CENTRO UNIVERSITÁRIO DE BRASÍLIA- UniCEUB

FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS

CURSO DE ENGENHARIA DA COMPUTAÇÃO

PAULO EDUARDO PROPATO SANDOVAL

MONITORAMENTO REMOTO APLICADO AO CONTROLE DA UMIDADE AMBIENTAL

BRASÍLIA – DF

2° SEMESTRE DE 2014

PAULO EDUARDO PROPATO SANDOVAL

MONITORAMENTO REMOTO APLICADO AO CONTROLE DA UMIDADE AMBIENTAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof.ª. MSc. Francisco Javier de Obaldía Díaz

Brasília

Dezembro, 2014

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, por me proporcionarem absolutamente tudo nessa vida. Pelos valores e educação passados ao longo desses anos e pelo apoio, compreensão e confiança que sempre tiveram comigo.

Agradeço às minhas irmãs, minhas melhores amigas. A presença de vocês duas é essencial na minha vida.

Um agradecimento aos professores do UniCEUB, em especial ao professor e meu orientador MSc. Francisco Javier pela compreensão da falta de tempo para reuniões e pela motivação e estímulo que sempre me passou, atitude fundamental para que este projeto obtivesse êxito.

Aos colegas de trabalho, principalmente ao meu diretor e amigo Rafael Aguiar, obrigado pela paciência e apoio durante essa etapa.

Aos monitores e colegas que de alguma forma participaram desta caminhada para me tornar engenheiro, meu muito obrigado!

Paulo Eduardo Propato Sandoval

"Action is the foundational key to all success."

Pablo Picasso

SUMÁRIO

CAPÍTULO 1 INTRODUÇÃO 11
1.1 Apresentação do problema 11
1.2 Objetivos 12
1.2.1 Objetivo Geral 12
1.2.2 Objetivos Específicos 12
1.3 Motivação 13
1.4 Metodologia 13
1.5 Resultados Esperados 14
1.6 Estrutura do trabalho 14
CAPÍTULO 2 REFERENCIAL TEÓRICO 15
2.1 Elementos principais que compõem o protótipo 15
2.1.1 Plataforma Arduino 15
2.1.2 Sensor de temperatura e umidade DHT11 17
2.1.3 Sensor de BlueTooth HC-05 17
2.1.4 Sensor de nível da água ICOS LA16M-40 18
2.1.5 Relé 19
2.1.6 Transistor 2N3904 20
2.1.7 Resistor 22
2.1.8 Diodo 1N4007 23
2.2 Programação 23
2.2.1 Algoritmos 24
2.2.2 Estruturas de Dados 25
2.2.3 Linguagem C++ 28
2.2.4 GUI (Graphic Use Interface) 28
2.2.5 QT Creator 29
2.3 Umidade Relativa do Ar 32
2.4 Umidificador 33
CAPÍTULO 3 DESENVOLVIMENTO DO PROTÓTIPO 35
3.1 Visão Geral do Projeto 35
3.2 Descrição das Etapas 39

3.2.1 Montagem do Hardware 39
3.2.2 Escolha do modo de funcionamento.....40
3.2.3 Comunicação com Arduino.....44
3.2.4 Repasse de dados e acionamento do umidificador 49
CAPÍTULO 4 TESTES E RESULTADOS ALCANÇADOS 55
4.1.1 Captação dos sensores.....55
4.1.2 Comunicação com bluetooth 56
4.1.3 Sensor de nível.....58
4.1.4 Acionamento do relé.....61
4.1.5 Desligamento automático 61
CAPÍTULO 5 CONCLUSÕES E CONSIDERAÇÕES FINAIS 63
5.1 Conclusões 63
5.2 Sugestões para trabalhos futuros 63
Referências.....64
APÊNDICE A – PROGRAMA DO ARDUINO 66
APÊNDICE B – HEADER INTERFACE USUÁRIO.....71
APÊNDICE C – MAIN INTERFACE USUÁRIO 76
APÊNDICE D – PROGRAMA PRINCIPAL INTERFACE USUÁRIO.....77

LISTA DE FIGURAS

FIGURA 2. 1 - PLACA ARDUINO UNO 16
FIGURA 2. 2 - SENSOR DE UMIDADE E TEMPERATURA 17
FIGURA 2. 3 - MÓDULO BLUETOOTH HC-05 18
FIGURA 2. 4 - SENSOR DE NÍVEL DA ÁGUA 19
FIGURA 2. 5 RELÉ 1POLO 5V 20
FIGURA 2. 6 - TRANSISTOR COM DESCRITIVO DOS FILAMENTOS 21
FIGURA 2. 7 - TRANSISTOR 2N3904 21
FIGURA 2. 8 - RESISTORES 22
FIGURA 2. 9 - DIODO 1N4007 23
FIGURA 2. 10 - INTERFACE DO USUÁRIO DESENVOLVIDA PARA O PROJETO 29
FIGURA 2. 11 - PÁGINA DE ENTRADA DA IDE DO QT CREATOR 30
FIGURA 2. 12 - QT CREATOR OVERVIEW 30
FIGURA 2. 13 - EDITOR DO QT CREATOR 31
FIGURA 2. 14 - DESIGNER DO QT CREATOR 32
FIGURA 2. 15 - ÍNDICE DE UMIDADE RELATIVA X TEMPERATURA 33
FIGURA 2. 16 - UMIDIFICADOR UTILIZADO NO PROJETO 34
FIGURA 3. 1 - VISÃO GERAL DO PROJETO 35
FIGURA 3. 2 - HARDWARE DESENVOLVIDO 36
FIGURA 3. 3 - INTERFACE DE COMUNICAÇÃO 36
FIGURA 3. 4 - DIAGRAMA DE BLOCOS DO PROJETO 37
FIGURA 3. 5 - ESQUEMÁTICO DOS COMPONENTES NO PROTEUS 39
FIGURA 3. 6 GUI DESENVOLVIDO 41
FIGURA 3. 7 PORTA SERIAL DO BLUETOOTH 46
FIGURA 3. 8 - TABELA ASCII GRIFADA. 54
FIGURA 4. 1 - TESTE DE CAPTAÇÃO DE UMIDADE E TEMPERATURA 55
FIGURA 4. 2 - VARIAÇÃO NA UMIDADE E TEMPERATURA 56
FIGURA 4. 3 - INICIANDO O PROGRAMA TERA TERM PARA FAZER A CONEXÃO SERIAL 57
FIGURA 4. 4 - TERMINAL COM O PROGRAMA FUNCIONANDO 57
FIGURA 4. 5 - HARDWARE COM SENSOR ABERTO 58
FIGURA 4. 6 - RESPOSTAS DOS CENÁRIOS PROPOSTOS NO TERA TERM 59
FIGURA 4. 7 - RESPOSTA DO TESTE DE SENSOR DE NÍVEL NA INTERFACE PARA ESTADO VAZIO 59
FIGURA 4. 8 - HARDWARE COM SENSOR FECHADO 60
FIGURA 4. 9 - RESPOSTA DO TESTE DE SENSOR DE NÍVEL NA INTERFACE PARA ESTADO CHEIO 60
FIGURA 4. 10 - FUNCIONAMENTO DO UMIDIFICADOR 61

SIGLAS

EDR – Enhanced Data Rate
EPROM – Erasable Programmable Read Only Memory
GND – Ground
GUI – Graphical User Interface
IDE – Integrated Developmnet Enviroment
OMS – Organização Mundial de Saúde
RAM – Random Access Memory
RFCOMM - Radio Frequency Communications
SI – Sistema Internacional
SSP – Serial Port Protocol
USB – Universal Serial Bus

RESUMO

Este trabalho tem como proposta, desenvolver um sistema remoto de controle de umidificação. O projeto consiste em uma interface gráfica desenvolvida em C++ que é utilizada para controlar os modos de operação do umidificador. Para a comunicação entre o sistema e o *hardware* foi utilizada a conexão *bluetooth*. Além do controle, o sistema informa a temperatura e a umidade relativa do ar, também disponibilizando na interface, o nível do reservatório de água do umidificador. A plataforma Arduino é peça fundamental, uma vez que a conexão com o sistema esteja estabelecida, é ele quem, utilizando os sensores acoplados, disponibiliza os dados de temperatura do ambiente, umidade relativa do ar e o nível da água do reservatório, além de interpretar as instruções do sistema e acionar o funcionamento do umidificador.

Palavras chaves: Arduino, Umidificação, Bluetooth, C++, Remoto.

ABSTRACT

This work has as a purpose, develop a humidity control remote system. The project consists of a graphical interface developed using C++ which is used to control the humidifier's operational modes. It was used a Bluetooth connection for the communication between the system and the hardware. Besides the control, the system informs the temperature and the relative air humidity, also displaying at the interface, the water level at the humidifier's container. The Arduino's platform is fundamental key, once the connection with the system is established, it is it that, using coupled sensors, make available the environment temperature data, relative air humidity and the water container's level, besides to interpret the system instructions and activate the humidifier functioning.

Keywords: Arduino, Humidifier, Bluetooth, C++, Remote.

CAPÍTULO 1 INTRODUÇÃO

1.1 Apresentação do problema

O período de inverno é conhecido por potencializar várias doenças típicas da época, como gripes e alergias. Porém, outro fator vem preocupando as entidades de saúde: a falta de umidade.

Umidade relativa é definida como a relação de água no ar ambiente para a quantidade total de umidade que pode ser carregada no ar numa determinada temperatura ambiente, expressa como percentual.

