometer SLIB-M003

Solar Radiation Sensors Specifications

Detector: High-stability silicon photovoltaic detector (blue enhanced).
Output: 4-20 mA
Range: 0 to 1500W/m^2
Spectral Response: 400 to 1100 nm
Accuracy: 1% of full scale
Operating Voltage: 10-36 VDC
Current Draw: Same as sensor output
Warm Up Time: 3 seconds minimum
Operating Temp: -40° to +131°F (-40° to +55°C)
Sensor Size: 3inch diameter x 1-1/2 inch (7.6 cm dia. x 3.8 cm long)
Weight: 1/4 lb. (114 g)

A ligação elétrica deste sensor deve ser tal que o fio vermelho deve ser alimentado (10 – 36VDC) e o fio preto é o sinal de output de 4 – 20mA.

3.3.5 Relé de Estado Sólido - T2405Z-M

Um dispositivo muito utilizado para acionamento de cargas de corrente alternada são as chaves estáticas. Elas promovem o chaveamento da potência fornecida para a carga, liga e desliga, mas não a modifica em nenhum outro aspecto. Chaves estáticas, disjuntores, contactores e relés de estado sólido são exemplos de dispositivos utilizados na área de chaveamento estático [7].

Os relés de estado sólido (Solid-State Relays – SSR's) consistem na alternativa moderna ideal para uso em lugar dos antigos relés eletromecânicos com seus contatos ruidosos e seus problemas mecânicos. Na verdade, uma tendência é a substituição dos relés antigos pelos de estado sólido, porém ainda existem aplicações em que isso ainda não é possível.

A escolha de se usar o SSR T2405Z-M da Teletrom, (Figura 3.12) foi devido à facilidade no seu uso e manuseio dentro deste projeto, pelo do fato que a sua alimentação pode variar de 3-32V DC e sua saída de 240V AC e até 5A, e a saída do ZigBit é de 3,3V DC.

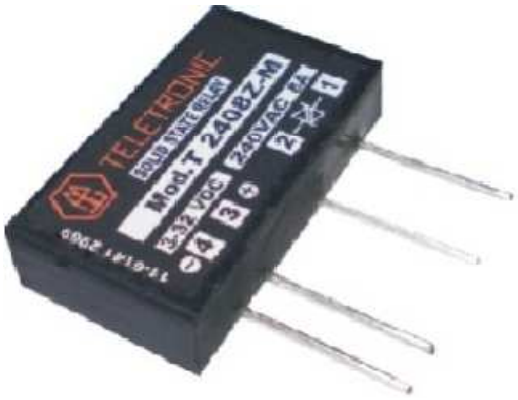


Figura 3.12 – Relé de Estado Sólido – Teletronic

Em um SSR tem-se um dispositivo semicondutor que é excitado emitindo radiação infravermelha (por exemplo). Essa radiação atua sobre um sensor que então é usado para controlar um circuito externo através de um dispositivo de potência como, por exemplo, um transistor de potência, um SCR, um TRIAC, etc [22].

Os SSR's escolhidos para este projeto, substituem os relés eletromecânicos com as seguintes vantagens [23]:

- Vida útil muito superior;

- Imune a ambientes corrosivos e vibrações mecânicas;
- Isento de centelhamento, não provoca combustão;
- Necessita a partir de 10mA para o acionamento e pode ser comutado diretamente por microprocessadores;
- Totalmente silencioso e rápido;
- Acoplado opticamente e encapsulado.

O SSR da empresa Teletronic, modelo MINI-I/O, possui o esquema elétrico equivalente ao da figura 3.13.

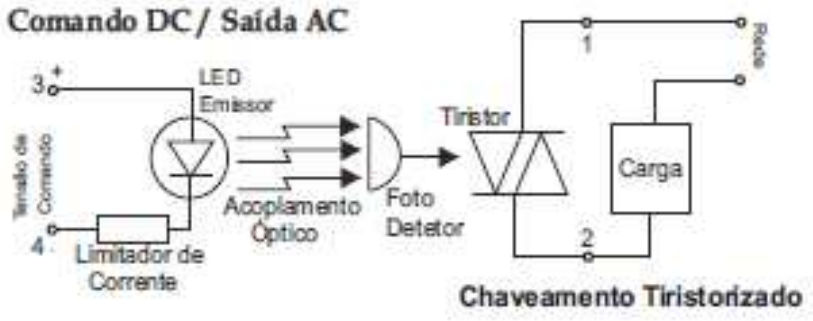


Figura 3.13 – Esquema Elétrico – SSR Teletronic

3.4 Módulos sensores/atuador de medição de dados *wireless*

No intuito de aplicar este trabalho em qualquer ambiente foram projetados módulos sem fio para aquisição e processamento de dados. Para a comunicação, sem fio, dos módulos utilizou-se a tecnologia ZigBee. Os sensores que compõem a "observabilidade" do sistema, para um cálculo preciso do PMV, são incluídos no sistema de forma que em sua implementação pode-se ter a realização do controle térmico de um ambiente utilizando conceitos de automação predial *wireless* e ambientes inteligentes.

3.4.1 Módulo Coordenador

O módulo coordenador desenvolvido tem a função de teste operacional e é o único módulo em qual se faz a gravação de programas nos módulos ZigBit.

Neste módulo, as principais portas do módulo ZigBit foram disponibilizadas em bornes, para flexibilizar e facilitar o manuseio do módulo na realização de testes e manutenabilidade do sistema (Figura 3.14).

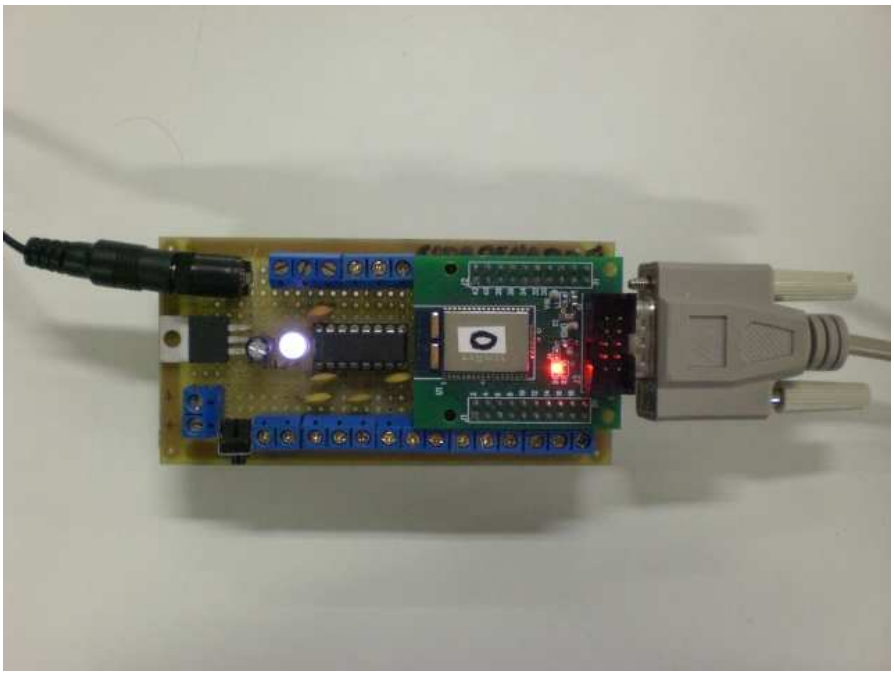


Figura 3.14 – Módulo Coordenador

O módulo coordenador é responsável por receber os valores mensurados dos módulos sensores e do módulo atuador e enviá-los, via USBCON, que simula uma porta serial para um software supervisório em um computador.

Os dados mensurados e enviados pelos outros módulos são atualizados no software supervisório de acordo com a recepção de qualquer novo valor no coordenador.

Com o recebimento dos dados, o software supervisório avalia se os equipamentos devem ser ligados ou não e manda esta informação de volta ao coordenador, para que este repasse ao módulo atuador. O esquema elétrico do módulo coordenador pode ser visto na figura 3.15. E a programação deste módulo pode ser vista no Anexo III.

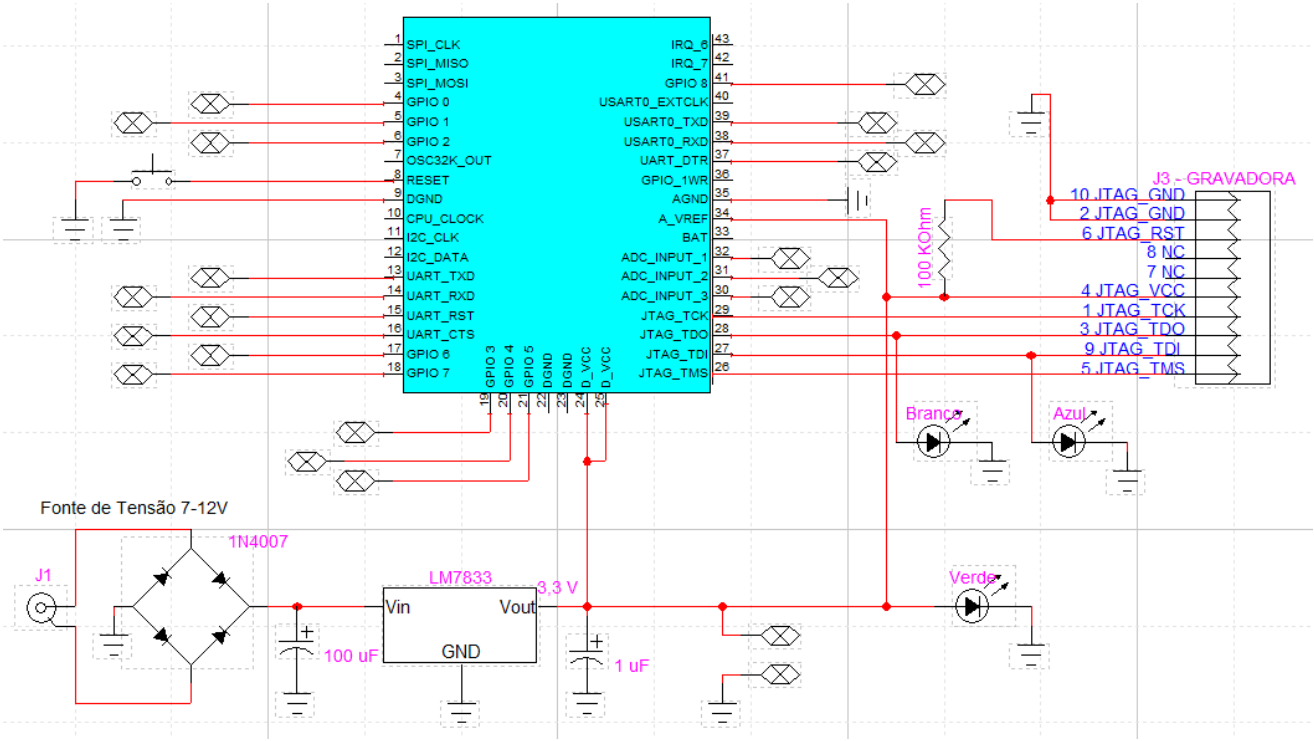


Figura 3.15 – Esquema Elétrico do Módulo Coordenador

3.4.2 Módulo Atuador

O módulo atuador é o destinatário da análise realizada pela rede de sensores para que seja feita a atuação no sistema do ar condicionado híbrido (Sistema de Refrigeração e Sistema Evaporativo), figura 3.16. Ele tem função de gerenciar os SSR's que farão o Liga/Desliga do compressor, do *damper*, do ventilador e da bomba d'água, além de enviar ao módulo coordenador o *status* destes equipamentos.

Figura 3.16 – Módulo Atuador

De acordo com o diagrama elétrico do ar condicionado híbrido elaborado por José Luís Olmo Flores [2] o acionamento do *damper* (figura 3.17) é contínuo e ocorre sempre que o compressor estiver ligado. Porém o acionamento do compressor só é validado se o ventilador também receber o comando de ligar. Sendo assim, para que o modo refrigeração ligue, dever-se mandar o comando para acionar o ventilador e o compressor e, assim o *damper* se abre automaticamente, fazendo que o mesmo ar que se encontra dentro da sala circule pelo sistema de refrigeração.

Figura 3.17 – Acionamento do *Damper*

Já para se fazer o acionamento do modo evaporativo, tem-se que mandar o comando de acionar o ventilador e a bomba d'água. Portanto, neste modo de operação o ar que circula no ambiente monitorado é o do meio externo.

Depois de setar o acionamento de acordo com o recebido, o módulo atuador manda o *status* dos SSR's ao módulo coordenador.

O modelo elétrico pode ser visualizado na figura 3.18, abaixo e o seu código no Anexo III.

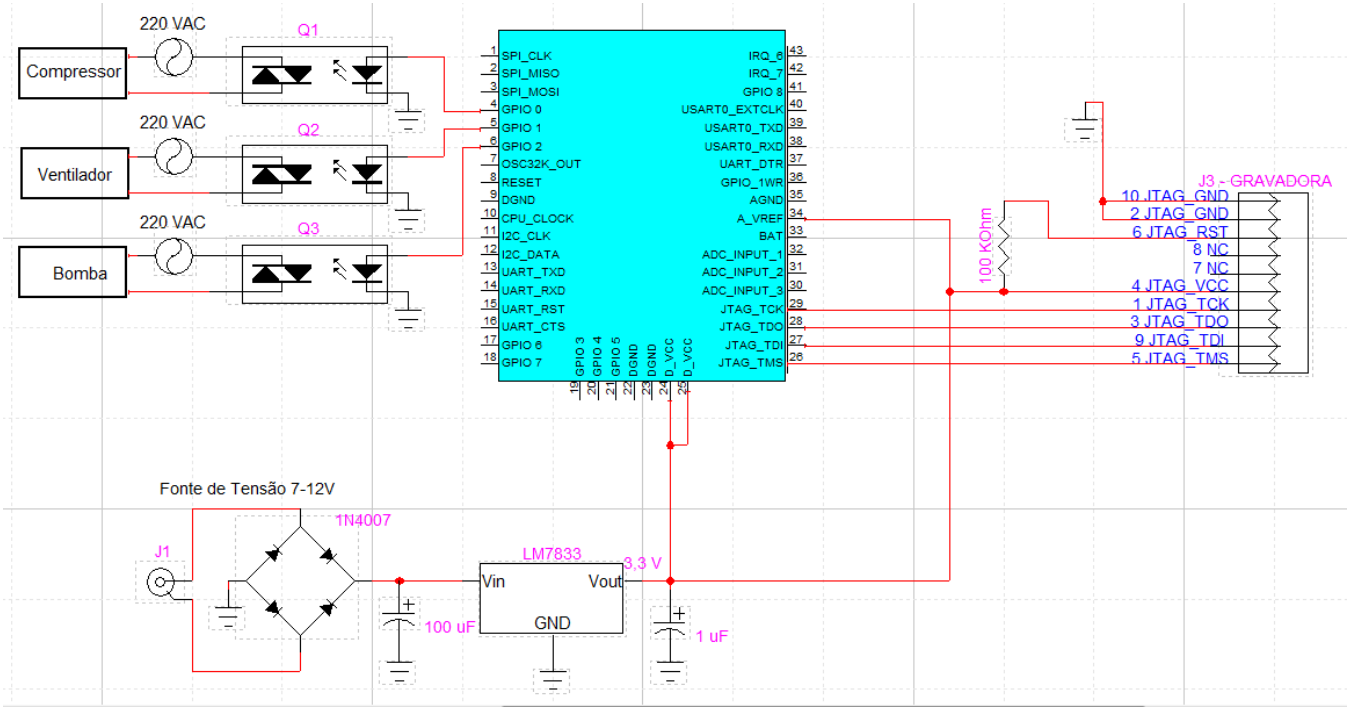


Figura 3.18 – Esquema Elétrico do Módulo Atuador

3.4.3 Módulo Interno

O módulo interno (Figura 3.19) foi desenvolvido para mensurar os valores de temperatura e umidade após o sistema evaporativo e após o sistema de refrigeração, além do valor da temperatura radiante.

Este módulo disponibiliza portas I/O's e ADC em bornes para se fazer a alimentação dos dois sensores SHT-71 e duas I/O's para cada sensor de temperatura e umidade, além da entrada para o sinal de *output* do sensor de Temperatura Radiante. A alimentação do sensor de Temperatura Radiante não é feita pelo módulo, mas por um transformador de 220V AC/ 24V AC individualizado para tal fim. Apesar de existir um transformador no módulo interno, este se encontra somente para se fazer a alimentação do módulo como um todo e não do sensor de temperatura radiante. Isto ocorreu devido a um erro de projeto inicial.

Figura 3.19 – Módulo Interno

O esquema elétrico deste módulo pode ser observado na figura 3.20 e sua programação no Anexo III.