Em 2011, nas primeiras semanas de agosto, Brasília chegou a registrar um nível de umidade comparável ao deserto do Saara, com 10% de umidade relativa do ar (SANTOS, 2011) e em 2010, 7%. (ESTADO, 2010). Segundo a Organização Mundial de Saúde (OMS), a umidade do ar ideal é de 60%. A OMS recomenda a declaração do estado de atenção quando os índices ficam abaixo de 30%.

Níveis baixos de umidade podem acentuar os efeitos de alergias respiratórias, já que a umidade do ar é essencial para ajudar o organismo a eliminar as impurezas provenientes da aspiração do ar. Pessoas com bronquite, rinite e asma, precisam redobrar os cuidados.

Para diminuir os efeitos da baixa umidade, normalmente, empregam-se umidificadores, que, aumentando os índices de umidade relativa do ambiente, podem ajudar a melhorar a qualidade do ar. Seu funcionamento é bastante simples: um pequeno tanque de água é acondicionado junto a um emissor de ultrassom, que quando ligado, quebra as moléculas de água. Combinados a um ventilador, há a dispersão da água em forma de névoa sobre o ambiente.

1.2 Objetivos

1.2.1 Objetivo Geral

O projeto tem como objetivo criar um sistema de *hardware* e *software* que irá controlar um umidificador para melhorar a umidade relativa do ar tomando cuidado para que também não exceda o nível considerado prejudicial pela Comissão Municipal da Defesa Civil de Campinas, definido como 80%.

Este sistema informa a temperatura do ambiente, umidade e nível da água no reservatório do umidificador, possuindo quatro modos: manual, automático, temporizador e desligado.

O modo manual se caracteriza por ser capaz de configurar a umidade relativa do ar desejada para que o umidificador seja ativado.

O automático se caracteriza por ativar o sistema até que a umidade chegue a 80%.

O temporizador se caracteriza por ser capaz de configurar o tempo desejado que o umidificador permanece ligado.

O desligado desativa o sistema.

1.2.2 Objetivos Específicos

1. Desenvolver um sistema em C++ que irá comandar o *hardware* de controle montado em cima do Arduino e conectado ao funcionamento do umidificador.
2. Adaptar interface do umidificador à do Arduino.
3. Programação do sistema, mostrando o acionamento dos dispositivos conforme dados adquiridos do ambiente.
4. Mostrar as informações de controle na própria tela do computador.

1.3 Motivação

A motivação deste projeto se originou nos alertas constantes sobre a baixa umidade do ar, no período de inverno, em Brasília. O uso de umidificadores é cada vez mais necessário e um sistema remoto para controlar o uso destes aparelhos facilitaria o dia a dia das pessoas.

1.4 Metodologia

Para obter o resultado esperado, a interface desenvolvida deve apresentar as informações da temperatura e umidade relativa do ar, e ser capaz de controlar a seleção dos módulos do umidificador; Automático, Manual, Temporizador e Desligado, por meio de comunicação *Bluetooth*. Estas informações são obtidas por *hardware* composto de sensores de temperatura e umidade relativa do ar. Neste cenário, como metodologia para desenvolver o trabalho proposto, são delineadas as seguintes etapas:

1 A primeira etapa é composta pela coleta da temperatura e umidade do ar e do nível da água no reservatório do umidificador. Esta coleta será realizada a partir de sensores instalados em *hardware* desenvolvido como parte do projeto.
2 A segunda etapa é composta pela transferência desta informação para o *software*. Realizada via *Bluetooth*.
3 A terceira etapa será a seleção do modo de operação do sistema. O *software* é responsável por enviar o modo que o *hardware* irá operar.
4 Por fim, se o nível da água ficar abaixo de um nível mínimo definido na instalação do sensor, o sistema entra em modo Desligado.

1.5 Resultados Esperados

Como resultado deste projeto, espera-se desenvolver um sistema que irá auxiliar no controle, de maneira automatizada, da umidade relativa do ar em um ambiente.

1.6 Estrutura do trabalho

A estrutura do trabalho se divide nos seguintes capítulos:

O capítulo 1 expõe uma breve introdução, motivação, objetivos e resultados esperados em relação ao projeto composto.

No capítulo 2 é apresentado o referencial teórico sobre umidade relativa dor ar, o umidificador, elementos que compõem o *hardware* e programação.

No capítulo 3 é apresentado o desenvolvimento do protótipo, mostrando descrição da construção e a descrição dos materiais utilizados.

O capítulo 4 apresenta os testes realizados no protótipo.

No capítulo 5 são apresentadas as conclusões do trabalho.

CAPÍTULO 2 REFERENCIAL TEÓRICO

Neste capítulo será abordada a teoria e o cenário que envolve este projeto. Funcionamento de umidificadores, a importância da umidade relativa do ar e uma introdução em relação aos principais componentes utilizados.

2.1 Elementos principais que compõem o protótipo

Neste item será abordada a teoria que envolve os principais componentes utilizados no desenvolvimento do protótipo. O uso de cada componente detalhado e suas funções no projeto serão descritos no capítulo 3, o de desenvolvimento.

2.1.1 Plataforma Arduino

Arduino é uma plataforma de prototipagem eletrônica de hardware livre e placa única (LEMOS, 2013). Projetado com um microcontrolador Atmega328 com suporte I/O embutido, uma linguagem de programação padrão, a qual tem origem em Wiring e é essencialmente C/C++ (ARDUINO, 2014).

O projeto Arduino originou-se na cidade de Ivrea, Itália, em 2005, por Massimo Banzi e David Cuartielles, com o intuito de interagir em projetos escolares. Desde seu lançamento, o Arduino se tornou um produto de sucesso por conta de sua facilidade de uso e durabilidade (MONK, 2013).

A IDE (*Integrated Development Environment*) é responsável pela comunicação entre o Arduino e o computador. Esse *software*, desenvolvido em Java, é executado no computador e possibilita a criação de *sketches*, ou códigos, para a placa Arduino. Depois de criado, é feito o *upload* do *sketch* para a placa, o código é traduzido para a linguagem C, e este novo código é transmitido a um compilador que realiza a tradução final dos comandos para uma linguagem compreendida pelo microcontrolador (BANZI, 2011).

Na Figura 2.1, é possível observar a placa do Arduino UNO, último modelo da série mais popular de placas Arduino, a qual é utilizada neste projeto. O

número 1 representa o conector USB (*Universal Serial Bus*), responsável pela comunicação do Arduino com o computador. O botão *RESET* representado pelo número 2 reinicializa o microcontrolador, começando a executar o código desde o início. Indicadas pelo número 3 estão as conexões digitais que podem ser uma entrada ou uma saída. Quando utilizada como saída, seu comportamento se assemelha aos conectores de alimentação elétrica de 5 volts. Entre os conectores digitais e de alimentação, existem dois conectores GND (*Ground* ou Terra) que possuem 0 volts. O número 4 representa o microcontrolador Atmega328. Contém: Processador, memória RAM (*Random Acess Memory*) para armazenar dados e memória EPROM (*Erasable Programmable Read Only Memory*) que armazena os programas desenvolvidos. O número 5 indica as entradas analógicas utilizadas como entradas e saídas digitais. Representado pelo número 6, estão os conectores de alimentação elétrica. Os conectores fornecem tensões de 5 volts e 3,3 volts conforme indicados na placa. E por fim, o número 7 é o conector de alimentação do Arduino (MONK, 2013).

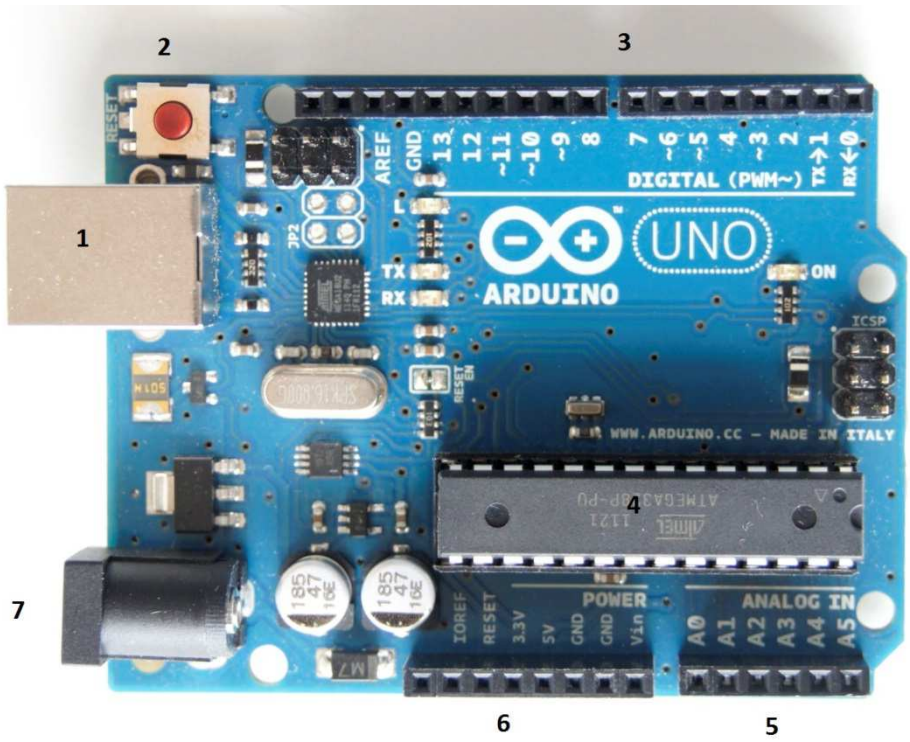


Figura 2. 1 - Placa Arduino UNO (Fonte: Autor)

2.1.2 Sensor de temperatura e umidade DHT11

Um sensor é um dispositivo que responde a um estímulo físico/químico de maneira específica e mensurável analogicamente. Os sensores de temperatura e umidade da família DHTxx são compostos por um sensor capacitivo para medir a umidade e por um termistor, que são dispositivos elétricos que têm sua resistência elétrica alterada termicamente. Ambos os sensores são calibrados nos laboratórios durante sua fabricação (ELETRONICS, 2013). A Figura 2.2 ilustra o sensor DHT11 que foi utilizado no projeto para coletar dados de temperatura e umidade.

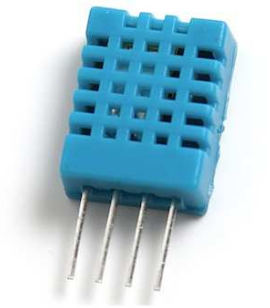


Figura 2. 2 - Sensor de umidade e temperatura (Fonte: www.inmotion.pt)

2.1.3 Sensor de BlueTooth HC-05

O *Bluetooth* é uma tecnologia desenvolvida que visava a substituição de cabos de sinal, mas atualmente permite até implementar redes de comunicação com vários dispositivos interconectados, tais como: computadores, fones de ouvido, celulares, impressoras, câmeras digitais, *tablets*, entre outros (HUGNEY, 2014).

O protocolo *Bluetooth* é complexo e composto por várias camadas, uma parte feita por *hardware* e outra pelo *software* embarcado no dispositivo de aplicação. Porém, o *Bluetooth* permite um protocolo denominado RFCOMM (*Radio Frequency Communications*), que emula uma comunicação serial padrão, permitindo

a troca de informações entre dispositivos como se fossem conectados por um cabo destinado à comunicação RS232.

A Operação EDR (*Enhanced Data Rate*) com modulação de 3Mbps com rádio transceptor em 2.4GHz, a qual emprega um algoritmo de alta velocidade de deslocamento pseudoaleatório de frequência (altera a frequência de trabalho continuamente) com uma chave de criptografia de 128 bits, torna a comunicação muito segura (HUGNEY, 2014).

A Figura 2.3 mostra o módulo *bluetooth* HC-05. Esse módulo integra um chip *Bluetooth* SPP (*Serial Port Protocol*) responsável por realizar o protocolo de porta serial, tornando a comunicação sem fio transparente ao usuário (HUGNEY, 2014).

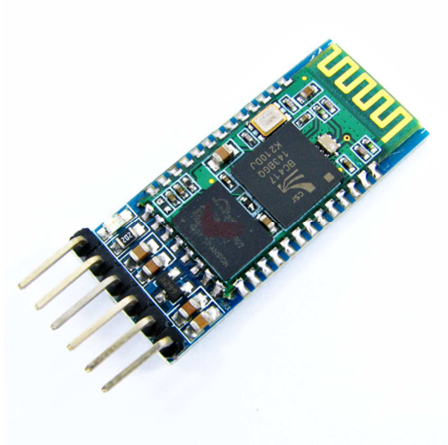


Figura 2. 3 - Módulo BlueTooth HC-05 (Fonte: blog.roman-mueller.ch)

2.1.4 Sensor de nível da água ICOS LA16M-40

Sensores de nível detectam nível de líquidos em reservatórios na altura em que forem instalados, com contato *ON/OFF* como saída. Também conhecidos como “chave de nível” ou “bóia de nível”, funcionam com contato *Reed Switch* (contatos hermeticamente selados que comutam com um campo magnético) e flutuador magnético (ICOS, 2014).

Para determinar um modelo de sensor ou chave de nível, é fundamental saber qual líquido se deseja detectar o nível. A Figura 2.4 ilustra o sensor LA16M-40 utilizado neste projeto. O LA16M-40 é indicado para água, óleo e combustíveis (ICOS, 2014).

Figura 2. 4 - Sensor de nível da água (Fonte:http://www.icos.com.br/SensorDeNivel/LA16M40/)

2.1.5 Relé

Tendo o surgimento em torno do século XIX, o Relé é um dispositivo eletromecânico, formado por um magneto móvel, que desloca unindo dois contatos metálicos funcionando como um interruptor. A movimentação física desse interruptor ocorre quando a corrente elétrica percorre as espiras da bobina do relé, criando um campo magnético que atraia a alavanca responsável pela mudança do estado dos contatos (SANTOS, 2012).

A Figura 2.5 ilustra o relé 1POLO 5V utilizado no projeto para acionar o funcionamento do umidificador.

Figura 2. 5 Relé 1POLO 5V (Fonte:http://www.huinfinito.com.br/reles/361-rele-1polo-5v.html)

2.1.6 Transistor 2N3904

Criado na década de 1950, o transistor é um componente eletrônico responsável pela revolução da eletrônica, segundo (MORIMOTO, 2004), na década de 1960. A importância do transistor é relacionada a ter tornado possível a disseminação dos computadores e equipamentos eletrônicos, pelo seu baixo preço e sua possibilidade de ser produzido em enormes quantidades utilizando técnicas simples. São utilizados como amplificadores e interruptores de sinais elétricos sendo conveniente salientar que é praticamente impossível encontrar um circuito integrado que não possua, internamente, centenas, milhares ou mesmo milhões de transistores.

Um transistor é composto, basicamente, de três filamentos, base, emissor e coletor, como pode ser visto na Figura 2.6, para o caso de um transistor NPN. O emissor é o polo positivo, o coletor o negativo, e a base é quem controla o estado do

transistor. Quando se aplica uma tensão na base, normalmente maior que 0,7 Volts, o circuito é fechado e é estabelecida a corrente entre o emissor e o coletor.

Figura 2. 6 - Transistor com descritivo dos filamentos (Fonte: www.instructables.com)

A Figura 2.7 ilustra o transistor 2N3904 utilizado no projeto. O transistor funciona como interruptor para o acionamento do relé, recebendo uma tensão proveniente do Arduino em sua base e controlando a corrente para o relé.

Figura 2. 7 - Transistor 2N3904 (Fonte:Autor)

2.1.7 Resistor

Conhecidos como componentes básicos de um circuito elétrico, os resistores são utilizados para controlar a intensidade de corrente que passa pelos diversos componentes bem como controlar a tensão aplicada em cada parte do circuito.

O funcionamento dos resistores está baseado na resistência elétrica que todos os materiais possuem. A resistência, que serve como relação de tensão para a corrente, é medida em Ohms, unidade SI (Sistema Internacional). Um componente tem a resistência de 1 Ohm se uma tensão de 1 volt no componente fizer com que percorra, pelo mesmo, uma corrente com a intensidade de 1 Ampére.

Devido ao seu tamanho reduzido, foi criado um código de cores que é impresso no resistor para indicar sua resistência.

A Figura 2.8 representa o resistor de 1k2 utilizado no projeto para limitar a tensão passada para o transistor.

Figura 2. 8 - Resistores (Fonte: www.comptotal.com.br)

2.1.8 Diodo 1N4007

O diodo é um componente que possibilita o fluxo de corrente em uma única direção. Caso a corrente tente voltar na direção contrária, ele a impede. É composto por uma junção P-N. Possui uma faixa branca perto do terminal negativo. Esta faixa representa uma barreira. A eletricidade corre através do diodo pelo terminal que não tem a barreira. Quando a voltagem é revertida e tenta fluir pelo lado que tem a faixa, há uma interrupção na corrente (MCROBERTS, 2011).

A Figura 2.9 ilustra o diodo utilizado no projeto para que sirva como uma barreira de corrente, não permitindo que flua para a placa do Arduino, possivelmente queimando-a.

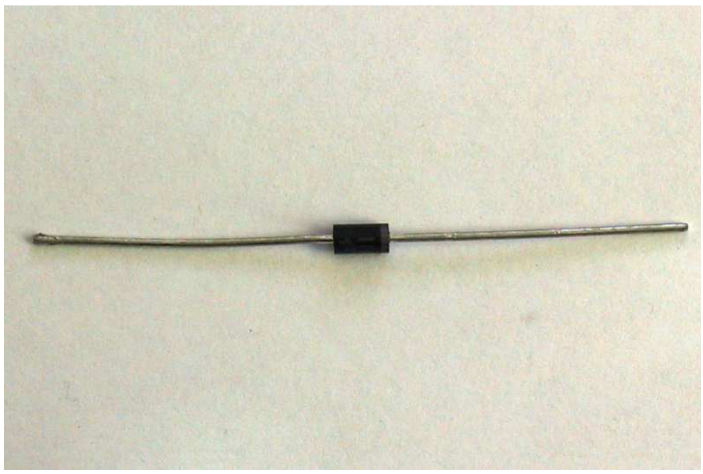


Figura 2. 9 - Diodo 1N4007 (Fonte: Autor)

2.2 Programação

O homem sempre procurou, desde o início de sua existência, criar máquinas que o auxiliasse em seu trabalho, diminuindo esforço e economizando tempo. O computador, até o momento, vem se mostrando ser uma das mais versáteis, rápidas e seguras, dentre estas máquinas.