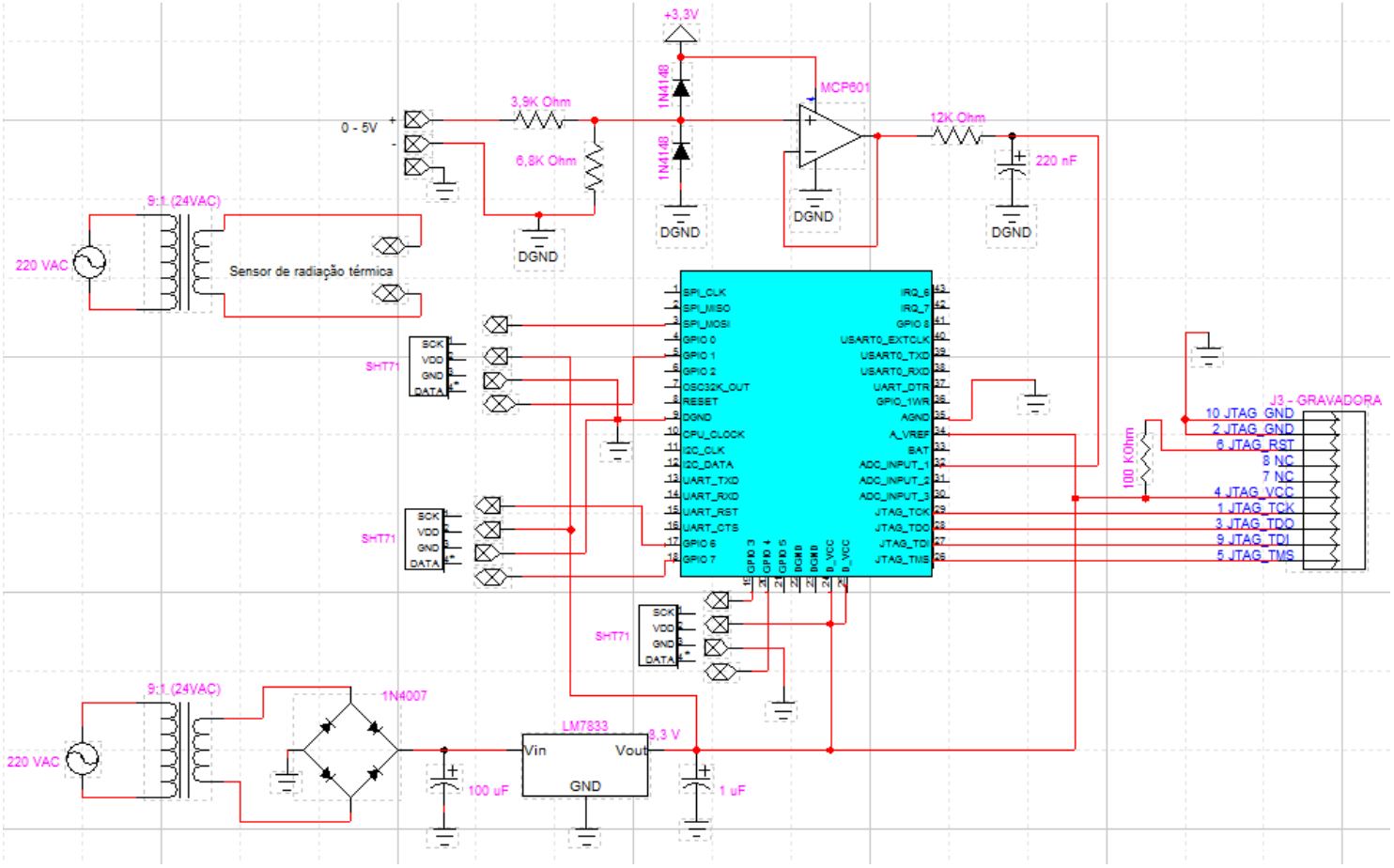


Figura 3.20 – Esquema Elétrico do Módulo Interno

3.4.4 Módulo Móvel

Este módulo simula a posição real de onde se mensurar o conforto térmico. Ele possui além dos bornes de alimentação dos sensores, dois bornes de I/O's disponibilizados para a medição da temperatura e umidade e uma porta AD, para conversão do sinal de output do anemômetro (Figura 3.21).

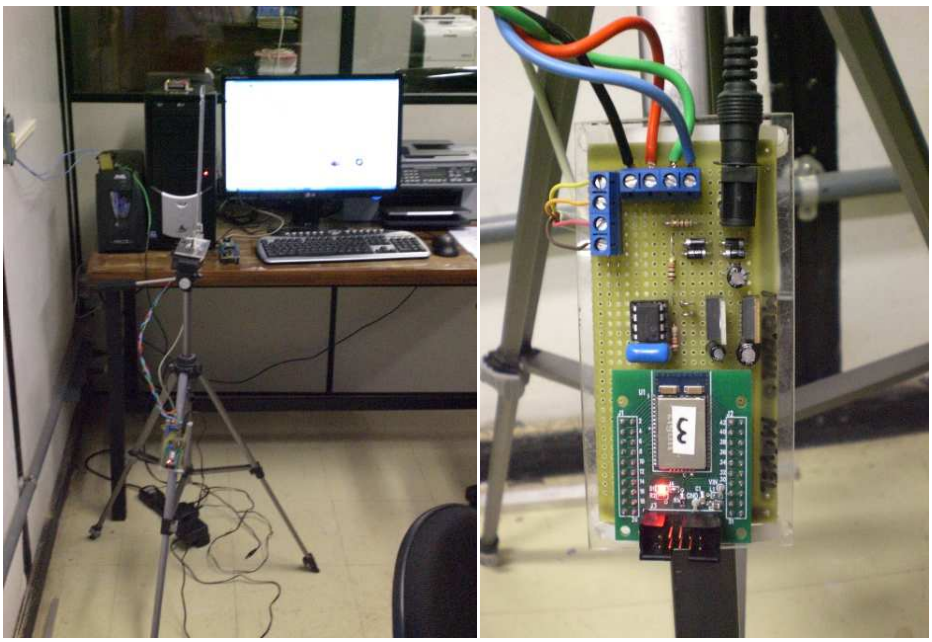


Figura 3.21 – Módulo Móvel

A programação do módulo móvel se distingue da programação do módulo interno pela quantidade do sensor de temperatura e umidade. E seu esquema elétrico pode ser visualizado na figura 3.22.

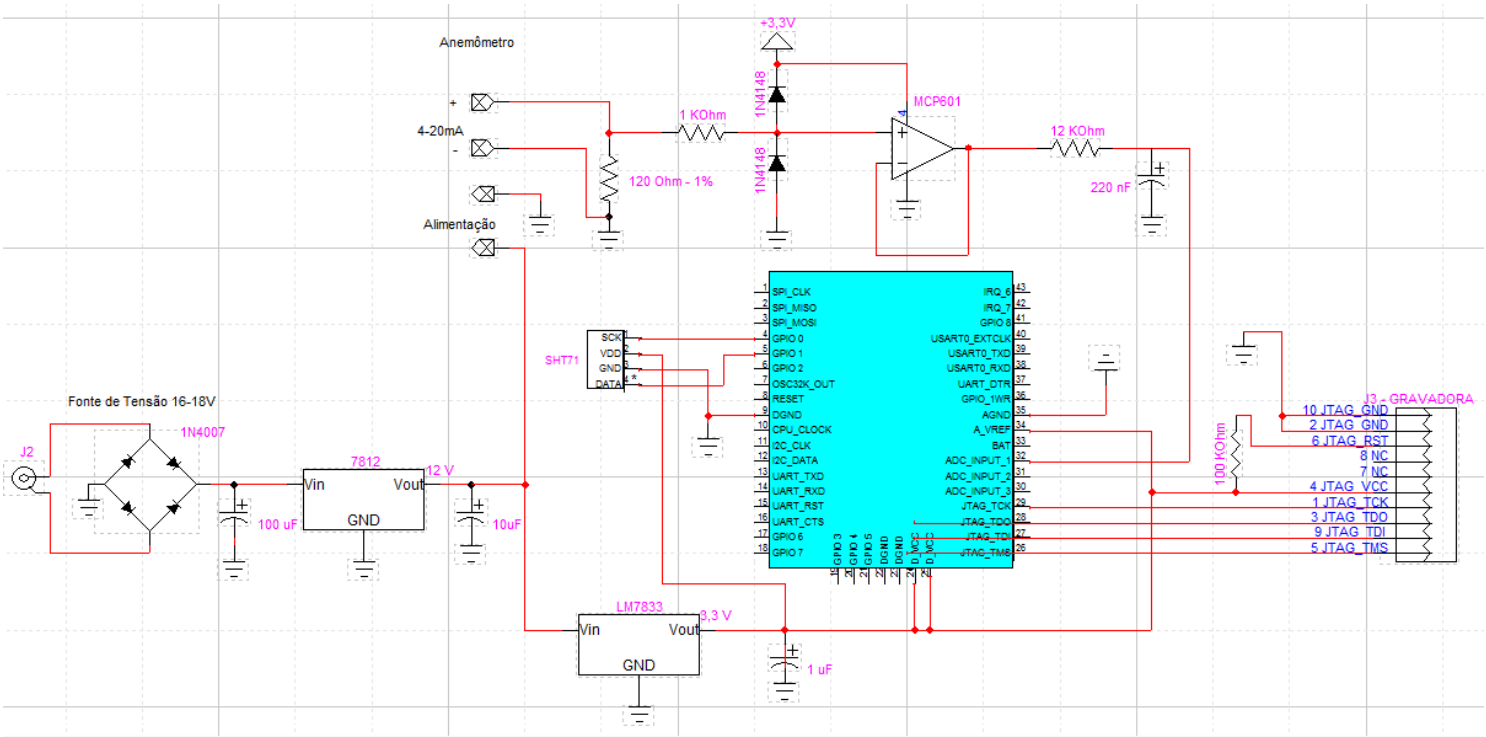


Figura 3.22 – Esquemático Elétrico do Módulo Móvel

3.4.5 Módulo Externo

O módulo externo mensura as variáveis externas ao prédio do LAVSI. Ou seja, a temperatura e umidade externa e o valor da radiação solar. Apesar dos sensores se encontrarem externamente, o módulo se localiza no ambiente interno (Figura 2.23).

Figura 2.23 – Módulo Externo

Este módulo possui programação tal qual o módulo móvel e, portanto, também só se diferencia da programação do módulo interno na quantidade de rotinas de aquisição dos dados de um sensor de temperatura e umidade.

Apesar de que o outro sensor que contém no módulo móvel ser o piranômetro, este também possui sinal *output* de 4 – 20mA, assim como o anemômetro, mensurado no módulo móvel, o que também não o diferencia em seu esquemático, além do fato que o piranômetro só necessita de dois bornes, um para alimentação e outro para o seu sinal de *output* (Figura 3.24).

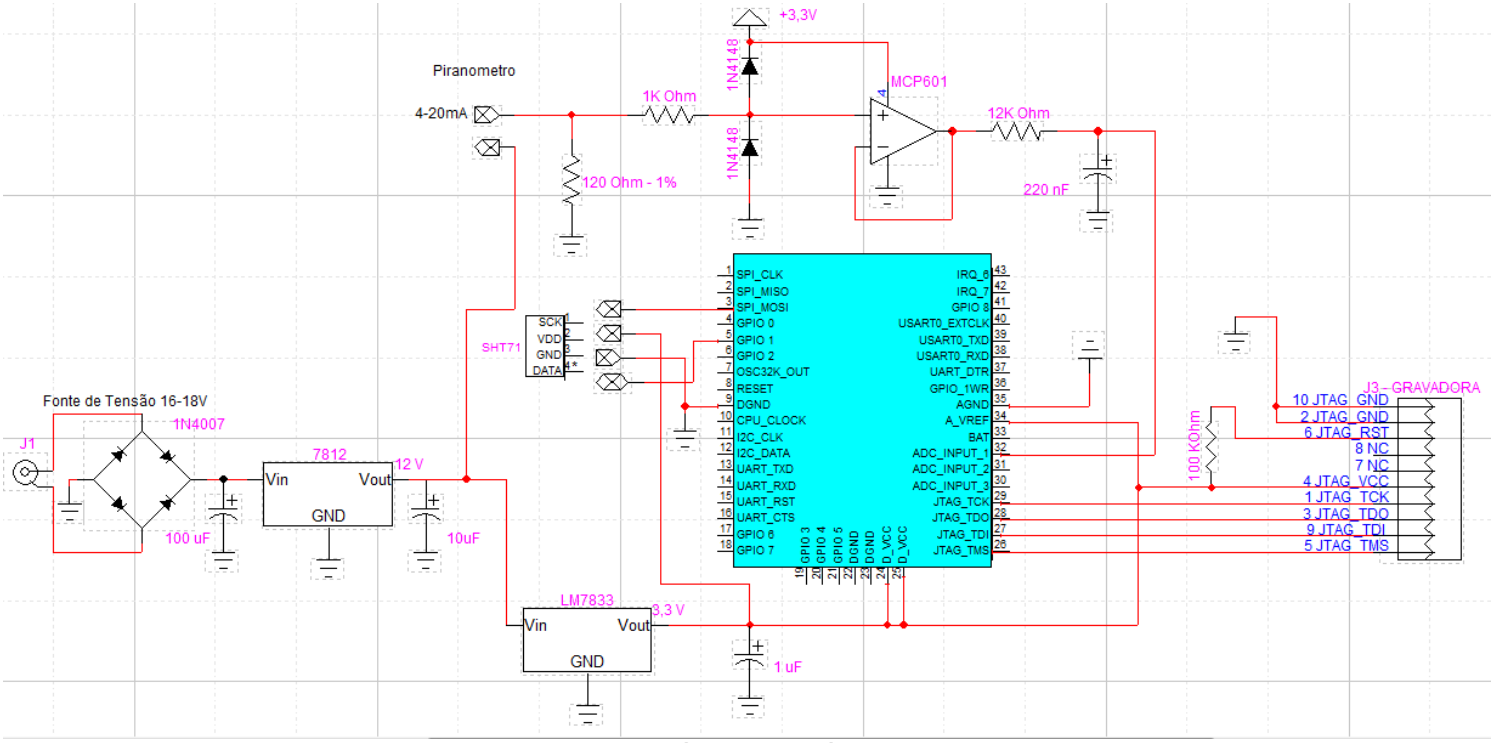


Figura 3.24 – Esquema elétrico do Módulo Externo.

3.5 Estratégia de Controle

O módulo de controle, módulo atuador, tem a função de acionamento do aparelho de ar condicionado, fazendo o chaveamento para o controle da temperatura e umidade, os módulos sensores são responsáveis pela aquisição das grandezas das variáveis do experimento. O supervisório coordena o fluxo de dados entre os módulos sensores e o módulo atuador, além da execução da rotina de controle, geração de sinais e transmissão de dados aos módulos sensores e atuador.

3.5.1 Controle liga-desliga com uso de histerese

Uma forma de ser fazer controle em alguns ramos comerciais é por meio de controle liga-desliga com histerese. Este é o controle mais fácil de implementação em simulação e em tempo real.

Num sistema de controle do tipo liga-desliga, também chamado de controle de duas posições, o elemento atuante funciona basicamente com dois níveis fixos: liga e desliga. Neste tipo de controle, o sinal de saída do controlador assume uma determinada posição física, dependendo se o erro atuante, associado ao sistema, for positivo ou negativo [7].

O sinal de realimentação é verificado pelo controlador que faz uma comparação com o seu padrão de referência. Caso o valor da realimentação seja superior à referência, o

controlador deve atuar de maneira a levar o sinal de saída ao seu valor mínimo. Caso contrário, a atuação deve elevar o sinal de saída para o valor máximo (Figura 3.25).

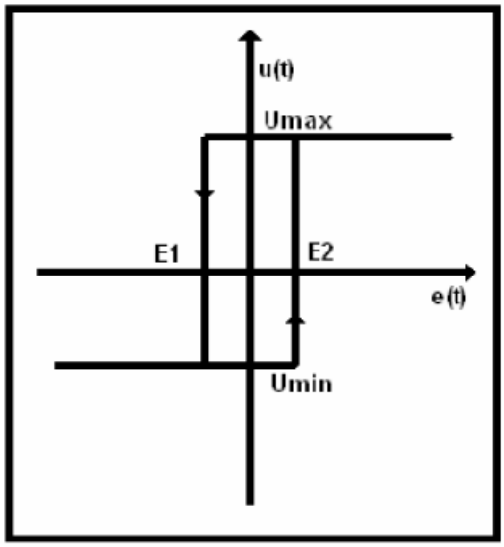


Figura 3.25 - Resposta por Histerese

Este processo caracteriza um controle liga-desliga com uso de histerese, a qual é usada para evitar chaveamentos excessivos do controlador e do atuador.

3.5.2 Controle implementado

O controle desenvolvido neste projeto foi o Liga/Desliga com histerese em relação aos parâmetros de temperatura e umidade.

Isto é, os valores mensurados de temperatura e umidade pelo módulo móvel são comparados aos valores de *set point* e histerese. *Set point* este, inicializado em 23℃ para a temperatura e de 60% para a umidade. Porém estes valores podem ser alterados no decorrer do funcionamento do sistema, através do software supervisório.

A lógica de controle se encontra no software supervisório. Este, envia o sinal de controle ao coordenador, que tem como destinatário o módulo atuador.

3.6 Software Supervisório

No intuito de se gerenciar o sistema de automação desenvolvido neste projeto, utilizou-se como base o aplicativo de supervisão implementado em linguagem Visual Basic no trabalho “Redes de sensores sem fio para automação predial com módulos MeshBeans” [16].

Para alcançar tais fins, realizaram-se adaptações no código e na interface gráfica em relação ao apresentado em [16], em que esta pode ser observada na figura abaixo.

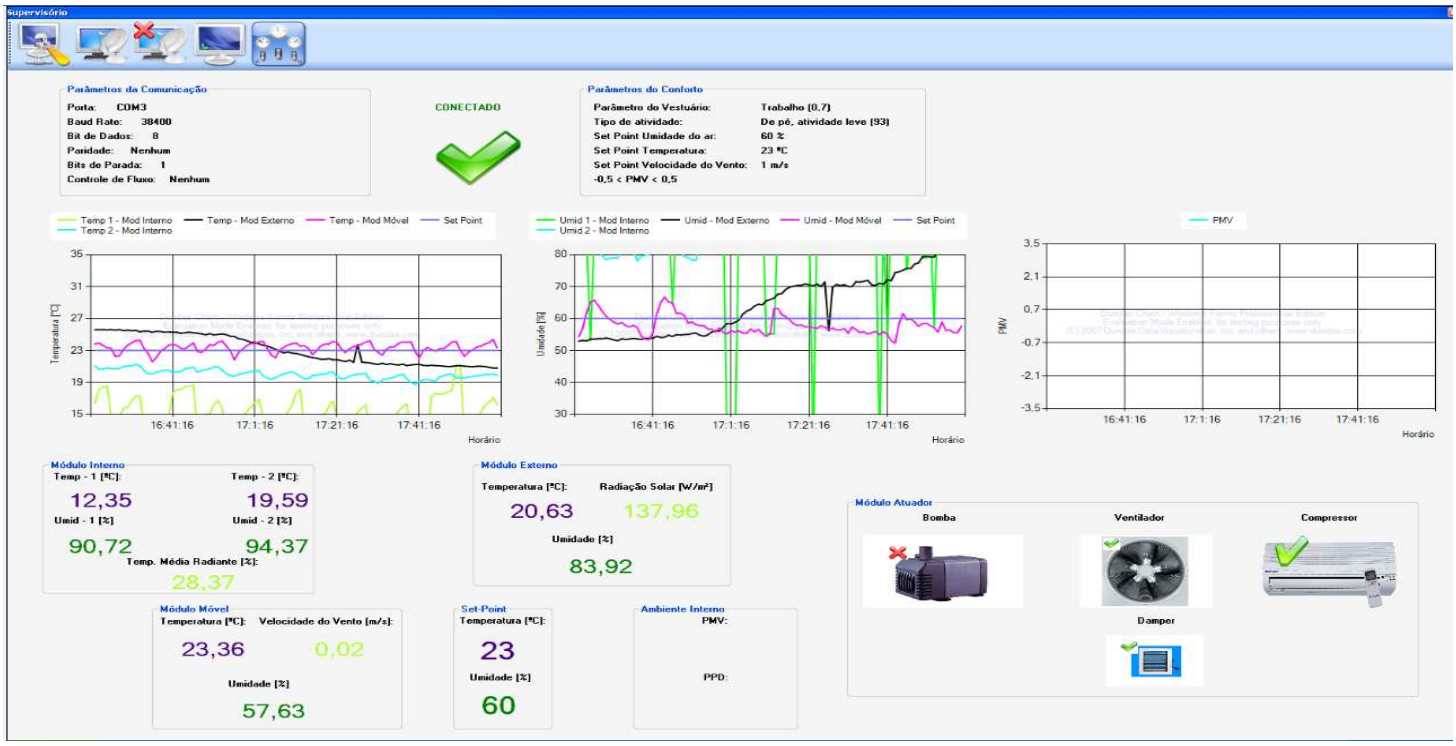


Figura 3.27 – Tela principal do Software supervisório

Assim, o software supervisório e a aquisição de dados são utilizados para monitorar e supervisionar as variáveis e os dispositivos. Além da função de administrar a rede, supervisionar e monitorar as variáveis coletadas e enviadas pelo módulo coordenador, o supervisório também possui a função de processar e enviar os comandos necessários para o controle do sistema ao módulo atuador.

A linguagem adotada Visual Basic é dirigida por eventos (*Event Driven*), e possui também um ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*) totalmente gráfico, facilitando enormemente a construção da interface das aplicações (GUI - *Graphical User Interface*) [13].

A interface gráfica elaborada é de fácil uso, sendo que se implementou cinco botões, que podem ser observados na figura 3.28, que possuem as seguintes funcionalidades, respectivamente:

- Configuração da comunicação;
- Conectar;
- Desconectar;
- Comandos e Dados;
- Parâmetros do Conforto Térmico.

Figura 3.28 - Botões do Supervisório

A comunicação entre o módulo coordenador e o computador é de forma serial RS-232, em que estes dados são coletados pelo software e assim, amostrados e analisados. Para se inicializar a comunicação entre a porta serial e o software, deve-se, somente, selecionar o número da porta COM, figura 3.29, que pode variar de computador, para computador, situado no botão “Configuração da Comunicação” e apertar o botão “Conectar”.

Figura 3.29 – Configuração da Comunicação

Porém, ainda existe a possibilidade de se configurar parâmetros analisados para se determinar o conforto térmico, como o tipo de vestimenta e o tipo de atividade, ou mesmo, mudar o *set point* de temperatura ou da umidade, utilizada para se fazer o controle do sistema como um todo (Figura 3.30).

Figura 3.30 – Parâmetros do Conforto Térmico

Apesar de que nessa tela há a possibilidade de se mudar o *set point* da temperatura e umidade como variável de entrada de controle, não se pode fazer o mesmo com as variáveis: umidade relativa do ar, em que somente, pode ter seu valor inicializado diferente do padrão e, posteriormente, é amostrado de acordo com os valores lidos dos sensores através da aquisição de dados.

Caso haja a escolha de configurar os parâmetros de conforto térmico, este deve ser feito antes de se apertar o botão “Conectar” (Figura 3.28).

Os dados são enviados pelo módulo coordenador pela porta serial RS-232, assim que o módulo coordenador receba algum dado diferente da rede de sensores ou do módulo atuador e gera 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 3.31 [16]. Deve-se salientar que os dados amostrados nesta tela são os mesmos dados enviados pelo módulo coordenador pela porta serial, portanto não há nenhum tratamento nos seus valores e as suas atualizações estão de acordo com os dados recebidos dos outros módulos.

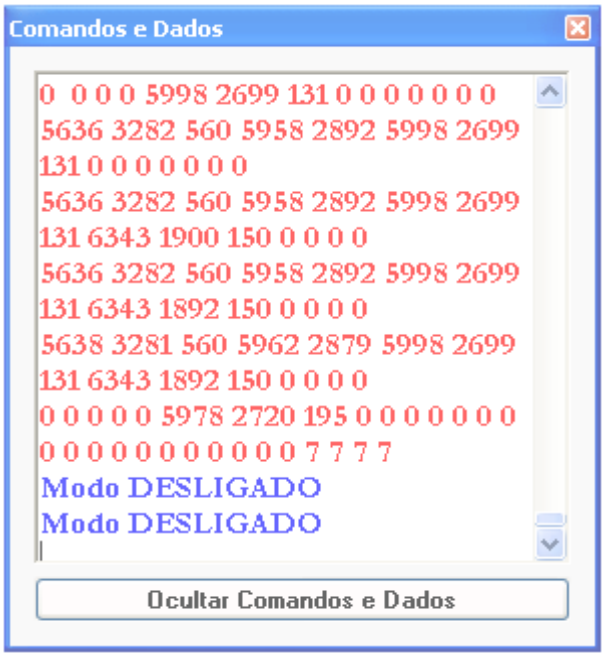


Figura 3.31 – Comandos e Dados

Os dados recebidos, através da rede de sensores pelo coordenador são enviados para a porta serial na seqüência:

- Temperatura após Refrigeração – Módulo Interno;
- Umidade após Refrigeração – Módulo Interno;
- Temperatura Média Radiante – Módulo Interno;
- Temperatura após Evaporativo – Módulo Interno;
- Umidade após Evaporativo – Módulo Interno;
- Temperatura – Modulo Externo;
- Umidade – Módulo Externo;
- Radiação Solar – Módulo Externo;
- Temperatura – Módulo Móvel;
- Umidade – Módulo Móvel;
- Velocidade do Vento – Módulo Móvel;
- Status Bomba – Módulo Atuador;
- Status Ventilador – Módulo Atuador;
- Status Compressor – Módulo Atuador;
- Status *Damper* – Módulo Atuador.

Através do *status* dos atuadores, enviam-se à tela “Comandos e Dados” os modos de operação do sistema híbrido de climatização da sala de reunião do Laboratório LAVSI. Em que estes modos podem ser observados na tabela abaixo.

Tabela 3.5 - Modos de operação do Sistema Híbrido.

Modo de Operação		Situação dos Equipamentos			
Símbolo	Descrição	Ventilador	Bomba D'água	Compressor	Damper
DESL	**Desligado**	Desligado	Desligado	Desligado	Desligado
EVAP	**Evaporativo**	Habilitado	Habilitado	Desligado	Desligado
REF	**Refrigeração**	Habilitado	Desligado	Habilitado	Habilitado

O modo de operação em que o sistema habilita evaporativo e refrigeração, ao mesmo tempo não está sendo realizado. Igualmente acontece com o modo em que só o ventilador está habilitado, devido a implementações relativas ao próprio ar condicionado híbrido e o seu consumo. Portanto o sistema estará atuando ou no modo refrigeração, ou no modo evaporativo, ou desligado. Sendo assim, apesar do nome “híbrido” o sistema não está atuando nos dois modos de operação, evaporativo e refrigeração, concomitantemente.

4 Dados Experimentais e Análise

O experimento foi realizado durante um período e 11 horas, de 9:28 até as 20:30 no dia 24/08/09. Neste dia o clima se apresentava nublado e chuvoso, o que aumentou a umidade e diminuiu a temperatura externa, favorecendo o controle do sistema de ar condicionado híbrido. Constatou-se uma grande perda de dados e leitura errada de dados durante o experimento. As perdas são devido a não robustez da rede, onde foi claro durante o período de experimentos um conflito de entrega de pacotes entre os dispositivos ZigBit coletores de dados e o dispositivo ZigBit coordenador. A leitura incorreta dos dados ocorre durante a transmissão e armazenamento dos dados do módulo coordenador para o banco de dados do software supervisório. Os dados eram alterados e aprestavam valores incoerentes. O tempo crítico de resoluções de tarefas também apresenta uma dificuldade para a coleta de dados, já que o número de tarefas aumentava muito com o grande volume de pacotes recebidos pelo coordenador, não atendendo a limitação de tempo para uma tarefa.

Os dados coletados são referentes às variáveis necessárias para o controle da qualidade de conforto térmico: temperatura e umidade externa à sala, temperatura e umidade pós-sistema evaporativo, temperatura e umidade pós-sistema de refrigeração e temperatura e umidade representativo de uma pessoa. Além de variáveis para controle de temperatura e umidade também são coletados dados de média radiante, radiação solar e velocidade do ar. Estes dados são utilizados para o levantamento do sistema térmico da sala de reunião e possibilitar a modelagem matemática desse sistema.

O gráfico da figura 4.1 mostra os dados de temperatura coletados diretamente do supervisório sem filtro. Pode-se notar uma qualidade muito ruim dos dados. Com a observação de várias horas de coleta foi concluído que o problema não estava na rede ZigBee, pois ao utilizar o software de terminal serial para a visualização direta da porta COM foi constatado que os dados eram enviados corretamente, portanto havia uma grave falha do supervisório na recepção e processamento dos dados do módulo coordenador. O software não era capaz de garantir a conversão dos valores enviados pelo módulo coordenador em valores das variáveis reais, já que para poupar tempo de execução dos módulos ZigBits, optou-se por enviar os valores coletados em sua forma lida diretamente das conversões A/D e da leitura do SHT-71, que vem em valores que ainda necessitam de processamento. Para atenuar este erro foi elaborado um filtro que descarta significativamente os dados errados.

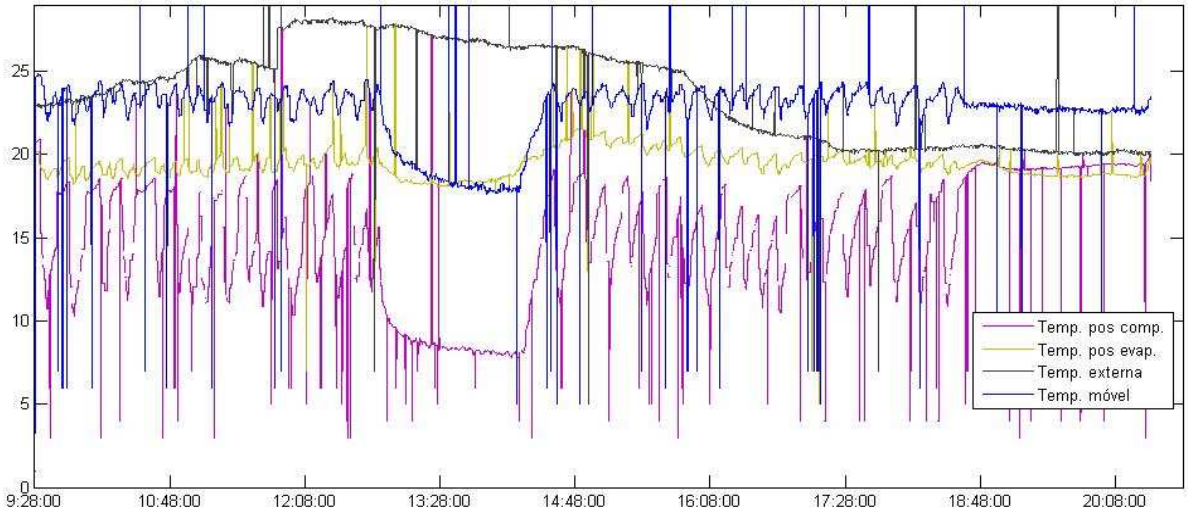


Figura 4.1 – Gráfico dos dados de temperatura coletados, sem filtragem.

O filtro descarta os valores considerados incoerentes, como os muito baixos ou os muito altos ou ainda com uma variação muito grande de um dado qualquer “x” para seu antecessor “x-1”. Quando o dado é descartado, em seu lugar é armazenado o valor de seu antecessor. O filtro foi utilizado para cada gráfico de maneira iterativa, através do método de tentativa e erro, para que não se tornasse nem muito rigoroso (onde dados verdadeiros fossem descartados) e nem muito brando (onde dados errados fossem aceitos). Segue a função do filtro escrita no software MatLab:

```
function SR = TG(array,tamanho,min,max,dif)
  for i=1:tamanho,
  if i>1
   if array(i)< min,
       array(i)=array(i-1);
   end
    if array(i) > max,
        array(i)=array(i-1);
    end
    if abs(array(i)- array(i-1))>dif,
        array(i)=array(i-1);
    end
```

```
    end
  end
  SR = array;
end
```

Como resultados desta filtragem obtiveram-se gráficos satisfatórios para uma análise qualitativa correta.