O computador pode auxiliá-lo em qualquer tarefa. É trabalhador, possui muita energia, mas não tem iniciativa, nenhuma independência, não é criativo nem inteligente, por isso precisa receber instruções nos mínimos detalhes (ASCENCIO, 1999).

A finalidade de um computador é de realizar a tarefa de processamento de dados, isto é, receber dados por um dispositivo de entrada (por exemplo, teclado, *mouse*, *scanner*, entre outros), realizar operações com esses dados e gerar um resposta que será expressa em um dispositivo de saída (por exemplo, impressora, monitor de vídeo, entre outros) (ASCENCIO, 1999).

Portanto, um computador possui duas partes diferentes que trabalham juntas: o *hardware*, composto por partes físicas, e o *software*, composto pelos programas. Para criar ou desenvolver um *software* para realizar determinado tipo de processamento de dados, é necessário escrever um programa ou vário programas interligados. No entanto, para que o computador compreenda e execute esse programa, deve-se escrevê-lo usando uma linguagem que tanto o computador quanto o criador de *software* entendam. Essa linguagem é chamada de Linguagem de Programação (ASCENCIO, 1999).

O conceito central da programação é o de algoritmo (LAGES, 1988). Programar é basicamente construir algoritmos. Programas são formulações concretas de algoritmos abstratos, baseados em representações e estruturas específicas de dados.

Outro aspecto fundamental da construção de programas são as estruturas de dados usadas no algoritmo para representar as informações do problema a ser resolvido (LAGES, 1988).

2.2.1 Algoritmos

Para que seja possível entender o conceito de algoritmo, é importante que o conceito de ação seja fixado. Ação é um acontecimento que, a partir de um estado inicial, após um período de tempo finito, produz um estado final previsível e

bem definido (FARRER, BECKER, *et al.*, 1989). Dessa forma, algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão finita de ações.

Exemplo de algoritmo:

```
Algoritmo
   repita
         invente um problema
         escreva um algoritmo para sua resolução
         se estiver cansado
               então interrompa
         fim se
         mostre os algoritmos feitos para um colega
Fim algoritmo
```

Um algoritmo geralmente se destina a resolver um problema. Fixa um padrão de comportamento a ser seguido, uma norma de execução a ser trilhada, para se atingir como resultado, a solução de um problema (FARRER, BECKER, *et al.*, 1989).

2.2.2 Estruturas de Dados

Para se resolver um problema por meio de um programa, a primeira providência é conceber um algoritmo adequado. No entanto, a eficiência de um algoritmo qualquer está intimamente relacionada à disposição, na memória, dos

dados que são tratados pelo programa. Por exemplo, se frequentemente enfrentarmos o problema de descobrir os números de telefones de nossos conhecidos, é conveniente dispor de uma relação de números, organizada em uma agenda. Se a organização for feita por ordem alfabética, a agenda de fato ajuda. Se, porém, organizássemos nossa agenda pela ordem de altura das pessoas, com raras exceções, a agenda se tornaria difícil de manusear (PÍCCOLO, 2000).

Estruturas de dados são formas de distribuir e relacionar os dados disponíveis, de modo a tornar mais eficiente os algoritmos que manipulam esses dados. Por exemplo:

Problema Manipular um conjunto de fichas de um fichário. Solução Organizar as fichas em ordem alfabética. Operações possíveis Inserir ou retirar uma ficha, procurar uma ficha, etc. Estrutura de Dados Correspondente LISTA – sequência de elementos dispostos em ordem.

Outro exemplo:

Problema

Organizar as pessoas que querem ser atendidas num guichê.

Solução

Colocar as pessoas na fila.

Operações possíveis

À medida que uma pessoa é atendida no guichê, outra entra no final da fila. Não é permitido furar a fila.

Estrutura de Dados Correspondente

FILA – sequência de elementos dispostos em ordem com uma regra para a entrada e saída dos elementos.

O estudo de estruturas de dados é parte fundamental para o desenvolvimento de programas e algoritmos. Assim como um número pode ser representado em vários sistemas diferentes, também um conjunto de dados relacionados entre si pode ser descrito através de várias estruturas de dados distintas.

Quando o programador cria um algoritmo para solucionar um problema, ele também cria uma estrutura de dados que é manipulada pelo algoritmo. A escolha de uma determinada estrutura pode afetar substancialmente a quantidade de área de armazenamento requerida para o processamento bem como o tempo deste processamento.

É, portanto, de grande importância o estudo de diferentes estruturas que possam ser utilizadas eventualmente na resolução de um problema, de maneira a simplificar a sua implementação prática, como é o caso do projeto aqui desenvolvido.

2.2.3 Linguagem C++

Com o princípio de escrever e manter o sistema operacional UNIX, a linguagem C foi desenvolvida por Dennis Ritchie, dos AT&T Bell Laboratories, na década de 70 (SAVITCH, 2004).

Por ser uma linguagem de alto nível com muitos recursos de linguagem de baixo nível, C possui vantagens e desvantagens por sua peculiaridade. Como o Assembly, a linguagem C pode manipular diretamente a memória do computador. Por outro lado, ela possui recursos de linguagem de alto nível, o que a torna mais fácil de ler e escrever que o Assembly. No entanto, para alguns programas, C não é tão fácil de entender quanto outras linguagens. Além disso, não possui tantas verificações automáticas quanto outras linguagens de alto nível.

Para superar essas e outras desvantagens, segundo (SAVITCH, 2004), Bjarne Stroustrup, dos AT&T Bell Laboratories, desenvolveu o C++ no início da década de 80. A maior parte de C é subconjunto da C++, no entanto, ao contrário de C, C++ possui recursos para classes e, portanto, pode ser usada para a programação orientada a objetos.

2.2.4 GUI (Graphic Use Interface)

GUI (*Graphical User Interface*) é um tipo de interface do usuário que permite a interação com dispositivos digitais através de elementos gráficos como ícones e outros indicadores visuais, em contraste com a interface de linha de comando.

A iteração é feita geralmente através de um mouse ou teclado, permitindo que o usuário selecione símbolos representativos de forma a obter algum resultado prático.

A Figura 2.10 mostra a interface desenvolvida no projeto. Nela as informações de temperatura e umidade, como também o nível do reservatório do umidificador, são disponibilizadas. Abaixo dos sensores, é possível notar os controles de modos de operação do sistema.

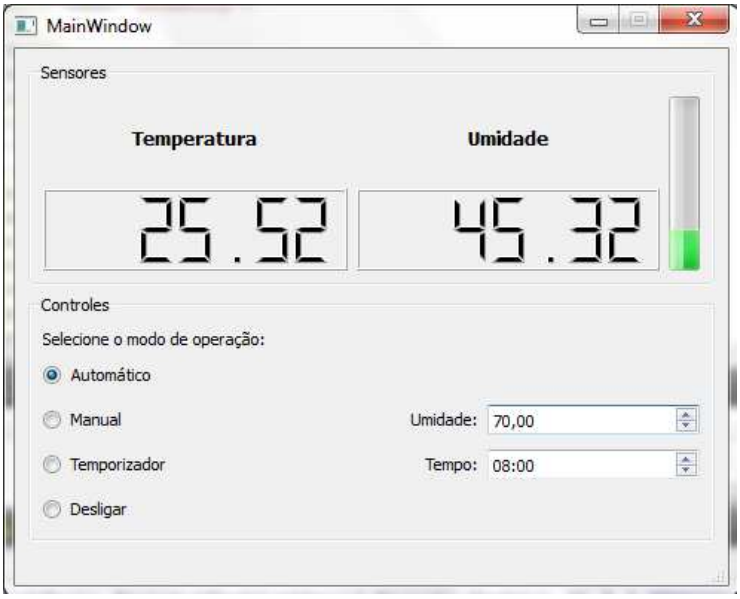


Figura 2. 10 - Interface do usuário desenvolvida para o projeto (Fonte: Autor)

2.2.5 QT Creator

O desenvolvimento do que eventualmente iria se transformar no QT Creator começou em 2007, no escritório da TrollTech GmbH, com o nome de Workbench e posteriormente Project Greenhouse (TESKE, 2009).

Qt Creator é uma IDE multi-plataforma e traz consigo a Qt SDK. A Figura 2.11 ilustra as ferramentas que o Qt proporciona para desenvolver aplicações *desktop* e celulares.

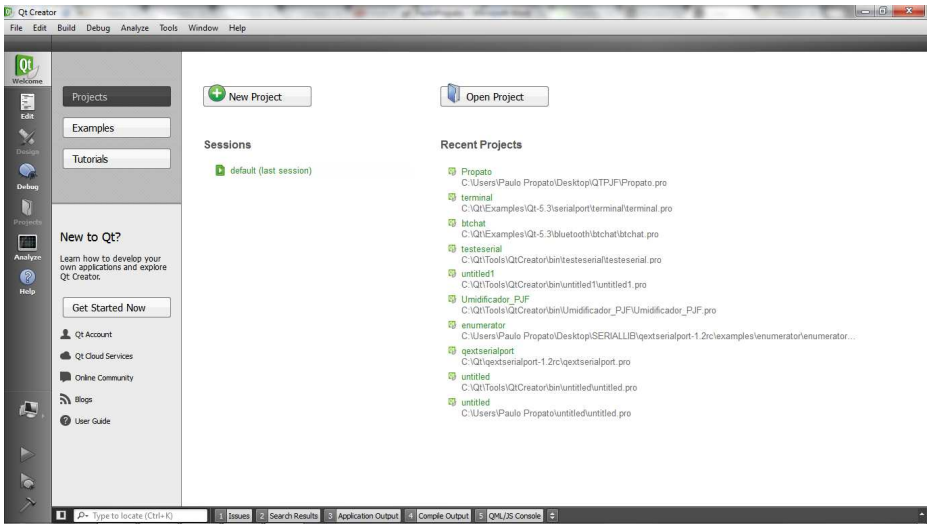


Figura 2. 11 - Página de entrada da IDE do QT Creator (Fonte: Autor)

Para ser capaz de construir e rodar aplicações, o Qt Creator precisa das mesmas informações que um compilador precisaria. Essas informações são especificadas no *Project build* e *run settings*. A criação de um Projeto no Qt Creator permite que você agrupe arquivos, adicione padronização no passo a passo para desenvolvimento, inclua formas e arquivos de recursos e especifica as configurações da aplicação. A Figura 2.12 ilustra o que foi dito.

Figura 2. 12 - QT Creator Overview
(Fonte:http://developer.nokia.com/community/wiki/Qt_Creator)

O Qt Creator possui a parte de edição de código e um modelador de interface gráfica (Qt Designer) para desenvolvimento de GUIs. O Editor, ilustrado pela Figura 2.13, entende linguagens de programação C++ e QML, auxiliando na formatação do código, completando comandos, demonstra o erro na linha e mensagens de alerta, entre outros. O Designer, ilustrado pela Figura 2.14, proporciona as ferramentas necessárias para se criar GUIs através de *widgets* disponibilizados pelo QT. Formas e *widgets* criados no Designer são integradas com códigos programados facilitando o agendamento de função daquele elemento gráfico.

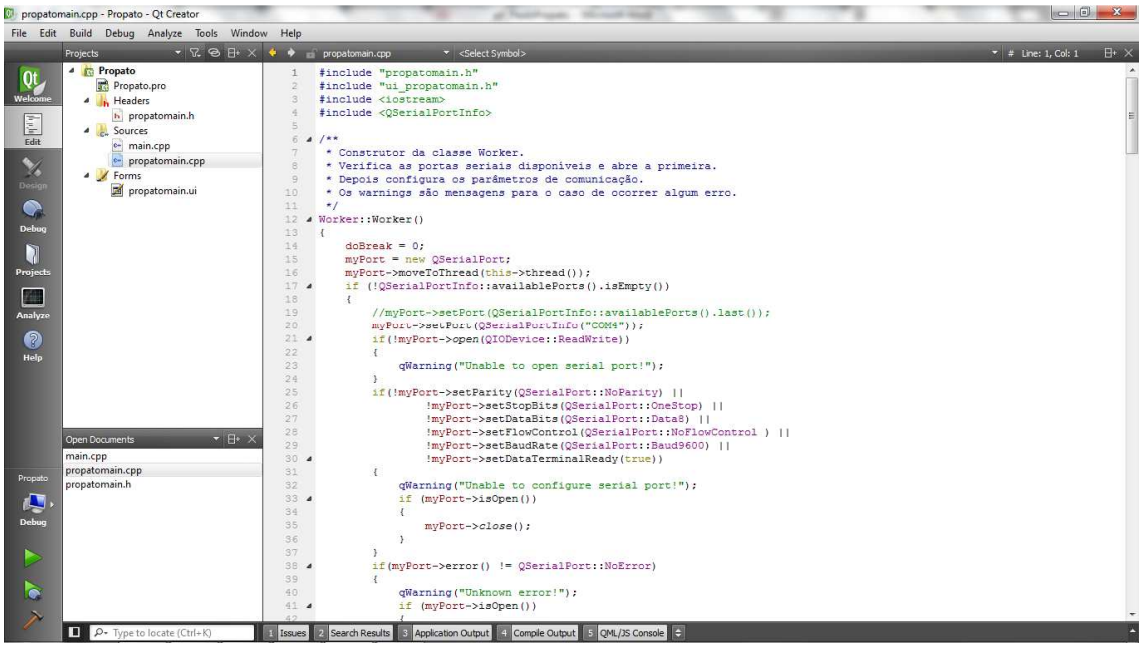


Figura 2. 13 - Editor do QT Creator (Fonte: Autor)

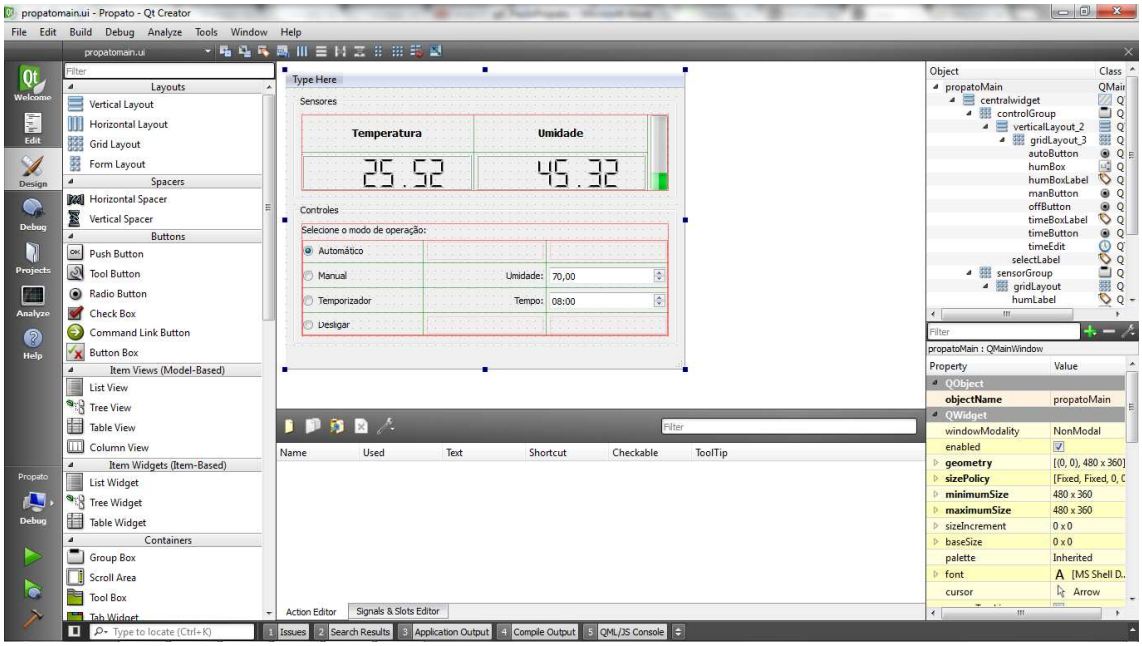


Figura 2. 14 - Designer do QT Creator (Fonte: Autor)

Se for instalado como parte do Qt SDK, a ferramenta de depuração GNU Symbolic Debbuger é instalada automaticamente e pronta para ser usada quando um novo projeto for criado. No entanto, podem-se utilizar outros sistemas de depuração, configurando para, por exemplo, utilizar ferramentas do *Windows*.

2.3 Umidade Relativa do Ar

Umidade relativa é relação entre a pressão de vapor do ar (medida em pascal) e a pressão de vapor do ar obtida em condições de equilíbrio ou saturação sobre uma superfície de água líquida ou gelo (WEATHERLY, 2008). Convencionalmente é denotada em porcentagem. Em outras palavras, pode-se dizer que umidade relativa do ar é a relação entre a quantidade de água existente no ar e a quantidade máxima que poderia haver na mesma temperatura.

A umidade relativa tem papel importante em nossa sensação de conforto. Se a umidade relativa for de 100%, significa que a água não vai evaporar, pois o ar

já está saturado. Nosso organismo depende da evaporação da água através da pele para esfriar.

A Figura 2.15 ilustra quanto uma determinada temperatura vai nos parecer quente (sensação térmica) em vários níveis de umidade relativa.

Umidade relativa	Temperatura do ar (graus Farenheit)										
	70	75	80	85	90	95	100	105	110	115	120
0%	64	69	73	78	83	87	91	95	99	103	107
10%	65	70	75	80	85	90	95	100	105	111	116
20%	66	72	77	82	87	93	99	105	112	120	130
30%	67	73	78	84	90	96	104	113	123	135	148
40%	68	74	79	86	93	101	110	123	137	151	
50%	69	75	81	88	96	107	120	135	150		
60%	70	76	82	90	100	114	132	149			
70%	70	77	85	93	106	124	144				
80%	71	78	86	97	113	136	157				
90%	71	79	88	102	122	150	170				
100%	72	80	91	108	133	166					

©2001 HowStuffWorks

Figura 2. 15 - Índice de umidade relativa x temperatura (Fonte: http://static.hsw.com.br/gif/humidifier-heat-index.gif)

Se a umidade relativa for de 100%, sentimos mais calor que a temperatura real porque nosso suor não consegue evaporar. No entanto, se a umidade relativa estiver baixa, sentimos menos calor do que a temperatura real devido à rápida evaporação do suor.

2.4 Umidificador

Umidificador é um aparelho cuja função básica é manter a umidade relativa do ar dentro dos níveis recomendados para o conforto humano. O tipo mais simples de umidificador é o umidificador evaporativo. Seu funcionamento consiste em um reservatório captar a água fria e a introduzir em um recipiente. A água é absorvida por um recipiente com parede porosa e um ventilador força o ar passar

nesse recipiente que fica úmido. Conforme o ar passa, um pouco de água evapora (BRAIN, 2010).