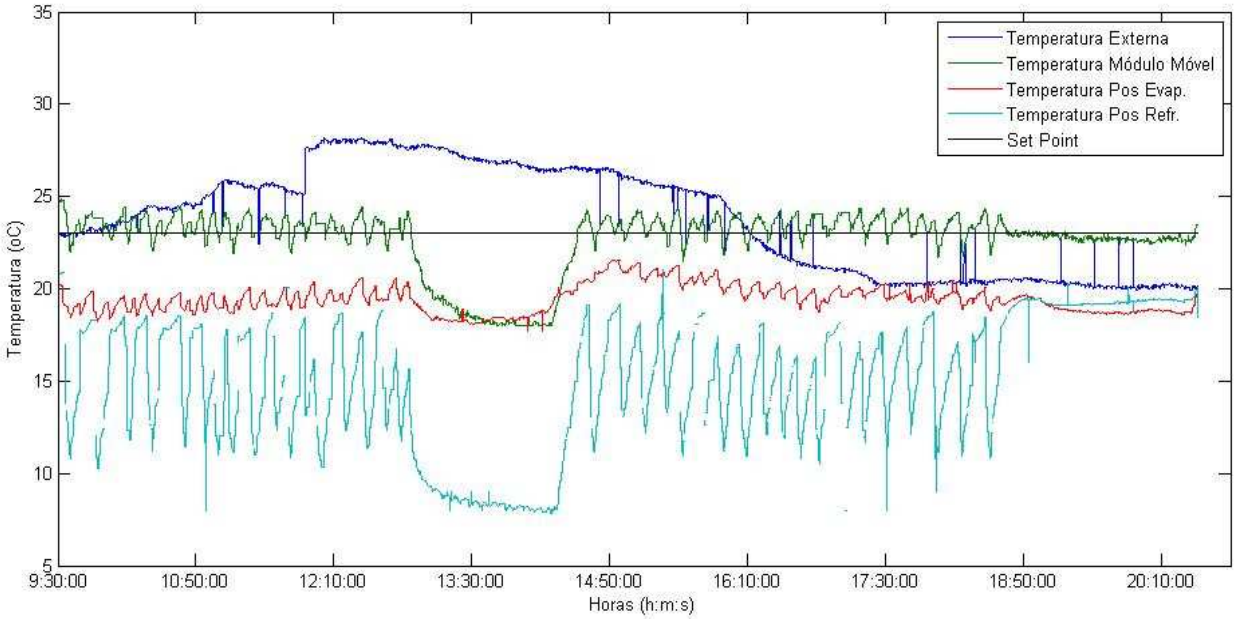


Figura 4.2 – Gráfico referente às temperaturas mensuradas

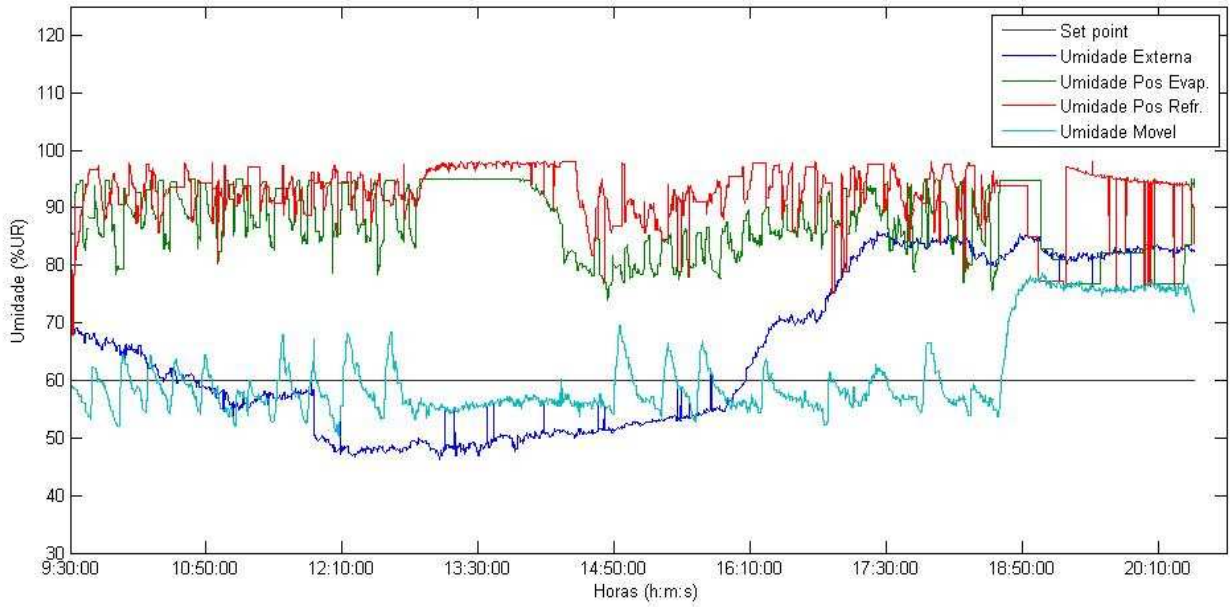


Figura 4.3 – Gráfico referente às umidades mensuradas

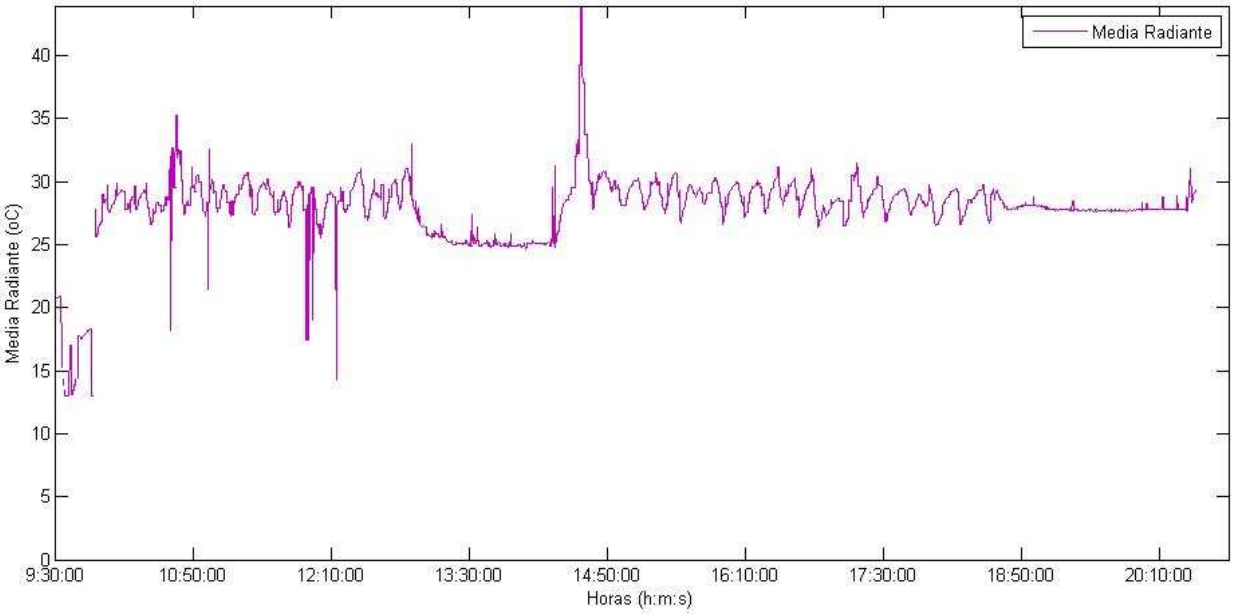


Figura 4.4 – Gráfico dos dados referentes à Temperatura Radiante

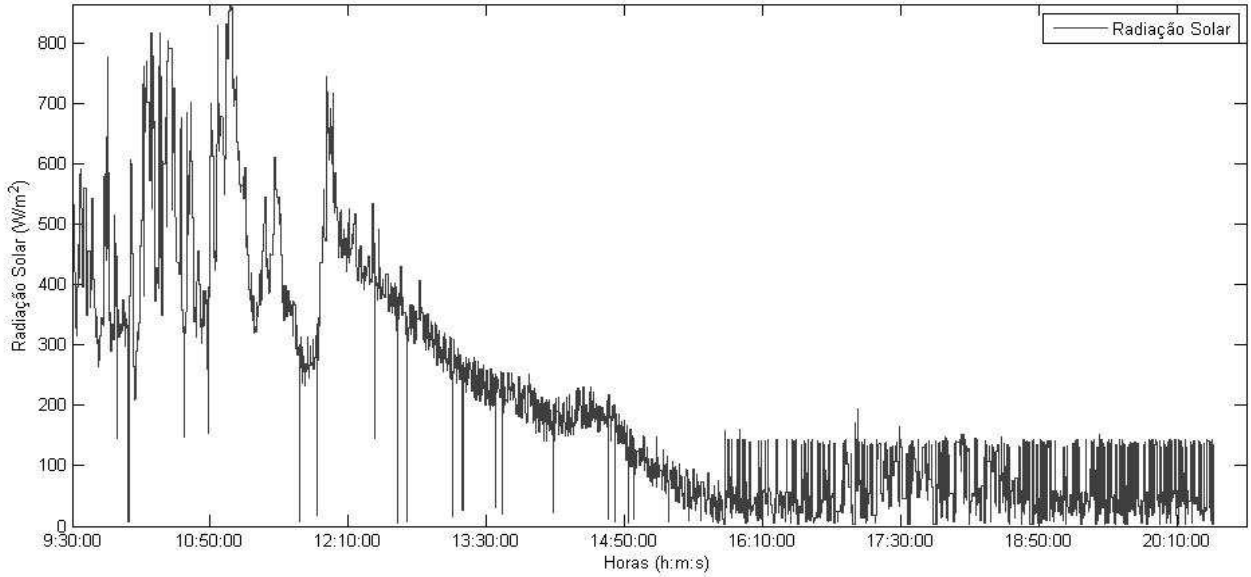


Figura 4.5 – Gráfico dos dados referentes à Radiação Solar

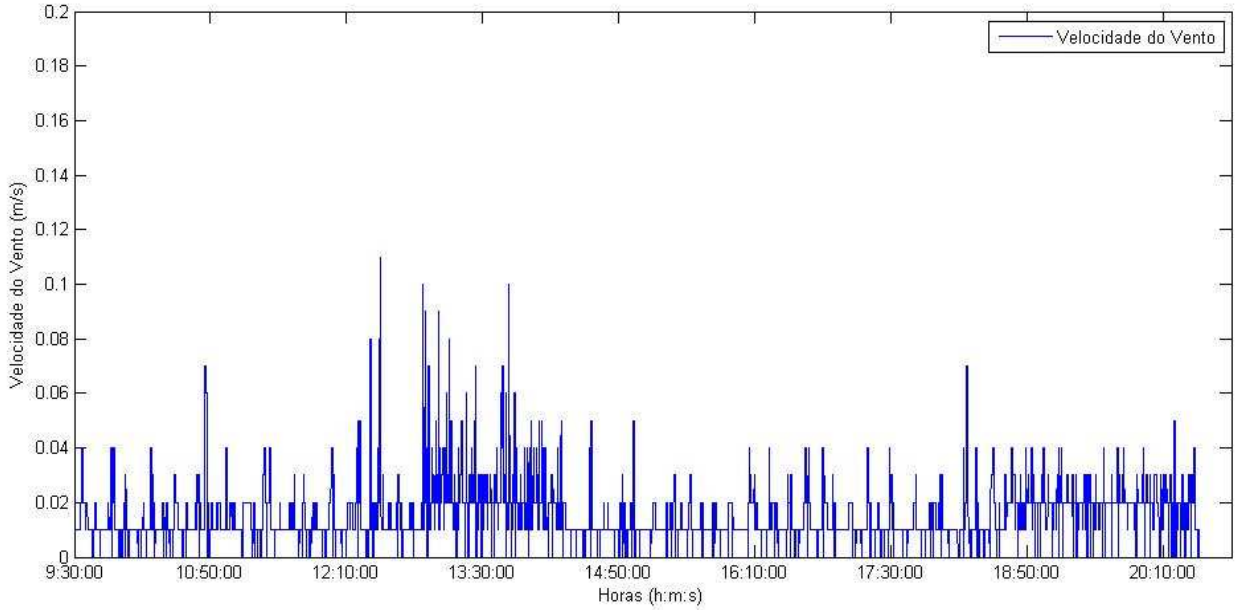


Figura 4.6 - Gráfico dos dados referentes à Velocidade do Vento

A lógica de atuação do sistema foi baseada na medição de temperatura e umidade medidas pelo módulo móvel. Foi estabelecido como *set point* para o controle de conforto térmico uma temperatura de 23 graus e uma umidade de 60%. Foi utilizada uma histerese de 0,5 ℃ para a temperatura e 5% UR para a umidade. A lógica deve garantir primeiramente o controle de temperatura e depois de assegurada a temperatura deve ser controlada a umidade.

A mediação do PMV é feita utilizando também como referência a temperatura e umidade coletadas pelo modulo móvel, assim como a velocidade do ar onde o sensor também esta junto ao módulo móvel. Ainda é utilizado o sensor de média radiante. Quanto aos valores de esforço físico e vestimenta, estes valores são inicializados no software supervisório. O índice de PMV não foi implementado corretamente. O seu valor não esta coerente devido ao grande número de falhas na recepção dos dados.

Os gráficos das figuras de 4.2 a 4.6 mostram o resultado das medições realizadas. Os valores estão coerentes de acordo com o esperado. Os sensores apresentaram bons resultados e um funcionamento adequado.

Nos gráficos das figuras 4.2 e 4.3 observa-se que apesar do controle liga-desliga com uso de histerese ser uma das lógicas de controle mais simples ele também é muito eficaz na manutenção dos valores controlados na margem desejada. Margem esta regulada através da largura de histerese.

Também é notado no gráfico dois pontos de falha do sistema, um ocorrido por volta das 13 horas e outro por volta das 18 horas e 40 minutos. Na primeira falha o sistema ficou em seu modo de refrigeração constantemente. Já na segunda falha o sistema ficou em seu modo evaporativo constantemente. Estas falhas estão relacionadas a rede sem fio dos módulos

ZigBit. O módulo atuador deixa de receber qualquer mensagem enviada a ele e deve ser reiniciado para voltar ao seu estado normal.

Através da análise dos períodos de falha pode ser observado que a atuação do ar condicionado é super dimensionada à sala, tanto no modo de refrigeração, quanto no modo evaporativo. Dessa forma o tempo de resposta do sistema é rápido. Nos gráficos das figuras 4.2 e 4.3, podem-se notar algumas características do sistema. O sistema, quanto à temperatura, apresenta comportamento de um sistema de 1ª ordem e apresenta erro em estado estacionário, que provavelmente é ocasionado pela interferência térmica externa à sala. O sistema também apresenta saturação, tanto na temperatura quanto na umidade.

O consumo de energia elétrica no período analisado foi de aproximadamente 9,7 KW, porém pelas condições climáticas e curto período de analise não se pode afirmar nada quanto a eficiência energética do sistema.

5 Conclusões e Expectativas Futuras

A comunicação sem fio ZigBee utilizando os módulos ZigBits mostrou, esporadicamente, perdas de pacotes que ocorriam pela falha de comunicação no tempo esperado entre o coordenador e dispositivos de aquisição de dados e entre o coordenador e o dispositivo atuador. A implementação de um protocolo de comunicação, e.g. BACNet deverá identificar tais situações e solicitar o reenvio dos dados perdidos garantindo uma maior confiabilidade no tráfego dos dados.

O uso do software supervisório implementado em *visual basic* também não se mostrou robusto. Ele não garante o processamento em tempo hábil das informações recebidas pelo coordenador da rede ZigBee. Houve neste ponto perda de dados. O uso da ferramenta excel do pacote Office para o armazenamento dos dados, apesar de prático não se mostrou uma boa solução, já que aumentou consideravelmente o tempo de processamento do software supervisório. A utilização de um software supervisório mais robusto com a utilização de um banco de dados especializado deve ser integrado ao projeto, corrigindo este ponto de falha.

Apesar dos problemas constatados, ainda foi possível um controle de temperatura e umidade adequado. A instalação dos módulos sensores e módulo atuador é uma etapa fundamental para o projeto geral, “Controle do ambiente de um edifício visando o conforto térmico humano e economia de energia para sistemas de refrigeração”, abrindo possibilidades para um controle mais robusto e complexo, assim como um estudo mais aprofundado do sistema térmico. Além de estudos de redes sem fio sob protocolo ZigBee.

Sendo assim o projeto proposto se mostrou abrangente e envolve a maioria das áreas pertinentes para um engenheiro de controle e automação como automação, controle, instrumentação e sistemas computacionais. O projeto tem aplicação direta no mercado e na sociedade, criando uma ótima oportunidade para a contribuição do nosso conhecimento para melhorar a sociedade.

Sempre adotando o paradigma de economia energética e conforto térmico, o trabalho aqui exposto, procurou mostrar uma possibilidade de implementar táticas de controle convenientes para ambientes reais encontrados. Buscando uma configuração de sistema bem flexível para os diversos problemas pertinentes à automatização do controle de ambientes.

Referências Bibliográficas

[1] Bauchspiess, A. (2004). “Introdução aos Sistemas Inteligentes.” In: Aplicações em Engenharia de Redes Neurais Artificiais, Lógica Fuzzy e Sistemas Neuro-Fuzzy;

[2] Flores, J. L. O. (2009). “Sistema Híbrido de Climatização Visando Conforto Térmico e Eficiência Energética”. Dissertação de mestrado, Universidade de Brasília;

[3] Santos, R.J., Melo,G.A.F., Bauchspiess, A. Borges, G.A.. (2005) Controle Fuzzy para Racionalização de Energia;

[4] Gallo, E. A., Ribeiro, F. N., (2007). Índice de Conforto Térmico ISO7730 em Automação Predial;

[5] Ferreira Jr, P.A., (2008). Racionalização de energia em automação predial;

[6] Lamberts, R.; Xavier, A. A. de P. Conforto térmico e *stress* térmico. 111 p. Florianópolis, 2002.

[7] 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,67p.

[8] Nise, Norman S. Engenharia de Sistemas de Controle. Rio de Janeiro: LTC, 2002. 695 p.

[9] Ogata, Katsuhiko Engenharia de controle moderno; tradução Paulo Álvaro Maya; revisão técnica Fabrizio Leonardi [et al.]. – 4. ed. –São Paulo: Prentice Hall, 2003. 788p.

[10] pt.wikipedia.org/wiki/ZigBee

[11] MARKOV, D. Standards in thermal comfort. In: ANNUAL INTERNATIONAL COURSE: VENTILATION AND INDOOR CLIMATE, Sofia, 2002. P. Stankov (Ed). p. 147 – 157.

[12] www.prof2000.pt/users/eta/Amb_Termico.htm

[13] pt.wikipedia.org/wiki/Visual_Basic

[14] INDRIA, Y. *Design of an individual mobile measurement of thermal comfort.* 2006. 51 p. Tese de mestrado. Universidade de Kaiserslautern, Alemanha.

[15] 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.

[16] Azevedo, R. C. A. & Queiroz, Rede de sensores sem fio para automação predia com módulos MeshBean. Trabalho de graduação em Engenharia Elétrica, faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 81p.

[17] ATMEL CORPORATION, *BitCloud User Guide* 2009, disponível em <http://www.atmel.com/dyn/resources/prod_documents/doc8199.pdf.> , acesso 15 de agosto de 2009.

[18] 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 15 de agosto de 2009.

[19] www.yamatake.com/products/bi/ba/ss/AB-5361.pdf.

[20] www.dwyer-inst.com/Products/Product.cfm?Group_ID=519

[21] www.agsolve.com.br/pdf/artigos/sitio.pdf

[22] www.sabereletronica.com.br/secoes/leitura/367

[23] www.teletronic.ind.br/catalogo.pdf

ANEXOS

Anexo I – Cálculo do PMV;

Anexo II – Especificações do Projeto do Ar Condicionado Híbrido;

Anexo III – Código do Aplicativo ZigBit/Coordenador e ZigBit/EndDevices;

ANEXO I

Este anexo apresenta as fórmulas para o cálculo dos termos individuais da equação do índice de conforto térmico *PMV*.

AI.1 CÁLCULO DA TAXA METABÓLICA *M*

A taxa metabólica *M* é um fator que varia de acordo com o nível de atividade física que um indivíduo realiza. Quanto maior a atividade física, maior será a taxa metabólica. A Tabela AI.1 apresenta alguns valores para a taxa metabólica em função do nível de atividade física realizada:

Tabela AI.1 – Taxas metabólicas segundo a norma ISSO 7730 [4].

ATIVIDADES	TAXAS METABÓLICAS	
	[W/m^2]	[met]
Atividade sedentária (escola, residência, escritório, laboratório)	70	1,2
Atividade leve em pé (compras, laboratório, indústria leve)	93	1,6
Atividade média em pé (balconista, trabalho doméstico, trabalho em máquinas)	117	2,0
Atividade intensa	175	3,0

Observando a Tabela AI.1, podemos perceber que a taxa metabólica também pode ser expressa em termos da unidade [met]. Além disto, fazendo a proporção, concluímos que 1 *met* é igual a 58 *W/m2*.

AI.2 CÁLCULO DO CALOR PERDIDO POR DIFUSÃO *Edif*

O calor perdido por difusão de vapor de água através da pele *Edif* é calculado através da Eq. AI.1 dada a seguir:

$$Edif = 3{,}05\,(5{,}73 - 0{,}007\,M - pa) \qquad \text{(AI.1)}$$

Na Equação AI.1, temos a presença da pressão de vapor *pa*, cujo cálculo pode ser encontrado a seguir.

AI.2.1 Cálculo da pressão de vapor *pa*

A pressão de vapor *pa* pode ser calculada através da Eq. AI.2 que se encontra a seguir:

$$Pa = UR \times psat \qquad \text{(AI.2)}$$

Na Equação AI.2, *UR* é a umidade relativa do ar, que pode ser medida através de sensores de umidade conhecidos como higrômetros. O termo *psat* pode ser calculado através da fórmula encontrada na seqüência.