A Figura 2.16 mostra o umidificador ultra-sônico utilizado no projeto. Seu princípio de funcionamento é através de um sistema ultra-sônico. Utiliza um cristal piezoelétrico, que imerso em um reservatório de água, vibra a uma frequência muito alta (cerca de 1,6 milhões de oscilações por segundo), gerando uma fina e homogênea névoa de vapor de água frio e inodoro (BRAIN, 2010).

Figura 2. 16 - Umidificador utilizado no projeto (Fonte: Autor)

Os aspectos teóricos e técnicos abordados neste capítulo são fundamentais para o desenvolvimento da solução, conforme será visto nos próximos capítulos.

CAPÍTULO 3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo será abordado o desenvolvimento do protótipo idealizado. Passos da construção, *debugs* e soluções aplicadas são alguns dos tópicos tratados.

3.1 Visão Geral do Projeto

A visão geral do projeto é ilustrada na Figura 3.1. Basicamente, o funcionamento do sistema é iniciado a partir do momento em que o usuário seleciona o modo que o irá atuar, por meio da interface. A interface, então, repassa a instrução para o *hardware* por comunicação *bluetooth* e ativa o funcionamento do umidificador.

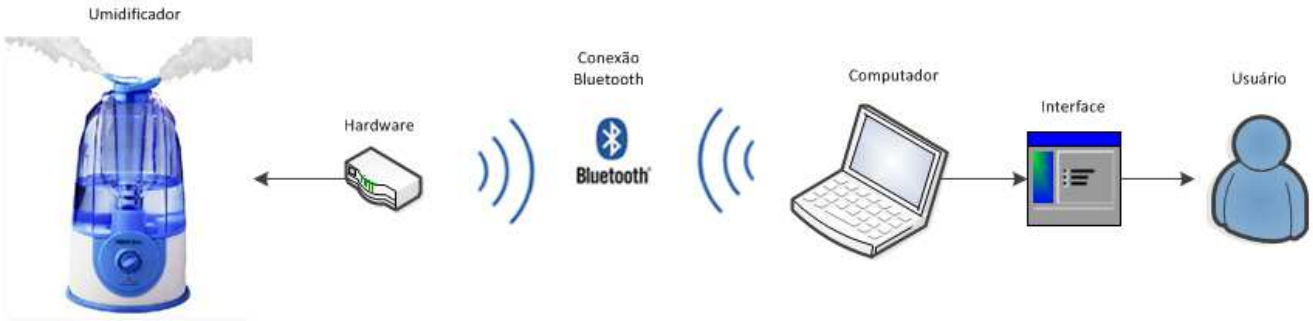


Figura 3. 1 - Visão geral do projeto. (Fonte: Autor)

O projeto é dividido em duas partes. A Figura 3.2 ilustra a primeira parte, constituído pelo *hardware* desenvolvido.

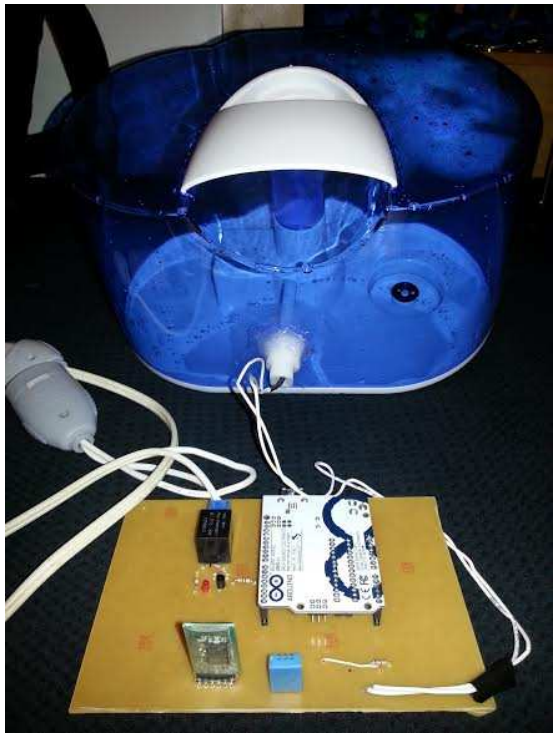


Figura 3. 2 - Hardware desenvolvido (Fonte: Autor)

A segunda parte, vide Figura 3.3, é formada pela interface de comunicação com o usuário.

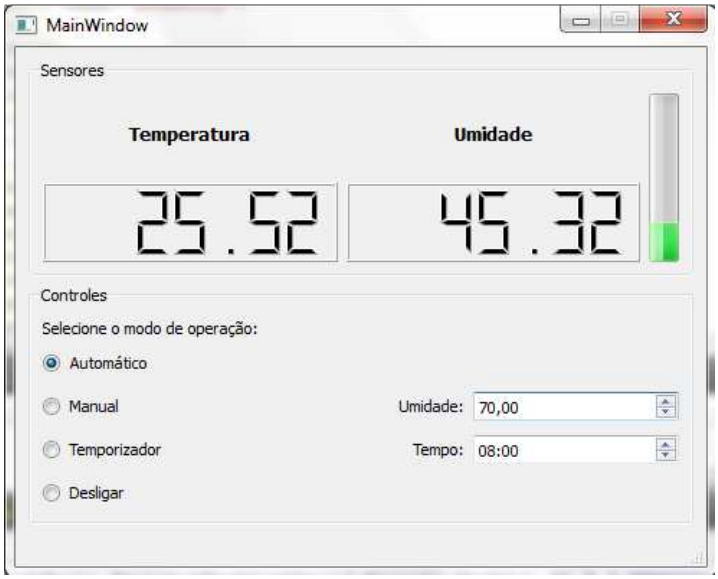


Figura 3. 3 - Interface de Comunicação (Fonte: Autor)

O diagrama a seguir, Figura 3.4, retrata as etapas do processo de seleção do modo de funcionamento através da interface do usuário, comunicação entre *software* e *hardware*, captação dos dados e disponibilização, e acionamento do umidificador.

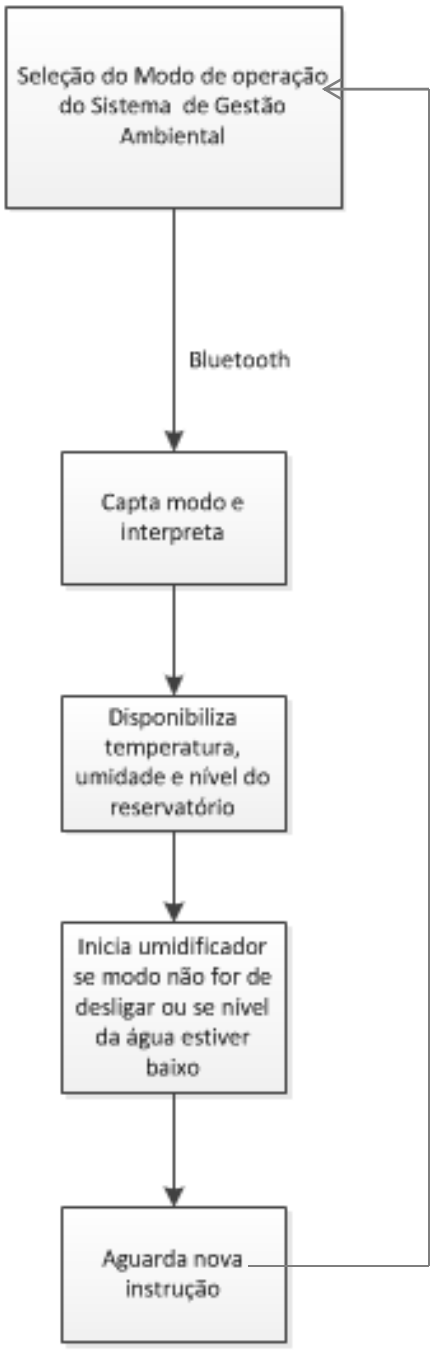


Figura 3. 4 - Diagrama de blocos do projeto (Fonte: Autor)

O marco inicial é a seleção do modo de operação do sistema: Manual, Automático e Temporizador. Após essa etapa, o sistema envia o modo para o Arduino por *bluetooth* ativando seu funcionamento. O Arduino retorna os dados de umidade do ar, temperatura do ar, o nível da água do repositório e ativa o funcionamento do umidificador. Este permanece ligado até o modo Desligado. Outros fatores que desligam o funcionamento do umidificador são o nível da água abaixo do mínimo, a umidade relativa atingir o limite repassado no modo manual ou a umidade relativa do ar chegar a 80%. O quadro a seguir, descreve a estrutura de dados utilizada para desenvolver o sistema.

Problema Interpretar instrução enviada pela porta serial. Solução Abrir conexão com a porta serial e interpretar o dado. Operações possíveis À medida que um dado é interpretado no programa, outro entra na fila da porta serial. Não é permitido furar a fila. Estrutura de Dados Correspondente A biblioteca do Qt Creator – QtSerialPort – utiliza FILA – sequência de elementos dispostos em ordem com uma regra para a entrada e saída dos elementos.

3.2 Descrição das Etapas

3.2.1 Montagem do Hardware

O *hardware* foi montado em cima de uma plataforma de prototipagem eletrônica, Arduino Uno. A Figura 3.5 ilustra, representativamente, os pinos utilizados, os componentes e sensores.

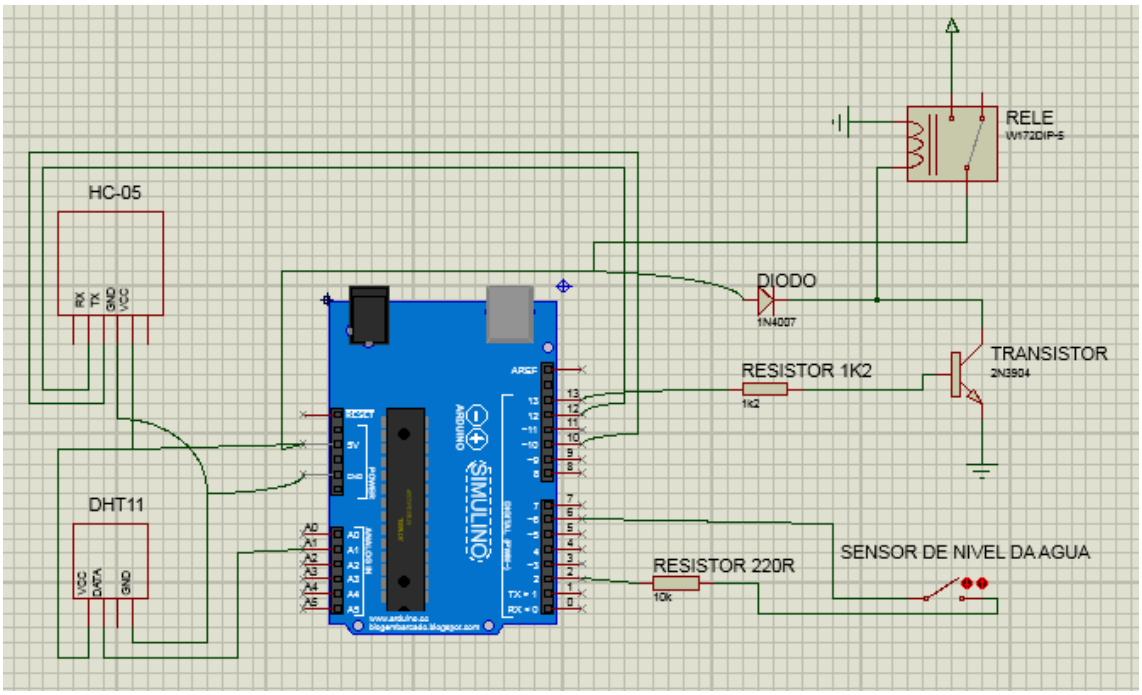


Figura 3. 5 - Esquemático dos componentes no Proteus (Fonte: Autor)

Do lado esquerdo do Arduino Uno (placa azul na Figura 3.5) encontram-se os sensores de *bluetooth* e umidade e temperatura, HC-05 e DHT11, respectivamente. O sensor HC-05 é conectado nos pinos 10 e 12 da placa e saída 5v e GND. O sensor DHT11 é conectado no pino A1 (analógico 1) da placa, e também ao 5v e GND. Para o esquema do relé, foi utilizado o transistor, diodo e resistor para o seu funcionamento. O esquema está conectado ao pino 13 da placa que quando acionado, a corrente limitada pelo resistor, passa para o transistor ativando a passagem de corrente para o relé funcionar. O diodo funciona como um

bloqueio da corrente para a placa do Arduino não sofrer avarias. Por último, o sensor de nível é conectado ao pino 8 e a um resistor de *pull-up* para captação do nível do reservatório.

3.2.2 Escolha do modo de funcionamento

Para a comunicação com o Arduino, foi desenvolvido uma GUI (*Graphical User Interface)* em C++ utilizando o Qt Creator como IDE. O objetivo do GUI é permitir ao usuário a interação com dispositivos digitais através de elementos gráficos como ícones e outros indicadores visuais.

A Figura 3.6 mostra o GUI desenvolvido no QT Creator. Representado pelos números 1 e 2, estão os *displays* de temperatura e umidade respectivamente. Seus valores são atualizados a partir do momento em que se escolhe um modo de funcionamento. Os números 3, 4, 6 e 8 representam os modos que o sistema possui. A particularidade dos modos Manual e Temporizador, são representados pelos números 5 e 7. O número 5 ilustra a umidade desejada para o ambiente, ou seja, o umidificador irá ficar ligado até atingir esse valor ou, o modo Desligar seja acionado, ou ainda, o reservatório chegue a um nível muito baixo, representado pelo número 9. O nível de água é representado apenas com duas situações, vazio ou cheio. Seguindo a mesma lógica do modo Manual, o número 7 representa a quantidade de horas que o umidificador ficará ligado se o modo Temporizador for acionado. Também desligando se houver a troca de modo para Desligado, ou se o nível da água ficar muito baixo.

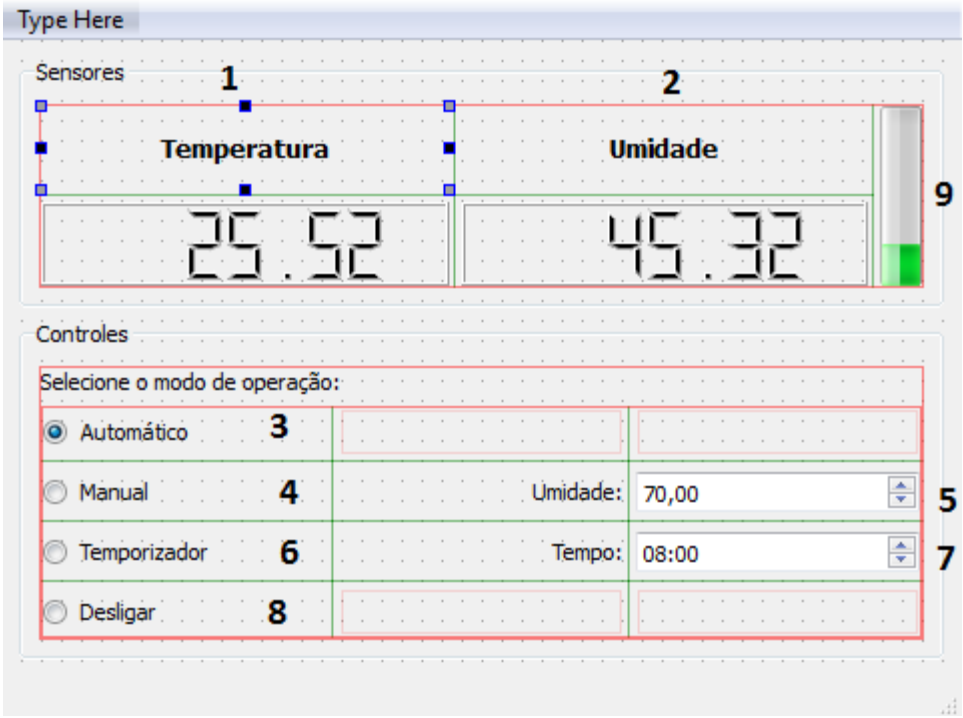


Figura 3. 6 GUI desenvolvido. (Fonte: Autor)

O código, em C++, de cada modo selecionado pode ser observado logo abaixo:

```
/**
 * Comandos da seleção do botão de modo automático;
 */
void propatoMain::on_autoButton_clicked()  //Quando o modo automático é selecionado
{
    this->timerStatus = false;   //seta o timer para falso
    this->myTimer->stop();       //para o timer
    this->mode = 'A';      //atribui para a variável mode a string “A”
```

```
    this->humidity = 80;          //atribui para a variável humidity o number 80
}

/**
 * Comandos da seleção do botão de modo manual;
 */
void propatoMain::on_manButton_clicked()          //quando o modo manual é acionado
{
    this->timerStatus = false;   // seta o timerStatus para falso
    this->myTimer->stop();       //para o timer
    this->mode = 'B';      //atribui a string “B” para a variável mode
    this->humidity = this->ui->humBox->value();     //captura o valor da umidade que esta na caixa humBox e atribui para a variável humidity.
}

/**
 * Comandos da seleção do botão de modo temporizador;
 */
void propatoMain::on_timeButton_clicked()          //quando o botão de temporizador for acionado
{
    if (this->timerStatus = false)         //condição do timerStatus para falso
```

```
            {
                connect(this->myTimer, SIGNAL(timeout()), this,
SLOT(on_offButton_clicked()));
                this->myTimer->setSingleShot(true);  //aciona o timer uma única vez
                this->myTimer->start(abs(this->ui->timeEdit-
>time().msecsTo(QTime())));       //checa o valor repassado na interface para
repassar para o timer
            }
            this->timerStatus = true;     //seta timerstatus como true
            this->mode = 'A';      //atribui a string “A” para a variável mode
            this->humidity = 80;           //atribui o number 80 para a variável humidity
        }


        /**
         * Comando do botão de desligar;
         */
        void propatoMain::on_offButton_clicked()   //quando acionado o botão de
desligar
        {
            this->timerStatus = false;   //seta o timerstatus para falso
            this->myTimer->stop();       //para o timer
            this->mode = 'D';      //atribui “D” para a variável mode
            this->humidity = 0;   //atribui 0 para a variável humidity
        }
```

3.2.3 Comunicação com Arduino

Pode-se observar que o sistema possui diferentes modos de funcionamento. No projeto, foi utilizado o *bluetooth* para realizar a comunicação entre Arduino e s*oftware*.