AI.2.2 Cálculo da pressão de saturação *psat*

A pressão de saturação *psat* deve ser calculada da seguinte maneira:

$$p_{sat} = 0{,}61078 \cdot e^{\frac{17{,}269 \cdot T_{ar}}{237{,}3 + T_{ar}}} \quad \text{(AI.3)}$$

Na Equação AI.3, *Tar* é a temperatura do ar dada em [℃], a qual pode ser medida através de diversos tipos de sensores.

AI.3 CÁLCULO DO CALOR PERDIDO POR TRANSPIRAÇÃO *Etrans*

O calor perdido por transpiração *Etrans* é calculado da seguinte maneira:

$$Etrans = 0{,}42 \times (M - 58{,}15) \quad \text{(AI.4)}$$

AI.4 CÁLCULO DO CALOR PERDIDO POR RESPIRAÇÃO LATENTE *Erl*

O calor perdido por respiração latente *Erl* é dado pela Eq. AI.5 a seguir:

$$Erl = 0{,}0173 \times M\,(5{,}87 - pa) \quad \text{(AI.5)}$$

AI.5 CÁLCULO DO CALOR PERDIDO POR RESPIRAÇÃO SENSÍVEL *Ers*

O calor perdido por respiração sensível *Erl* pode ser obtido através da seguinte equação:

$$Ers = 0{,}0014 \times M\,(34 - T) \quad \text{(AI.6)}$$

AI.6 CÁLCULO DO CALOR PERDIDO POR RADIAÇÃO *R*

O calor perdido por radiação *R* é calculado por meio da Eq. AI.6 dada a seguir:

$$R = 3{,}96 \cdot 10^{-8} \cdot f_{vest} \left[(T_{vest} + 273)^4 - (T_{rad} + 273)^4 \right] \quad \text{(AI.7)}$$

Na Equação AI.7, *fvest* é o fator de vestuário dado em [m2K/W], *Tvest* é a temperatura da vestimenta dada em [℃] e *Trad* é a temperatura radiante média dada em [℃]. Os cálculos destes termos individualmente podem ser encontrados na seqüência.

AI.6.1 Cálculo do fator de vestuário *fvest*

O fator de vestuário *fvest* é uma função com definições diferentes para dois intervalos distintos de sua variável independente. É calculado como se segue:

$$f_{vest} = 1{,}00 + 1{,}29 \cdot I_{vest} \text{ , } para\ I_{vest} < 0{,}078\ m^2K/W$$
$$f_{vest} = 1{,}05 + 0{,}645 \cdot I_{vest} \text{ , } para\ I_{vest} \geq 0{,}078\ m^2K/W \quad \text{(AI.8)}$$

Na Equação AI.8, *Ivest* é a resistência térmica da vestimenta dada em [m2K/W]. O valor de *Ivest* pode ser encontrado em tabelas como a que se encontra a seguir.

AI.6.2 Cálculo da resistência térmica da vestimenta *Ivest*

A resistência térmica da vestimenta *Ivest* varia de acordo com o tipo de roupa que uma pessoa está usando. Quanto mais peças de roupa, maior será a resistência térmica da vestimenta. A Tabela AI.2 apresenta alguns valores de *Ivest* em função do tipo de vestimenta utilizado.

Tabela AI.2 – Resistência térmica da vestimenta [2].

VESTIMENTA	RESISTÊNCIA TÉRMICA	
	[m^2K/W]	[clo]
Vestuário tropical	0,047	0,3
Vestuário leve de verão	0,078	0,5
Vestuário de trabalho	0,124	0,7
Vestuário de inverno para ambientes internos	0,155	1,0
Vestuário completo	0,233	1,5

Observando a Tabela AI.2, podemos perceber que a resistência térmica da vestimenta também pode ser expressa em termos da unidade [clo]. Além disto, vemos que *1 clo* é igual a *0,155 m2K/W*.

AI.6.3 Cálculo da temperatura da vestimenta *Tvest*

A temperatura da vestimenta *Tvest* é calculada através da Eq. AI.9 dada a seguir:

$$Tvest = 35{,}7 - 0{,}0275 \times M - 0{,}155 \times Ivest(R + C) \quad \text{(AI.9)}$$

AI.6.4 Cálculo da temperatura radiante média *Trad*

A temperatura radiante média *Trad* pode ser medida através de sensores. Para este trabalho, utilizou-se o Sensor de Radiação Térmica, marca YAMATAKE e modelo TY7321A1009.

AI.7 CÁLCULO DO CALOR PERDIDO POR CONVECÇÃO *C*

O calor perdido por convecção *C* deve ser calculado da seguinte maneira:

$$C = f_{vest} \cdot 12{,}1 \cdot \sqrt{v} \cdot (T_{vest} - T_{ar}) \qquad \text{(AI.17)}$$

Na Equação AI.17, *v* é a velocidade do vento dada em [m/s]. Esta velocidade pode ser medida por sensores conhecido como anemômetros. Como as velocidades a serem medidas durante o projeto são bem baixas (entre *0* e *0,5 m/s*), anemômetros de precisão, que utilizam como método de medida o fio quente, são os mais indicados.

AI.8 ANÁLISE DE RECURSIVIDADE

Observando as Eqs AI.7, AI.9 e AI.17, podemos notar que elas possuem recursividade entre si, em termos das variáveis *Tvest*, *R* e *C*. Uma maneira de se contornar este problema (maneira esta que foi usada neste projeto) é definir um valor inicial para uma das variáveis (no nosso caso, *Tvest*) e fazer iterações de cálculo das variáveis, sempre atualizando os valores das variáveis ao realizar novas iterações. Agindo assim, a tendência é que, após algumas iterações, as variáveis convirjam para um determinado valor.

ANEXO II

Algumas especificações do projeto do ar condicionado híbrido, instalado no LAVSI serão mostradas abaixo.

AII.1 Planta Baixa

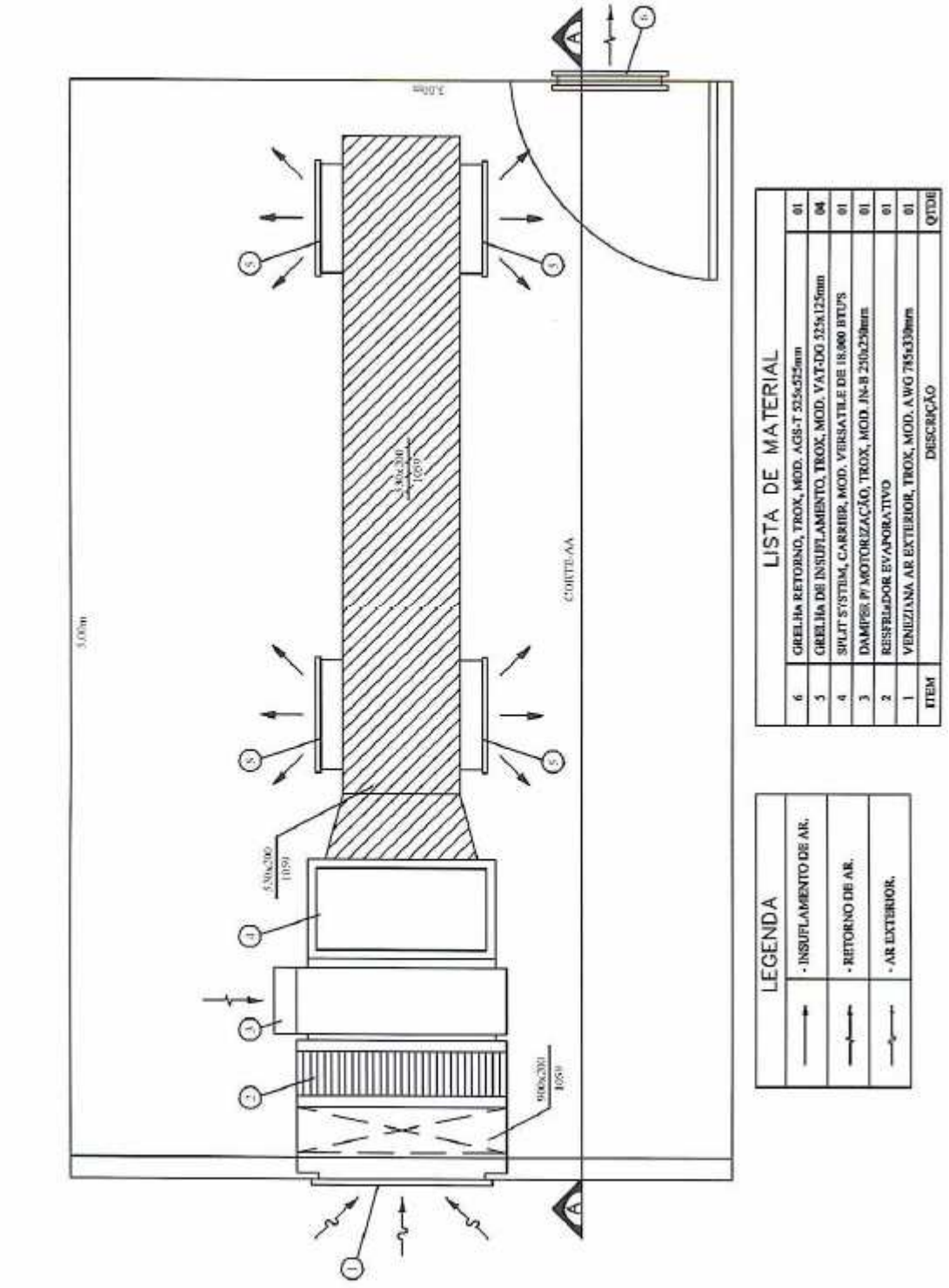


Figura AII.1 – Projeto do Sistema Híbrido do Laboratório LAVSI (Planta Baixa)

AII.2 Vista em corte

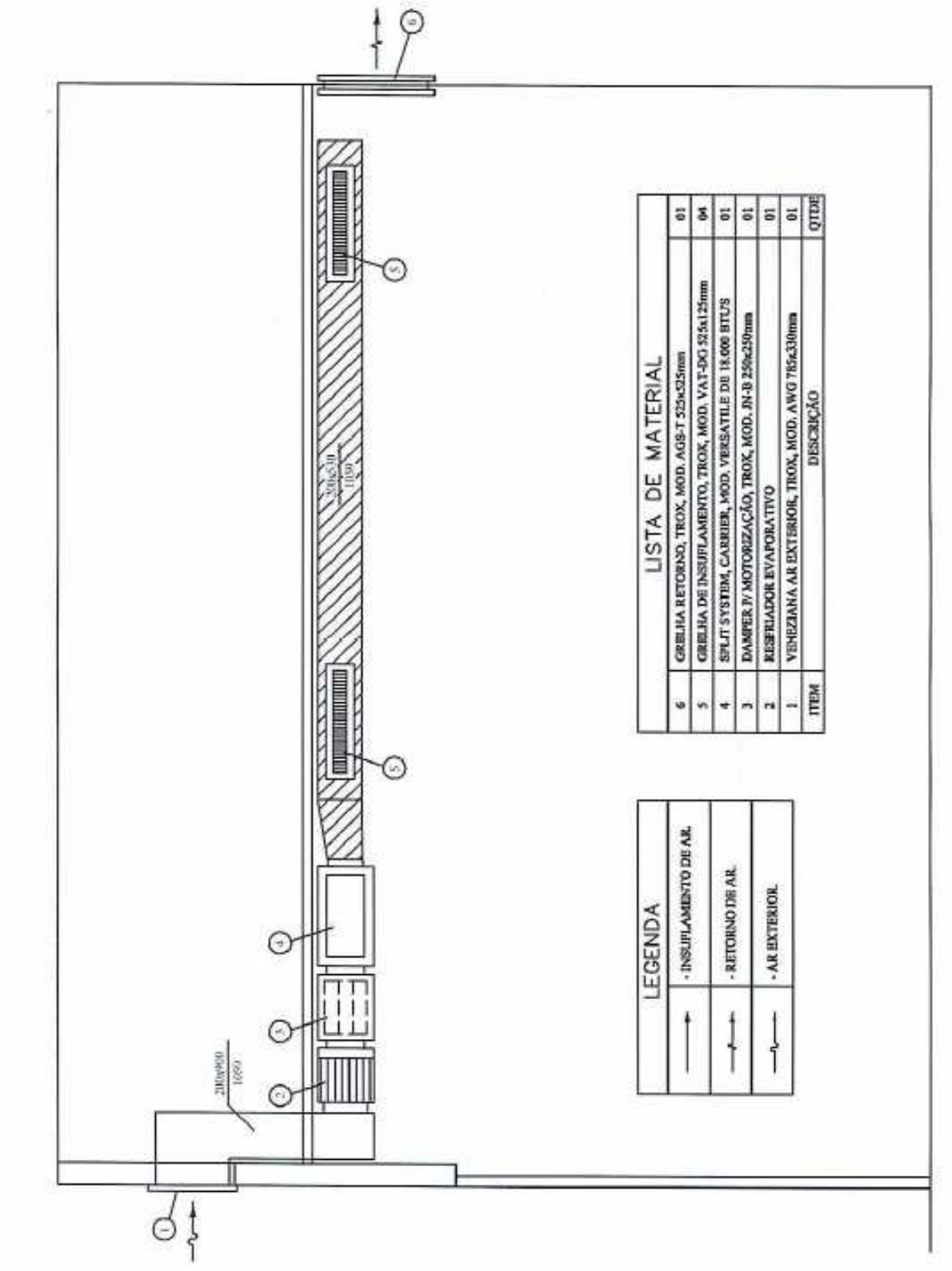


Figura AII.2 - Projeto do Sistema Híbrido do Laboratório LAVSI (Planta Baixa)

AII.3 Diagrama de Força

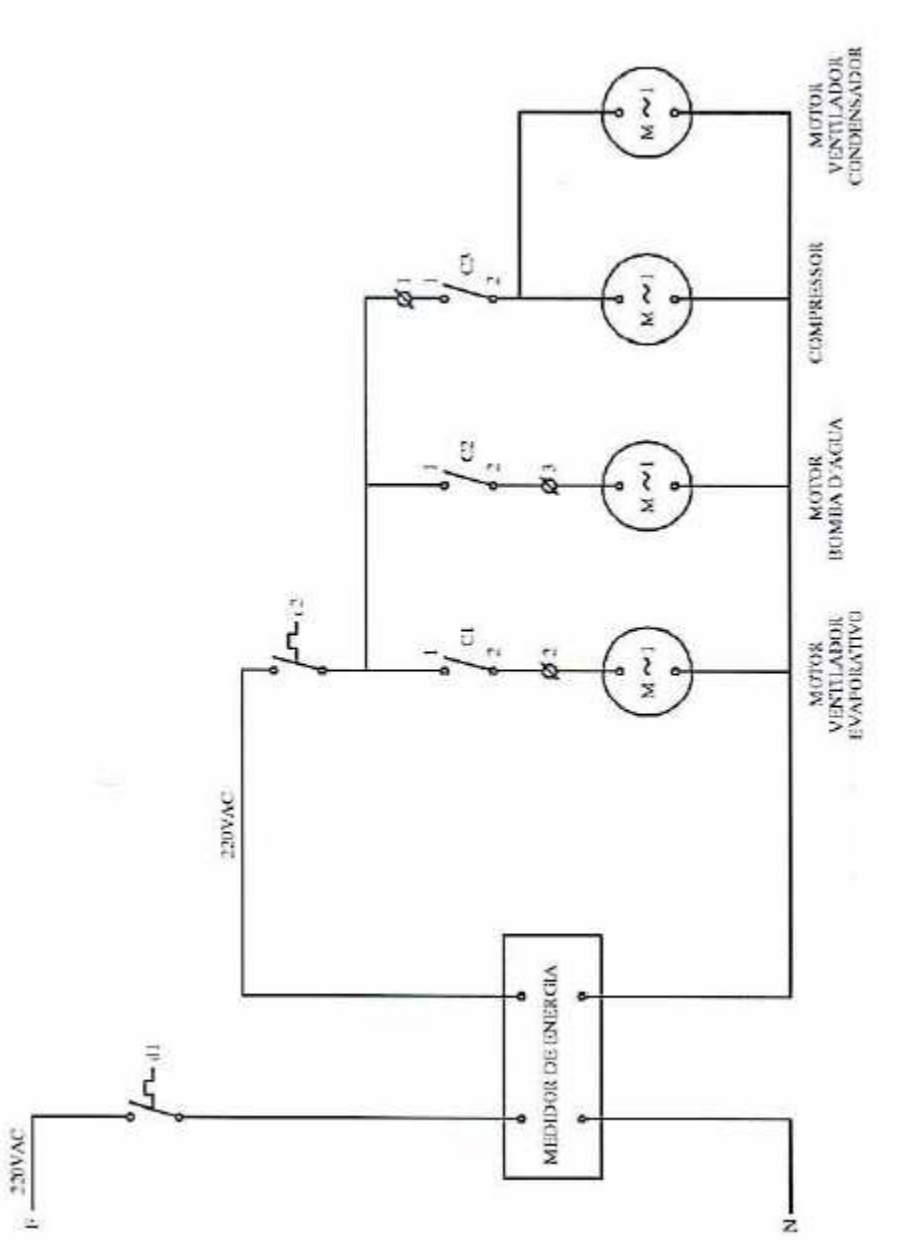


Figura AII.3 - Projeto do Sistema Híbrido do Laboratório LAVSI (Diagrama de Força)

AII.4 Diagrama de Comandos

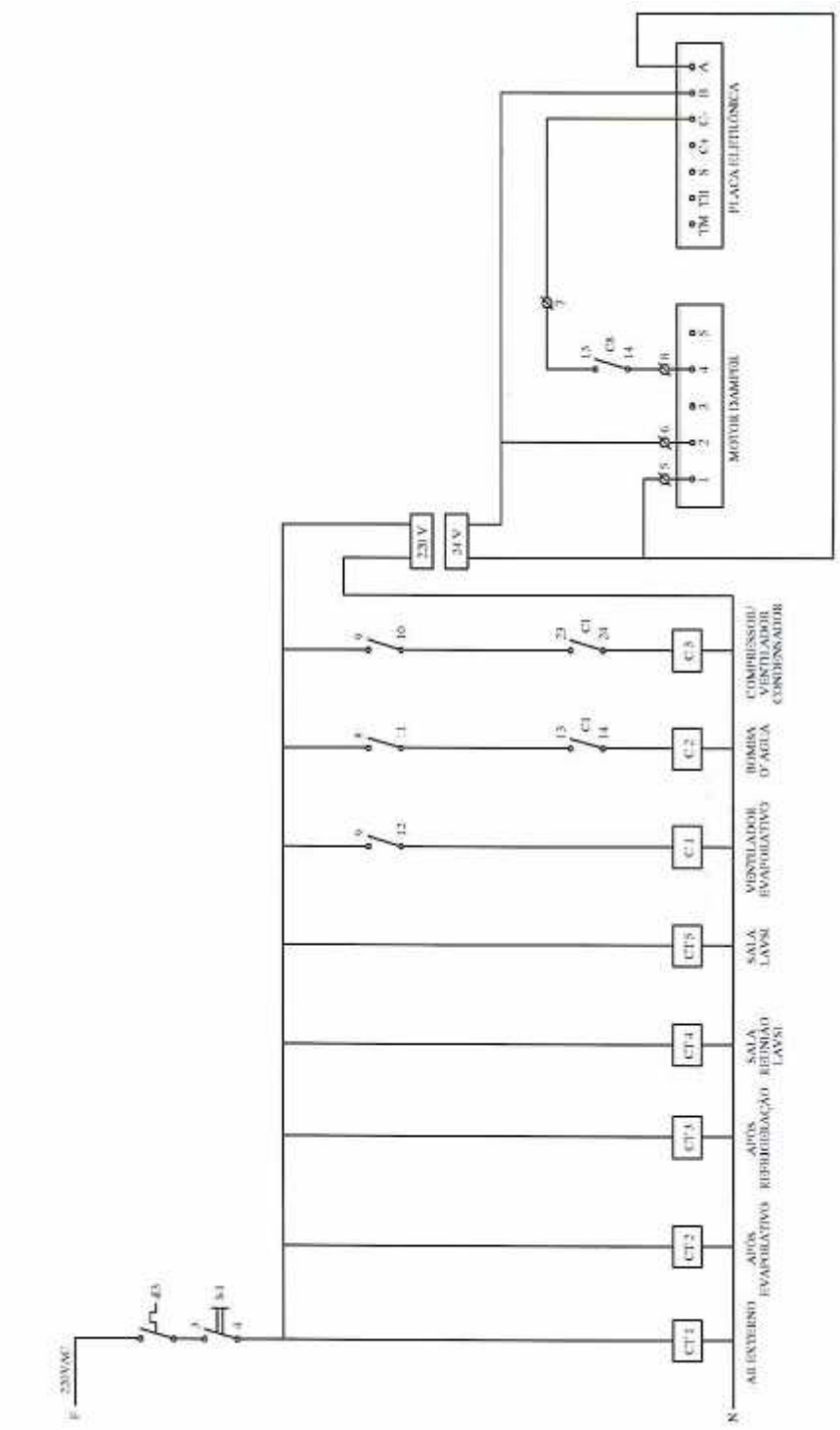


Figura AII.4 - Projeto do Sistema Híbrido do Laboratório LAVSI (Diagrama de Comando)

ANEXO III

Neste anexo se encontra o código gravado nos móulos ZigBit.

AIII.1 Código comum implementado no Módulo Coordenador.

```
/*****************************************************************************//**
  Coordenador.c
Autores:  Alexandra e Breno
*****************************************************************************/
 #include <lowpower.h>
/*****************************************************************************
  Global variables
*****************************************************************************/
AppState_t appState = APP_INITIAL_STATE;                    // Current application state
AppDeviceState_t appDeviceState = DEVICE_ACTIVE_IDLE_STATE;    // Current device state
DeviceType_t appDeviceType;
// Endpoint simple descriptor (ZDO endpoint descriptor)
SimpleDescriptor_t simpleDescriptor = {APP_ENDPOINT, APP_PROFILE_ID, 1, 1, 0, 0 , NULL, 0,
NULL};
/*****************************************************************************
  Local variables
*****************************************************************************/
static HAL_AppTimer_t networkTimer;                         // Timer indicating network start

static APS_RegisterEndpointReq_t apsRegisterEndpointReq;       // APS Register Endpoint Request
primitive (APS endpoint descriptor)

// ZDO primitives
static ZDO_StartNetworkReq_t  zdoStartNetworkReq;             // Request parameters for network start
/*****************************************************************************
  Static functions
*****************************************************************************/
#ifdef _BUTTONS_
static void buttonReleased(uint8_t button);                 // Button released handler
#endif // _BUTTONS_
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *conf);
static void initApp(void);                                  // Common application initial function
static void startNetwork(void);                             // Start Network
static void startingNetworkTimerFired(void);
static void APS_DataInd(APS_DataInd_t *ind);
```

```
/*************************************************************************************
  Implementation
*************************************************************************************/
/*************************************************************************
  Description: Application task handler
  Parameters: none.
  Returns:    none
*************************************************************************/
void APL_TaskHandler()
{
  switch (appState)
  {
    // node is in initial state
    case APP_INITIAL_STATE:               // Initial (after RESET) state
      initApp();                          // Init application
      break;

    case APP_NETWORK_JOINING_STATE:       // Network is in the joining stage
      startNetwork();                     // Start/joing network
      break;

    case APP_NETWORK_JOINED_STATE:        // Network was successfully started
#ifdef _COORDINATOR_
      if (DEVICE_TYPE_COORDINATOR == appDeviceType)
        appCoordinatorTaskHandler();
#endif // _COORDINATOR_
#ifdef _ENDDEVICE_
      if (DEVICE_TYPE_END_DEVICE == appDeviceType)
        appEndDeviceTaskHandler();
#endif // _ENDDEVICE_
      break;

    default:
      break;
  }
}
/*************************************************************************
  Description:  application and stack parameters init
  Parameters:   none
  Returns:      none
*************************************************************************/
static void initApp(void)
```

```
{
  ShortAddr_t nwkAddr;
  bool rxOnWhenIdle;

  appState = INITIAL_APP_STATE;

  // Read NWK address as dipswitch's state.
  nwkAddr = 1;//appReadSliders(); --> modificado para end device

  if (0 == nwkAddr)
  {
#ifdef _COORDINATOR_
    appDeviceType = DEVICE_TYPE_COORDINATOR;
    rxOnWhenIdle = true;

    appCoordinatorInit();
#else
    return; // This device can not be coordinator
#endif // _COORDINATOR_
  }
  else
  {
#ifdef _ENDDEVICE_
    appDeviceType = DEVICE_TYPE_END_DEVICE;
    rxOnWhenIdle = false;

    appEndDeviceInit();
#else
    return; // This device can not be end device
#endif // _ENDDEVICE_
  }

  // Set parameters to config server
  CS_WriteParameter(CS_NWK_ADDR_ID, &nwkAddr);
  CS_WriteParameter(CS_DEVICE_TYPE_ID, &appDeviceType);
  CS_WriteParameter(CS_RX_ON_WHEN_IDLE_ID, &rxOnWhenIdle);

  appState = APP_NETWORK_JOINING_STATE;   // Application state to join network switching
  SYS_PostTask(APL_TASK_ID);              // Application task posting

}
```

```
/*************************************************************************
  Description: ZDO_StartNetwork primitive confirmation was received.
  Parameters: confirmInfo - confirmation information
  Returns:    none
*************************************************************************/
void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confInfo)
{
  HAL_StopAppTimer(&networkTimer);              // Network join state indication timer stopping

  // Joined network successfully
  if (ZDO_SUCCESS_STATUS == confInfo->status)     // Network was started successfully
  {
    appState = APP_NETWORK_JOINED_STATE;          // Application state switching
    appDeviceState = DEVICE_ACTIVE_IDLE_STATE;    // Device state setting

    // Turn network indication on
    //appOnLed(APP_NETWORK_STATUS_LED); --> nao tem led

    // Set application endpoint properties and register endpoint
    apsRegisterEndpointReq.simpleDescriptor = &simpleDescriptor;
    apsRegisterEndpointReq.APS_DataInd = APS_DataInd;
    APS_RegisterEndpointReq(&apsRegisterEndpointReq);
  }

  SYS_PostTask(APL_TASK_ID);
}
/*************************************************************************
  Description: start network
  Parameters: none.
  Returns:    none
*************************************************************************/
static void startNetwork(void)
{
  // Configure timer for LED blinking during network start
  networkTimer.interval = APP_JOINING_INDICATION_PERIOD;
  networkTimer.mode     = TIMER_REPEAT_MODE;
  networkTimer.callback = startingNetworkTimerFired;
  HAL_StartAppTimer(&networkTimer);

  zdoStartNetworkReq.ZDO_StartNetworkConf = ZDO_StartNetworkConf; // Network started confirm
handler
  ZDO_StartNetworkReq(&zdoStartNetworkReq); // start network
```

```
}
/*************************************************************************
  Description: Starting network timer has fired. Toggle LED for blink
  Parameters: none.
  Returns:    none
*************************************************************************/
void startingNetworkTimerFired(void)
{
  //appToggleLed(APP_NETWORK_STATUS_LED);--> não usa led
}
/*************************************************************************
  Description:  Application endpoint indication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************************/
void APS_DataInd(APS_DataInd_t* ind)
{
#ifdef _COORDINATOR_
  if (DEVICE_TYPE_COORDINATOR == appDeviceType)
    appCoordinatorDataInd(ind);
#endif // _COORDINATOR_

#ifdef _ENDDEVICE_
  if (DEVICE_TYPE_END_DEVICE == appDeviceType)
    appEndDeviceDataInd(ind);
#endif // _ENDDEVICE_
}
/*************************************************************************
  Description: Network update notification

  Parameters: ZDO_MgmtNwkUpdateNotf_t *nwkParams - update notification

  Returns: nothing.
*************************************************************************/
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
```

AIII.2 Código comum implementado no Módulo Atuador.

```
/**************************************************************************//**
Atuador.c
Autores:    Alexandra e Breno
******************************************************************************/
#ifdef _ENDDEVICE_
#include <lowpower.h>
/******************************************************************************
  Global variables
 ******************************************************************************/
#define on  5
#define off 7
/******************************************************************************
  Local variables
 ******************************************************************************/
static AppDataTransmissionState_t appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
static APS_DataReq_t apsDataReq;                    // APS Data Request primitive (for application message
sending)
//static ZDO_SleepReq_t zdoSleepReq;                  // Request parameters for stack sleep
//static ZDO_WakeUpReq_t zdoWakeUpReq;                    // Request parameters for stack awakening
static AppMessageBuffer_t appMessageBuffer;             // Application message buffer

static int8_t sinal_compressor = off;
static int8_t sinal_bomba = off;

static int cont_comp = 0;
static int inicia = 0;
static int habilitado = 1;

static int8_t bomba = off;
static int8_t ventilador = off;
static int8_t compressor = off;
static int8_t damper = off;
```

```
//static int dado[5];

static HAL_AppTimer_t atuadorTimer;                         // Timer indicating network start
//static int iii = 0;// varialvel de teste de pacotes

// Temporary data received via network buffer
static uint8_t tmpDataBuffer[APP_TMP_DATA_BUFFER_SIZE];
static uint8_t tmpDataBufferActualLength = 0;

// USART related variables
static HAL_UsartDescriptor_t appUsartDescriptor;        // USART descriptor (required by stack)
static bool usartTxBusyFlag = false;                    // USART transmission transaction status
static uint8_t usartTxBuffer[APP_USART_TX_BUFFER_SIZE]; // USART Tx buffer

/******************************************************************************
  Static functions
******************************************************************************/
//static void ZDO_SleepConf(ZDO_SleepConf_t *conf);           // Sleep confirmation handler
//static void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf);         // Wake up confirmation handler
static void APS_DataConf(APS_DataConf_t *conf);             // Data transmission confirmation handler

//static void testeSensorHandler(bool result, int dado[5]);
static void DadosHandler(void);

static void atuadorTimerInit(void);
static void atualizaEstado(void);
static void habilita(void);

//static void manda_atuador(void);
//static void mandaatuadorTimerInit(void);

static void sendMessage(void);                          // Send the application message
void appEndDeviceDataInd(APS_DataInd_t* ind);

/******************************************************************************
  Local functions
 ********************************
 ***********************************************/
static void usartInit(void);
//static void sendDataToUsart(uint8_t* data, uint8_t length);
static void usartWriteConf(void);

static void gpioInit(void);

/******************************************************************
  Description:  End device initialization routine
  Parameters:   none
```

```
  Returns:      none
******************************************************************************/
void appEndDeviceInit(void)
{
  // Prepare APS Data Request
  apsDataReq.dstAddrMode           = APS_SHORT_ADDRESS;                  // Short addressing mode
  apsDataReq.dstAddress.shortAddress = 0;                                // Destination node short address
  apsDataReq.dstEndpoint           = APP_ENDPOINT;                       // Destination endpoint
  apsDataReq.profileId             = APP_PROFILE_ID;                     // Profile ID
  apsDataReq.clusterId             = APP_CLUSTER_ID;                     // Destination cluster ID
  apsDataReq.srcEndpoint           = APP_ENDPOINT;                       // Source endpoint
  apsDataReq.asduLength            = sizeof (AppMessage_t);              // ASDU size
  apsDataReq.asdu                  = (uint8_t *) &appMessageBuffer.message; // ASDU pointer as an application
message
  apsDataReq.txOptions.acknowledgedTransmission = 1;                     // Acknowledged transmission enabled
  apsDataReq.radius                = 0;                                  // Default radius
  apsDataReq.APS_DataConf          = APS_DataConf;                       // Confirm handler

  usartInit();
  atuadorTimerInit();
  gpioInit();

}
/*************************************************************************
  Description:  Device common task handler
  Parameters:   none
  Returns:      none
*************************************************************************/
void appEndDeviceTaskHandler(void)
{
    int dado[5];
  //uint8_t* data = "connect ";
  //uint8_t length = 8;

  switch (appDeviceState)                // Actual device state when one joined network
  {
    case DEVICE_ACTIVE_IDLE_STATE:           // Device ready to temperature measuring
            //sendDataToUsart(data , length); // write received data to USART
    default:
      break;
  }
}
/*************************************************************************
  Description:  Data intication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************************/
```

```
void appEndDeviceDataInd(APS_DataInd_t* ind)
{
  ind = ind; // Warning prevention

  AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;

  sinal_compressor = appMessage->dado[0][0];
  sinal_bomba = appMessage->dado[0][1];

  //uint8_t teste = 35;

            //atualizaEstado();

  // Data received indication
  //appToggleLed(APP_RECEIVING_STATUS_LED);// Data received indication
  SYS_PostTask(APL_TASK_ID);
}

/************************************************************************
  Description:  Data sent handler
  Parameters:   conf - APS Data Confirm primitive
  Returns:      none
************************************************************************/
static void APS_DataConf(APS_DataConf_t *conf)
{
  appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; // Data transmission entity is idle

  //appOffLed(APP_SENDING_STATUS_LED);

  if (APS_SUCCESS_STATUS == conf->status) // Data transmission was successfully performed
  {
    appDeviceState = DEVICE_ACTIVE_IDLE_STATE; // Switch device state to prepare for sleep
  }
  else
  {
    appDeviceState = DEVICE_ACTIVE_IDLE_STATE; // Data transmission wasn't successfully finished. Retry.
  }

  SYS_PostTask(APL_TASK_ID);
}

/************************************************************************
  Description:  Temperature measured handler breno
  Parameters:   result - measurement status (true - success, 0 - fail)
                temperature - value measured
  Returns:      none
************************************************************************/
```

```
static void DadosHandler(void)
{

//          uint8_t str[50];
//          uint8_t length;

  // Os dados serao enviados como uma menssagem de aplicação

    appMessageBuffer.message.dado[0][0] = bomba;
            appMessageBuffer.message.dado[0][1] = ventilador;
            appMessageBuffer.message.dado[0][2] = compressor;
            appMessageBuffer.message.dado[0][3] = damper;

  // Imprime na tela via serial os dados que foram enviados
            //length = sprintf((char*) str," %d %d %d %d \r\n", bomba,ventilador,compressor, damper);
            //sendDataToUsart(str , length);


  appDeviceState = DEVICE_MESSAGE_SENDING_STATE; //Troca o estado do end device para o estado de
envio de mensagem
  SYS_PostTask(APL_TASK_ID);                                    //passa o comando para o Application
task
}
/************************************************************************
  Description:  Send the application message
  Parameters:   none
  Returns:      none
************************************************************************/
static void sendMessage(void)
{

  if (APP_DATA_TRANSMISSION_IDLE_STATE == appDataTtransmissionState)  // If previous data transmission
was finished
  {
    appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;     // Data transmission entity is
busy while sending not finished
    APS_DataReq(&apsDataReq);

  }
  SYS_PostTask(APL_TASK_ID);
}

/************************************************************************
  Description: Device wakeup handler.
  Parameters: none
  Returns:    none
************************************************************************/
```

```
static void wakeUpHandler(void)
{
  appDeviceState = DEVICE_ACTIVE_IDLE_STATE;

  SYS_PostTask(APL_TASK_ID);
}

/************************************************************************
  Description: End device wake up indication

  Parameters: none.

  Returns: nothing.
************************************************************************/
void ZDO_WakeUpInd(void)
{
  if (DEVICE_SLEEP_STATE == appDeviceState)
    wakeUpHandler();
}

/************************************************************************
  Description: Wake up confirmation handler

  Parameters: conf - confirmation parameters

  Returns: nothing.
************************************************************************/
void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf)
{
  if (ZDO_SUCCESS_STATUS == conf->status)
    wakeUpHandler();
  else
    SYS_PostTask(APL_TASK_ID);
}
void usartInit(void)
{
  usartTxBusyFlag = false;

  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       = NULL;
  appUsartDescriptor.rxBufferLength  = 0;
  appUsartDescriptor.txBuffer       = NULL; // use callback mode
```

```
  appUsartDescriptor.txBufferLength  = 0;
  appUsartDescriptor.rxCallback      = NULL;
  appUsartDescriptor.txCallback      = usartWriteConf;
  appUsartDescriptor.flowControl     = USART_FLOW_CONTROL_NONE;

  HAL_OpenUsart(&appUsartDescriptor);
}
/*****************************************************************
  Description: Send Data To Usart

  Parameters: data - pointer to frame being sent to USART
              length - length of the frame in bytes

  Returns: nothing.
*****************************************************************/
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: 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;
  }
}
static void atuadorTimerInit(void)
{

  atuadorTimer.interval = 5000L;
  atuadorTimer.mode     = TIMER_REPEAT_MODE;//TIMER_ONE_SHOT_MODE;//
  atuadorTimer.callback = atualizaEstado;//habilita;//
  HAL_StartAppTimer(&atuadorTimer);
}
static void habilita(void)
{

        habilitado = 1;

}
static void atualizaEstado(void)
{

        if((compressor == on) && ((sinal_compressor != on) || (sinal_bomba != off)))
        {

                inicia = 1;
                habilitado = 0;
                //atuadorTimerInit();
        }

        if(inicia == 1 && cont_comp < 36)
        {
                cont_comp++;
                if(cont_comp == 30)
                {
                        habilitado = 1;
                        inicia = 0;
                        cont_comp = 0;

                }
        }
```

```
if((sinal_compressor == off) && (sinal_bomba == off))
{

GPIO_0_clr();
GPIO_1_clr();
GPIO_2_clr();

bomba = off;
ventilador = off;
compressor = off;
damper = off;

}else if((sinal_compressor == off) && (sinal_bomba == on))
{

GPIO_0_set();
GPIO_1_set();
GPIO_2_clr();

bomba = on;
ventilador = on;
compressor = off;
damper = off;

}else if((sinal_compressor == on) && (sinal_bomba == off) && (habilitado == 1) )
{

GPIO_0_clr();
GPIO_1_set();
GPIO_2_set();

bomba = off;
ventilador = on;
compressor = on;
damper = on;

}else if((sinal_compressor == on) && (sinal_bomba == on))
{

GPIO_0_clr();
GPIO_0_clr();
GPIO_2_clr();

bomba = off;
ventilador = off;
compressor = off;
```

```
                    damper = off;

                    }

          DadosHandler();
         sendMessage();

         SYS_PostTask(APL_TASK_ID);

}

static void gpioInit(void)
{

GPIO_0_make_out();
GPIO_1_make_out();
GPIO_2_make_out();
GPIO_3_make_out();

}

#endif // _ENDDEVICE_

// eof enddevice.c
```

AIII.3 Código comum implementado no Módulo Interno.

```
/*****************************************************************************//**
  Interno.c
Autores:  Alexandra e Breno
*****************************************************************************/

#ifdef _ENDDEVICE_
#include <lowpower.h>
#include <gpio.h>//uso das entradas e saidas do microcontrolador
#include <halW1.h> //void __delay_us(uint8_t delay)
#include <math.h>
#include <util/delay.h>
#include <avr/sleep.h>      // Funções de sleep
#include <avr/pgmspace.h>           // Uso da EEPROM

// Coeficientes de conversão d1 de acordo com o datasheet
```

```
 #define SHT_TEMP_OFFSET -39.7  //@ 3.5V // #define SHT_TEMP_OFFSET -40.1      //@ 5V  // #define
SHT_TEMP_OFFSET -39.8  //@ 4V //#define SHT_TEMP_OFFSET -39.6  //@ 3V // #define
SHT_TEMP_OFFSET -39.4  //@ 2.5V

//------------------------------------------------------------------------------
// Criar estruturas das variaveis
typedef union {
        unsigned int i;
        float f;
} sht_value;

// Modo de medida
enum {TEMP,HUMI};

#define noACK 0
#define ACK   1

// Definições de endereço (sempre 000) e comandos do SHT71
                    //adr  command  r/w
#define STATUS_REG_W 0x06   //000   0011    0
#define STATUS_REG_R 0x07   //000   0011    1
#define MEASURE_TEMP 0x03   //000   0001    1
#define MEASURE_HUMI 0x05   //000   0010    1
#define RESET       0x1e   //000   1111    0
#define HEATER_BIT 0x04          //000      0010      0

//Definição de periodo
#define QuartoDePeriodo 1
#define MeioPeriodo 2
#define Periodo 4

//Definição das variaveis
#define Temperatura_1 0
#define Humidade_1    1
#define MediaRadiante 2
#define Temperatura_2 3
#define Humidade_2    4
#define Envia         5

/*****************************************************************************
Sensor Sensirion SHT71
Baseado no exemplo fornecido pela Sensirion
Muitas funções estão com nomes parecidos com os presentes na nota de aplicação
Foram incluídas funções para verificação da integridade dos dados
*****************************************************************************/
void sht_raw_to_physical(sht_value *p_humidity ,sht_value *p_temperature);
unsigned char sht_get_result(sht_value *p_sht_value, unsigned char *p_checksum);
```

```
unsigned char sht_get_result1(sht_value *p_sht_value, unsigned char *p_checksum);
void sht_raw_to_physical(sht_value *p_humidity ,sht_value *p_temperature) ;
void sht_init_measure(unsigned char mode);
void sht_init_measure1(unsigned char mode);
void periodo(uint8_t T);
unsigned char sht_statusreg_write(unsigned char *p_sht_value);
unsigned char sht_statusreg_write1(unsigned char *p_sht_value);
unsigned char sht_statusreg_read(unsigned char *p_sht_value, unsigned char *p_checksum);
unsigned char sht_statusreg_read1(unsigned char *p_sht_value, unsigned char *p_checksum);
void sht_resetconnection(void);
void sht_resetconnection1(void);
unsigned char sht_le_byte(unsigned char ack);
unsigned char sht_le_byte1(unsigned char ack);
unsigned char sht_escreve_byte1(unsigned char sht_value);
unsigned char sht_escreve_byte(unsigned char sht_value);
void sht_inicio(void);
void sht_inicio1(void);

/*********************************************************************************
  Global variables
 *********************************************************************************/

//static unsigned char errorStatusReg = 0;
static unsigned char checksum;
//static unsigned char statusReg = 0;

static long int TimeOut = 0;

//ADC related variables
static uint16_t adcBuffer[1]; // ADC buffer
static HAL_AdcChannelNumber_t adcCanal;
static HAL_AdcParams_t adcParametros;
int open_succes = 1;
int read_succes = 1;

static int dado[5];//varialvel onde ser armazenada os valores
static int sht_estado = 0;

/*********************************************************************************
  Local variables
 *********************************************************************************/
static AppDataTransmissionState_t appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
static APS_DataReq_t apsDataReq;                    // APS Data Request primitive (for application message
sending)
static ZDO_SleepReq_t zdoSleepReq;                  // Request parameters for stack sleep
static ZDO_WakeUpReq_t zdoWakeUpReq;                // Request parameters for stack awakening
static AppMessageBuffer_t appMessageBuffer;         // Application message buffer
```

```
/*******************************************/
static  sht_value humidity;
static  sht_value temperature;
static int variavel = 0;//variavel que esta sendo medida
/*******************************************/

// Temporary data received via network buffer
static uint8_t tmpDataBuffer[APP_TMP_DATA_BUFFER_SIZE];
static uint8_t tmpDataBufferActualLength = 0;

// USART related variables
static HAL_UsartDescriptor_t appUsartDescriptor;        // USART descriptor (required by stack)
static bool usartTxBusyFlag = false;                    // USART transmission transaction status
static uint8_t usartTxBuffer[APP_USART_TX_BUFFER_SIZE]; // USART Tx buffer



/*****************************************************************************
  Static functions
*****************************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf);          // Sleep confirmation handler
static void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf);        // Wake up confirmation handler
static void APS_DataConf(APS_DataConf_t *conf);            // Data transmission confirmation handler

static void DadosHandler(void);
static void sendMessage(void);                             // Send the application message
static void prepareToSleep(void);

/*****************************************************************************
  Local functions
 *****************************************************************************/
static void usartInit(void);
//static void sendDataToUsart(uint8_t* data, uint8_t length);
static void usartWriteConf(void);
static void adcInit(void);
static void adcConf(void);
static void adcRead(void);

/*****************************************************************
  Description:  End device initialization routine
  Parameters:   none
  Returns:      none
*****************************************************************/
void appEndDeviceInit(void)
{
  // Prepare APS Data Request
```

```
  apsDataReq.dstAddrMode           = APS_SHORT_ADDRESS;                  // Short addressing mode
  apsDataReq.dstAddress.shortAddress = 0;                                // Destination node short address
  apsDataReq.dstEndpoint           = APP_ENDPOINT;                       // Destination endpoint
  apsDataReq.profileId             = APP_PROFILE_ID;                     // Profile ID
  apsDataReq.clusterId             = APP_CLUSTER_ID;                     // Destination cluster ID
  apsDataReq.srcEndpoint           = APP_ENDPOINT;                       // Source endpoint
  apsDataReq.asduLength            = sizeof (AppMessage_t);              // ASDU size
  apsDataReq.asdu                  = (uint8_t *) &appMessageBuffer.message; // ASDU pointer as an application
message
  apsDataReq.txOptions.acknowledgedTransmission = 1;                     // Acknowledged transmission enabled
  apsDataReq.radius                = 0;                                  // Default radius
  apsDataReq.APS_DataConf          = APS_DataConf;                       // Confirm handler

  CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD, 30000L);//Periodo do tempo de sleep
            usartInit();

            adcInit();

            sht_resetconnection ();
            sht_resetconnection1 ();

}

/*************************************************************************
  Description:  Device common task handler
  Parameters:   none
  Returns:      none
*************************************************************************/
void appEndDeviceTaskHandler(void)
{

  int comando = 0;
  int leitura = 1;


  switch (appDeviceState)                // Actual device state when one joined network
  {
    case DEVICE_ACTIVE_IDLE_STATE:          // Device ready to temperature measuring

            switch (variavel){


                  case Temperatura_1: //medição da temperatura no sht

                        TimeOut++;                          //conta o nemero de vezes que ele passou pelo
Temperatura_1
                        if(TimeOut > 5000)          // se for maior que 5000 vezes passa para a próxima leitura
```

```
                {
                        variavel = Humidade_1;          //proxima leitura
                        TimeOut = 0;                            //reinicia o TimeOut
                        sht_estado = comando ;          //reinicia o estado do sht(leitura ou
comando)
                        SYS_PostTask(APL_TASK_ID);      //retorna o controle de fluxo para o task
handler
                }
                else if(sht_estado == comando)      //se o sht ainda não foi requisitado a medição de
temperatura
                {
                        sht_init_measure(TEMP);         //comando de inicio de medição
                        sht_estado = leitura ;          //atualiza o estado do sht para que faça a
medição
                        SYS_PostTask(APL_TASK_ID);      //retorna o controle de fluxo para o task
handler
                }

        if(GPIO_1_read()==0){                                                   //checa se o
dado ja foi medido pelo sht para que possa ser feita a leitura
                        sht_get_result(&temperature, &checksum);  //le o dado colhido pelo sht e
armazena na struct temperature
                        sht_estado = comando;
    //atualiza o estado do sht para requisição de uma nova medida
                        dado[Temperatura_1] = temperature.i;            //atualiza o dado
                        variavel = Humidade_1;
    //Próxima varialvel a ser medida
                        sht_resetconnection ();
    //reseta a conexaão com o sht
                        TimeOut = 0;
    //reinicia o TimeOut
                        SYS_PostTask(APL_TASK_ID);
    //retorna o controle de fluxo para o task handler
                }else{
                        SYS_PostTask(APL_TASK_ID);
    //retorna o controle de fluxo para o task handler
                }

        break;

        case Humidade_1:  //medição da humidade no sht

                TimeOut++;
                if(TimeOut > 1500)
                {
                        variavel = MediaRadiante;
                        TimeOut = 0;
```

```
                    sht_estado = comando ;                          // case
Humidade1:
                    SYS_PostTask(APL_TASK_ID);                      // Igual a
mediçao de temperatura unica diferença é
                }
            // a requisiçaõ da humidade ao invés de temperatura
                else if(sht_estado == comando)                      // break;
                {
                    sht_init_measure(HUMI);
                    sht_estado = leitura ;
                    SYS_PostTask(APL_TASK_ID);
                }

                if(GPIO_1_read()==0){
                    sht_get_result(&humidity, &checksum);
                    sht_estado = comando;
                    dado[Humidade_1] = humidity.i;
                    variavel = MediaRadiante;
                    TimeOut = 0;
                    SYS_PostTask(APL_TASK_ID);
                }else{
                    SYS_PostTask(APL_TASK_ID);
                }

            break;

            case MediaRadiante:

                variavel = Temperatura_2; //Próxima varialvel a ser medida
                adcRead();                                  //o dado é atualizado dentro da
função adcRead() para ganhar tempo

            break;

            case Temperatura_2:

                TimeOut++;
                if(TimeOut > 5000)
                {
                    variavel = Humidade_2;
                    TimeOut = 0;
        // case Temperatura2:
                    sht_estado = comando ;                          // Igual
a mediçao de temperatura1 única diferença é
                    SYS_PostTask(APL_TASK_ID);                      // a
requisiçaõ em pinos diferentes
                }else if(sht_estado == comando){                    // break;
```

```
                    sht_init_measure1(TEMP);
                    sht_estado = leitura ;
                    SYS_PostTask(APL_TASK_ID);
                }

                if(GPIO_4_read()==0){
                    sht_get_result1(&temperature, &checksum);
                    sht_estado = comando;
                    dado[Temperatura_2] = temperature.i;
                    sht_resetconnection1 ();
                    variavel = Humidade_2;
                    TimeOut = 0;
                    SYS_PostTask(APL_TASK_ID);
                }else{
                    SYS_PostTask(APL_TASK_ID);
                }

        break;

        case Humidade_2:

                TimeOut++;
                if(TimeOut > 1500)
                {
                    variavel = Envia;
// case Humidade2:
                    TimeOut = 0;
        // Igual a mediçao de Humidade1 única diferença é
                    sht_estado = comando ;
// a requisiçaõ em pinos diferentes
                    SYS_PostTask(APL_TASK_ID);
// break;
                }else if(sht_estado == comando){
                    sht_init_measure1(HUMI);
                    sht_estado = leitura ;
                    SYS_PostTask(APL_TASK_ID);
                }

                if(GPIO_4_read()==0){
                    sht_get_result1(&humidity, &checksum);
                    sht_estado = comando;
                    dado[Humidade_2] = humidity.i;
                    variavel = Envia;
                    TimeOut = 0;
                    SYS_PostTask(APL_TASK_ID);
                }else{
                    SYS_PostTask(APL_TASK_ID);
```

```
                        }

                    break;

                    case Envia:

                        variavel = Temperatura_1;//reinicia o a primeira variavel a ser medida
                        sht_estado = comando ;          //reinicia o estado do sht de comando para
requisição da variavel
                        TimeOut = 0;                    //reinicia o TimeOut
                        DadosHandler();                 //função para enviar todos os dados
colhidos

                    break;
          }//fim do switch de requisição de variaveis

   break;

   case DEVICE_MESSAGE_SENDING_STATE:     // Estado de envio de mensagem
    sendMessage();                  // Aplicação de envio de mensagem
   break;

   case DEVICE_SLEEP_PREPARE_STATE:       // Estado de preparação para dormir
    prepareToSleep();               // Preparaçao para dormir
   break;

   case DEVICE_AWAKENING_STATE:                    // Estado acordando
    zdoWakeUpReq.ZDO_WakeUpConf = ZDO_WakeUpConf; // definição do ZDO WakeUp confirm handler
    ZDO_WakeUpReq(&zdoWakeUpReq);          // envio da requização do ZDO WakeUp
   break;

   default:
   break;
  }// fim do switch idle state
}

/*************************************************************************
  Description:  Data intication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************************/
void appEndDeviceDataInd(APS_DataInd_t* ind)
{
  ind = ind; // Warning prevention

}
```

```
/************************************************************************
  Description:  Data sent handler
  Parameters:   conf - APS Data Confirm primitive
  Returns:      none
************************************************************************/

static void APS_DataConf(APS_DataConf_t *conf)
{
  appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; // Data transmission entity is idle

  //appOffLed(APP_SENDING_STATUS_LED);

  if (APS_SUCCESS_STATUS == conf->status) // Data transmission was successfully performed
  {
    appDeviceState = DEVICE_SLEEP_PREPARE_STATE; // Switch device state to prepare for sleep
  }
  else
  {
    appDeviceState = DEVICE_ACTIVE_IDLE_STATE; // Data transmission wasn't successfully finished. Retry.
  }


  SYS_PostTask(APL_TASK_ID);
}

/************************************************************************
  Description:  Temperature measured handler breno
  Parameters:   result - measurement status (true - success, 0 - fail)
                temperature - value measured
  Returns:      none
************************************************************************/
static void DadosHandler(void)
{

//          uint8_t str[50];
//          uint8_t length;
 // Os dados serao enviados como uma menssagem de aplicação

    appMessageBuffer.message.dado[0][0] = dado[0];
            appMessageBuffer.message.dado[0][1] = dado[1];
            appMessageBuffer.message.dado[0][2] = dado[2];
            appMessageBuffer.message.dado[0][3] = dado[3];
            appMessageBuffer.message.dado[0][4] = dado[4];

  // Imprime na tela via serial os ados que foram enviados
```

```
//          length = sprintf((char*) str," %d %d %d %d %d \r\n", dado[0],dado[1],dado[2],dado[3],dado[4]);
//          sendDataToUsart(str , length);


  appDeviceState = DEVICE_MESSAGE_SENDING_STATE; //Troca o estado do end device para o estado de
envio de mensagem
  SYS_PostTask(APL_TASK_ID);                                    //passa o comando para o Application
task
}
/************************************************************************
  Description:  Send the application message
  Parameters:   none
  Returns:      none
************************************************************************/
static void sendMessage(void)
{

  if (APP_DATA_TRANSMISSION_IDLE_STATE == appDataTtransmissionState)  // Verifica se os dados
anteriores foram enviados
  {
    appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;     // Envio de mensagem esta
ocupado enquanto a transmissao nao se encerra
    APS_DataReq(&apsDataReq);

  }
}



/************************************************************************
  Description:  ZDO Sleep Confirm handler
  Parameters:   conf - ZDO Sleep Confirm primitive
  Returns:      none
************************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf)
{

  if (ZDO_SUCCESS_STATUS == conf->status)   // Pilha foi corretamente colocada para dormir
  {


    appDeviceState = DEVICE_SLEEP_STATE;    // EndDevice esta no estado de dormindo
  }
  else
    SYS_PostTask(APL_TASK_ID);              // Ainda no estado acordado
                                  // Passa para o handler de aplicação para uma nova tentativa
}
```

```
/*************************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
*************************************************************************/
static void prepareToSleep(void)
{

  zdoSleepReq.ZDO_SleepConf = ZDO_SleepConf;  // Define a função de confirmação do estado de sleep
  ZDO_SleepReq(&zdoSleepReq);                 // Função de requisição para dormir
}

/*************************************************************************
  Description: Device wakeup handler.
  Parameters: none
  Returns:    none
*************************************************************************/
static void wakeUpHandler(void)
{

  appDeviceState = DEVICE_ACTIVE_IDLE_STATE;  //Atualiza o estado do End Device para acordado

  SYS_PostTask(APL_TASK_ID);
}

/*************************************************************************
  Description: End device wake up indication

  Parameters: none.

  Returns: nothing.
*************************************************************************/
void ZDO_WakeUpInd(void)
{
  if (DEVICE_SLEEP_STATE == appDeviceState)
    wakeUpHandler();                                          //chama a função para acordar o end
device
}

/*************************************************************************
  Description: Wake up confirmation handler

  Parameters: conf - confirmation parameters

  Returns: nothing.
*************************************************************************/
void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf)
```

```
{
  if (ZDO_SUCCESS_STATUS == conf->status)
    wakeUpHandler();                                    //chama a função para acordar o end device
  else
    SYS_PostTask(APL_TASK_ID);
}
/************************************************************************
  Description: Configuraçao da porta Usart

  Parameters: conf - parametros de configuração

  Returns: nothing.
************************************************************************/

void usartInit(void)
{
  usartTxBusyFlag = false;

  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       = NULL;
  appUsartDescriptor.rxBufferLength  = 0;
  appUsartDescriptor.txBuffer       = NULL; // use callback mode
  appUsartDescriptor.txBufferLength  = 0;
  appUsartDescriptor.rxCallback     = NULL;
  appUsartDescriptor.txCallback     = usartWriteConf;
  appUsartDescriptor.flowControl    = USART_FLOW_CONTROL_NONE;

  HAL_OpenUsart(&appUsartDescriptor);
}
/************************************************************************
  Description: Envia dados para Usart

  Parameters: data - ponteiro para pointer to frame being sent to USART
              length - length of the frame in bytes

  Returns: nothing.
************************************************************************/
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: 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;
  }
}

static void adcInit(void)
{

uint16_t numeroAmostras = 1;

adcCanal = HAL_ADC_CHANNEL1;

adcParametros.resolution = RESOLUTION_10_BIT;
adcParametros.sampleRate = ADC_9600SPS;
adcParametros.voltageReference = AVCC;
adcParametros.bufferPointer = adcBuffer;
adcParametros.selectionsAmount = numeroAmostras;
```

```
adcParametros.callback = adcConf;


open_succes = HAL_OpenAdc(&adcParametros);

}

static void adcConf(void)
{

}

void adcRead(void)
{
	if(open_succes == 0)
	{
	read_succes = HAL_ReadAdc (HAL_ADC_CHANNEL1);

	}
//	else -->caso nao tenha sido lido corretamente

	if(read_succes == 0)
	{
	dado[2]= (unsigned int) *adcBuffer;

	}

  SYS_PostTask(APL_TASK_ID); // Application task posting
}

/*******************************************************************************
Sensor Sensirion SHT71 Breno Código

*******************************************************************************/

void periodo(uint8_t T)
{
	__delay_us(T);

}

// Início de transmissão para o 1º sht
void sht_inicio(void)
{
	GPIO_0_make_out();

	GPIO_1_make_out();//MAKE_SHT_DATA_PIN_OUTPUT;
```

```
//          periodo(Periodo);

            // Estado inicial
            GPIO_1_set();//SET_SHT_DATA;
            GPIO_0_clr();//SCK_OFF();

            periodo(MeioPeriodo);


            GPIO_0_set();//SCK_ON();

            periodo(QuartoDePeriodo);

            GPIO_1_clr();//CLEAR_SHT_DATA;

            periodo(QuartoDePeriodo);

            GPIO_0_clr();//SCK_OFF();

            periodo(MeioPeriodo);

            GPIO_0_set();//SCK_ON();

            periodo(QuartoDePeriodo);

            GPIO_1_set();//SET_SHT_DATA;

            periodo(QuartoDePeriodo);

            GPIO_0_clr();//SCK_OFF();

            periodo(QuartoDePeriodo);

            GPIO_1_clr();//CLEAR_SHT_DATA;

}

// Início de transmissão para o 2º sht
void sht_inicio1(void)
{
            GPIO_2_make_out();

            GPIO_4_make_out();//MAKE_SHT_DATA_PIN_OUTPUT;
//          periodo(Periodo);

            // Estado inicial
            GPIO_4_set();//SET_SHT_DATA;
```

```
	GPIO_2_clr();//SCK_OFF();

	periodo(MeioPeriodo);


	GPIO_2_set();//SCK_ON();

	periodo(QuartoDePeriodo);

	GPIO_4_clr();//CLEAR_SHT_DATA;

	periodo(QuartoDePeriodo);

	GPIO_2_clr();//SCK_OFF();

	periodo(MeioPeriodo);

	GPIO_2_set();//SCK_ON();

	periodo(QuartoDePeriodo);

	GPIO_4_set();//SET_SHT_DATA;

	periodo(QuartoDePeriodo);

	GPIO_2_clr();//SCK_OFF();

	periodo(QuartoDePeriodo);

	GPIO_4_clr();//CLEAR_SHT_DATA;

}

// Escreve byte e checa ack
unsigned char sht_escreve_byte(unsigned char sht_value)
{

	unsigned char i;
	unsigned char error = 0;
	GPIO_1_make_out();

	for (i=0x80;i>0;i/=2) {
		if (i & sht_value)
		GPIO_1_set();//SET_SHT_DATA;
		else
		GPIO_1_clr();//CLEAR_SHT_DATA;
```

```
                GPIO_0_set();//SCK_ON();
                periodo(MeioPeriodo);
                GPIO_0_clr();//SCK_OFF();
                periodo(MeioPeriodo);       // ????????????
        }

        GPIO_1_set();//SET_SHT_DATA;   // Libera DATA-line pull_up
        GPIO_1_make_in();//MAKE_SHT_DATA_PIN_INPUT;
        GPIO_1_set();//SET_SHT_DATA;   // Libera DATA-line pull_up

        GPIO_0_set();//SCK_ON()// clk #9 para ack
        periodo(MeioPeriodo);
        if (GPIO_1_read()) error = 1;          // checa ack (DATA pulled down pelo SHT71)
        GPIO_0_clr();//SCK_OFF();
        return error;         // error = 1 em caso de no ack
}

// Escreve byte e checa ack
unsigned char sht_escreve_byte1(unsigned char sht_value)
{

        unsigned char i;
        unsigned char error = 0;
        GPIO_4_make_out();

        for (i=0x80;i>0;i/=2) {
                if (i & sht_value)
                GPIO_4_set();//SET_SHT_DATA;
                else
                GPIO_4_clr();//CLEAR_SHT_DATA;

                GPIO_2_set();//SCK_ON();
                periodo(MeioPeriodo);
                GPIO_2_clr();//SCK_OFF();
                periodo(MeioPeriodo);       // ????????????
        }

        GPIO_4_set();//SET_SHT_DATA;   // Libera DATA-line pull_up
        GPIO_4_make_in();//MAKE_SHT_DATA_PIN_INPUT;
        GPIO_4_set();//SET_SHT_DATA;   // Libera DATA-line pull_up

        GPIO_2_set();//SCK_ON()// clk #9 para ack
        periodo(MeioPeriodo);
        if (GPIO_4_read()) error = 1;          // checa ack (DATA pulled down pelo SHT71)
        GPIO_2_clr();//SCK_OFF();
        return error;         // error = 1 em caso de no ack
}
```

```
// Lê byte e fornece ack em caso de ack = 1
unsigned char sht_le_byte(unsigned char ack)
{

	unsigned char i;
	unsigned char val=0;

	GPIO_1_make_in();//MAKE_SHT_DATA_PIN_INPUT;
	GPIO_1_set(); // SET_SHT_DATA; // release DATA-line

	for (i=0x80;i>0;i/=2)
	{	// shift bit
		GPIO_0_set();//SCK_ON();
		periodo(MeioPeriodo);

		if (GPIO_1_read())
		val=(val | i); // ler bit (SHT_DATA_PORT_PIN & (1<< SHT_DATA_PIN))

		GPIO_0_clr();//SCK_OFF();
		periodo(MeioPeriodo);//???????????????

	}
	GPIO_1_make_out();//MAKE_SHT_DATA_PIN_OUTPUT;

	if (ack)
	GPIO_1_clr();// CLEAR_SHT_DATA;
	else
	GPIO_1_set();//SET_SHT_DATA;

	GPIO_0_set();//SCK_ON(); //clk #9 for ack]
	periodo(MeioPeriodo);
	GPIO_0_clr();//SCK_OFF();
	periodo(MeioPeriodo);
	GPIO_1_set();//SET_SHT_DATA; //release DATA-line
	return val;
}

// Lê byte e fornece ack em caso de ack = 1
unsigned char sht_le_byte1(unsigned char ack)
{

	unsigned char i;
	unsigned char val=0;

	GPIO_4_make_in();//MAKE_SHT_DATA_PIN_INPUT;
	GPIO_4_set(); // SET_SHT_DATA; // release DATA-line
```

```
	for (i=0x80;i>0;i/=2)
	{	// shift bit
		GPIO_2_set();//SCK_ON();
		periodo(MeioPeriodo);

		if (GPIO_4_read())
		val=(val | i); // ler bit (SHT_DATA_PORT_PIN & (1<< SHT_DATA_PIN))

		GPIO_2_clr();//SCK_OFF();
		periodo(MeioPeriodo);//???????????????

	}
	GPIO_4_make_out();//MAKE_SHT_DATA_PIN_OUTPUT;

	if (ack)
	GPIO_4_clr();// CLEAR_SHT_DATA;
	else
	GPIO_4_set();//SET_SHT_DATA;

	GPIO_2_set();//SCK_ON(); //clk #9 for ack]
	periodo(MeioPeriodo);
	GPIO_2_clr();//SCK_OFF();
	periodo(MeioPeriodo);
	GPIO_4_set();//SET_SHT_DATA; //release DATA-line
	return val;
}

// Reseta comunicação: DATA-line = 1 e pelo menos 9 SCK seguido de transstart
void sht_resetconnection(void)
 {
	unsigned char i;
	GPIO_0_make_out();//MAKE_SHT_SCK_PIN_OUTPUT;
	GPIO_1_make_out();//MAKE_SHT_DATA_PIN_OUTPUT;
	GPIO_1_set();//SET_SHT_DATA;
	GPIO_0_clr();//CLEAR_SHT_SCK; // Estado inicial

	for(i=0;i<9;i++) {   //9 SCK cycles
		GPIO_0_set();//SCK_ON();
		periodo(MeioPeriodo);
		GPIO_0_clr();//SCK_OFF();
		periodo(MeioPeriodo);
	}
	sht_inicio(); // Início de transmissão
}

// Reseta comunicação: DATA-line = 1 e pelo menos 9 SCK seguido de transstart
```

```
void sht_resetconnection1(void)
 {
        unsigned char i;
        GPIO_2_make_out();//MAKE_SHT_SCK_PIN_OUTPUT;
        GPIO_4_make_out();//MAKE_SHT_DATA_PIN_OUTPUT;
        GPIO_4_set();//SET_SHT_DATA;
        GPIO_2_clr();//CLEAR_SHT_SCK; // Estado inicial

        for(i=0;i<9;i++) {   //9 SCK cycles
                GPIO_2_set();//SCK_ON();
                periodo(MeioPeriodo);
                GPIO_2_clr();//SCK_OFF();
                periodo(MeioPeriodo);
        }
        sht_inicio1(); // Início de transmissão
}

// Lê status register com checksum
unsigned char sht_statusreg_read(unsigned char *p_sht_value, unsigned char *p_checksum)
{

        unsigned char error = 0;

        sht_inicio();         // Início de transmissão
        error = sht_escreve_byte(STATUS_REG_R);          // Envia comando para o sensor
        *p_sht_value = sht_le_byte(ACK);   // Lê status register
        *p_checksum = sht_le_byte(noACK);          // Lê checksum
        return error;         // error = 1 caso o sensor não responda
}
// Lê status register com checksum
unsigned char sht_statusreg_read1(unsigned char *p_sht_value, unsigned char *p_checksum)
{
        unsigned char error = 0;

        sht_inicio1();        // Início de transmissão
        error = sht_escreve_byte1(STATUS_REG_R);          // Envia comando para o sensor
        *p_sht_value = sht_le_byte1(ACK);// Lê status register
        *p_checksum = sht_le_byte1(noACK);         // Lê checksum
        return error;         // error = 1 caso o sensor não responda
}
// Escreve status register com checksum
unsigned char sht_statusreg_write(unsigned char *p_sht_value)
{
        unsigned char error=0;

        sht_inicio();         // Início de transmissão
        error+=sht_escreve_byte(STATUS_REG_W);          // Envia comando para o sensor
```

```
	error+=sht_escreve_byte(*p_sht_value);	// Envia valor do status register
	return error;	// error >= 1 caso o sensor não responda
}

// Escreve status register com checksum
unsigned char sht_statusreg_write1(unsigned char *p_sht_value)
{
	unsigned char error=0;

	sht_inicio1();	// Início de transmissão
	error+=sht_escreve_byte1(STATUS_REG_W);	// Envia comando para o sensor
	error+=sht_escreve_byte1(*p_sht_value);	// Envia valor do status register
	return error;	// error >= 1 caso o sensor não responda
}
// Realiza uma medida de temperatura ou umidade com checksum
void sht_init_measure(unsigned char mode)
{
	unsigned char error=0;

	sht_inicio();

	switch(mode){	// Envia comando para o sensor
	case TEMP : error+=sht_escreve_byte(MEASURE_TEMP); break;
	case HUMI : error+=sht_escreve_byte(MEASURE_HUMI); break;
	default : break;
	}
}
// Realiza uma medida de temperatura ou umidade com checksum
void sht_init_measure1(unsigned char mode)
{
	unsigned char error=0;

	sht_inicio1();

	switch(mode){	// Envia comando para o sensor
	case TEMP : error+=sht_escreve_byte1(MEASURE_TEMP); break;
	case HUMI : error+=sht_escreve_byte1(MEASURE_HUMI); break;
	default : break;
	}
}
unsigned char sht_get_result(sht_value *p_sht_value, unsigned char *p_checksum)
{
	unsigned char error=0;
	sht_value sht_value_temp;

	if (GPIO_1_read()) error = 1;
```

```
	sht_value_temp.i = 256*sht_le_byte(ACK);   // Leitura do MSB
	sht_value_temp.i += sht_le_byte(ACK);      // Leitura do LSB
	*p_checksum = sht_le_byte(noACK);          // Leitura do checksum
	*(p_sht_value)= sht_value_temp;
	return error;
}

unsigned char sht_get_result1(sht_value *p_sht_value, unsigned char *p_checksum)
{
	unsigned char error=0;
	sht_value sht_value_temp;

	if (GPIO_4_read()) error = 1;

	sht_value_temp.i = 256*sht_le_byte1(ACK); // Leitura do MSB
	sht_value_temp.i += sht_le_byte1(ACK);    // Leitura do LSB
	*p_checksum = sht_le_byte1(noACK);        // Leitura do checksum
	*(p_sht_value)= sht_value_temp;
	return error;

}
// Calcula temperatura [ºC] e umidade [%RH]
// Entrada: humi (12 bit) temp (14 bit)
void sht_raw_to_physical(sht_value *p_humidity ,sht_value *p_temperature) {
	const float C1=-4.0; // para 12 bit
	const float C2= 0.0405; // para 12 bit
	const float C3=-0.0000028; // para 12 bit
	const float T1=0.01; // para 14 bit
	const float T2=0.00008; // para 14 bit

	float rh_lin; // rh_lin: umidade linear
	float rh_true; // rh_true: umidade compensada pela temperatura
	float t_C; // t_C : temperatura [ºC]

	t_C = 0.01*(*p_temperature).i +(SHT_TEMP_OFFSET);        // Cálculo da temperatura [ºC] a partir da
leitura (raw data)
	rh_lin=C3*(*p_humidity).i*(*p_humidity).i + C2*(*p_humidity).i + C1; // Cálculo da umidade [%RH] a partir
da leitura
	rh_true=(t_C-25)*(T1+T2*(*p_humidity).i)+rh_lin; // Cálculo da umidade compensada pela temperatura

	// Adequa aos limites físicos
	if(rh_true>100)rh_true=100;
	if(rh_true<0.1)rh_true=0.1;

	(*p_temperature).f=t_C;  // Retorna temperatura [ºC]
	(*p_humidity).f=rh_true; // Retorna umidade %RH]
}
```

```
    #endif // _ENDDEVICE_

// eof enddevice.c
```