O protocolo *bluetooth* é complexo e composto por várias camadas. Porém, o *bluetooth* permite um protocolo denominado RFCOMM, que emula uma comunicação serial padrão, permitindo a troca de informações entre dispositivos como se fossem conectados por um cabo destinado à comunicação RS232. O ponto inicial, então, é ter um *software* de *bluetooth* instalado no computador a ser utilizado.

No projeto, foi utilizado o *bluetooth* HC-05 que pode servir como mestre e escravo. O segundo passo para a conexão foi configurar o *bluetooth* como modo escravo, pois o *bluetooth* do computador agiria como mestre, requisitando serviços, e o do Arduino como escravo, provendo serviços. Para isso, foi necessário criar um *sketch* com o seguinte código de configuração, configurando o *bluetooth* não apenas para agir como escravo mas também alterando algumas outras configurações que estão comentadas no código:

```
void configuraBluetooth()
{
 //configurando a taxa de comunicação do Bluetooth com o microcontrolador Arduino em 115200bps(baud).
 bluetoothSerial.begin(115200);

 //configurando o Bluetooth para operar no modo escravo.
 bluetoothSerial.print("\r\n+STWMOD=0\r\n");

 //configuando a permissão de emparelhamento.
```

```
  bluetoothSerial.print("\r\n+STOAUT=1\r\n");

  //configurando o bloqueio de emparelhamento.
  bluetoothSerial.print("\r\n+STAUTO=0\r\n");

  //configurando o código de emparelhamento (PINCODE) com: 0000
  bluetoothSerial.print("\r\n+STPIN=0000\r\n");

  //configuando a solicitação de pino de entrada.
  bluetoothSerial.print("\r\n+RTPIN=0000\r\n");

  delay(2000); //aguarda um tempo em milissegundos.

  //configurar o Bluetooth para ser reconhecido como escravo.
  bluetoothSerial.print("\r\n+INQ=1\r\n");

  delay(2000); //aguarda um tempo em milissegundos.

  //descarrega a serial.
  bluetoothSerial.flush();
}
```

Feita a configuração, o terceiro passo foi reconhecer o *bluetooth* do Arduino com o do computador. O *driver* GL-6B ATSPP que, corresponde ao do *bluetooth,* foi instalado e disponibilizado na porta serial COM6 conforme Figura 3.7.

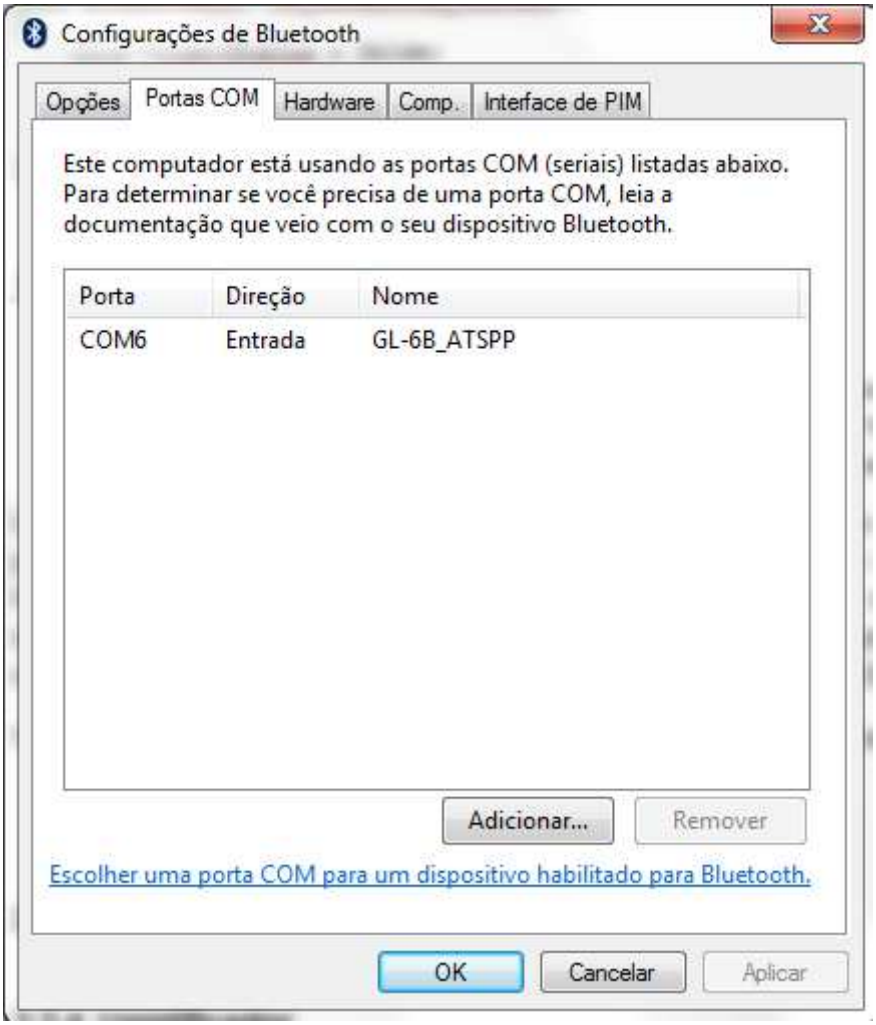


Figura 3. 7 Porta serial do *bluetooth* (Fonte: Autor)

O quarto e último passo para estabelecer a conexão foi configurar o *software* para que capte informações da porta serial. Dessa forma, foi desenvolvido um código para verificar as portas seriais disponíveis e abrir a COM4. Depois, configurar os parâmetros de comunicação e *warnings* para o caso de algum erro ocorrer. A codificação, desenvolvida como parte desse projeto, pode ser observada a seguir:

```
Worker::Worker()        //método Worker da classe Worker
{
    doBreak = 0;        //seta 0 na variavel dobreak
    myPort = new QSerialPort;        //criando função que reconhece comunicação serial
    myPort->moveToThread(this->thread()); //move função para a thread
    if (!QSerialPortInfo::availablePorts().isEmpty())        //condição de se não encontrar nenhuma porta serial aberta para conexão
    {
        myPort->setPort(QSerialPortInfo("COM4")); //seta a porta COM4
            if(!myPort->open(QIODevice::ReadWrite))      //condição de se não conseguir conectar
            {
                qWarning("Não foi possível abrir a porta!");      //aviso de não foi possível conectar
            }
    //condição se abaixo checa se foi possível configurar a porta ou não e caso não tenha sido possível, um aviso aparece
            if(!myPort->setParity(QSerialPort::NoParity) ||
                !myPort->setStopBits(QSerialPort::OneStop) ||
                !myPort->setDataBits(QSerialPort::Data8) ||
                !myPort->setFlowControl(QSerialPort::NoFlowControl ) ||
```

```
                !myPort->setBaudRate(QSerialPort::Baud9600) ||
                !myPort->setDataTerminalReady(true))
        {
            qWarning("Nao foi possivel configurar porta!");
            if (myPort->isOpen())
            {
                myPort->close();
            }
        }
```

//condição se abaixo checa se conexão teve erro ou não e lança um aviso case tenha erro.

```
        if(myPort->error() != QSerialPort::NoError)
        {
            qWarning("Erro desconhecido!");
            if (myPort->isOpen())
            {
                myPort->close();
            }
        }
    }
```

Else //caso não tenha nada conectado, aviso de nada conectado na porta serial

```
    {
```

```
            qWarning("Nao existe nada conectado!");
        }
    }
```

3.2.4 Repasse de dados e acionamento do umidificador

Após a comunicação ser estabelecida e o sistema enviar o modo, o Arduino recebe e interpreta esse modo. Foi desenvolvido o seguinte código para o recebimento do modo:

```
void loop(){       //função de repetição

    float h = dht.readHumidity();      //captação da umidade
    float t = dht.readTemperature();//captação da temperatura
    delay(50);

    if (isnan(t) || isnan(h))     //condição para verificar se houve erro na captação dos dados
    {
    bt.println("Failed to read from DHT");
    }

    else{

    if ( off == '0'){        //verifica se variável off é 0 ou 1
```

```
char Mode;          //declara variável Mode
if (bt.available()){  //condição se o serial do Bluetooth esta disponivel

Mode = bt.read(); //leitura do modo
bt.println(Mode);   //imprime leitura
}

delay(50);

if( Mode == 'A'){    //se modo for A, inicia modo automatico
   Humidity = GetStringNumber();       //recebe umidade
   digitalWrite(RELE, HIGH);     //aciona relé para umidificador
   bt.print(h);        //repassa umidade
   bt.print(" ");
   bt.print(t);        //repassa temperatura
   bt.print(" ");
   bt.println(digitalRead(IN));     //faz a leitura do sensor de nível
   if ( h == 80.00 || digitalRead(IN) = 0)          //condição   se   umidade
passar de 80 ele desliga o umidificador ou nível do reservatório abaixar
     off = '1';
}
```

```
if( Mode == 'B'){    //modo B segue a mesma lógica do modo A
    Humidity = GetStringNumber();
    digitalWrite(RELE, HIGH);
    bt.print(h);
    bt.print(" ");
    bt.print(t);
    bt.print(" ");
    bt.println(digitalRead(IN));
    if ( h >= Humidity || digitalRead(IN) = 0)     //condição se a umidade do ambiente for maior ou igual que a umidade repassada pelo usuário ele desliga o umidificador ou nível do reservatório abaixar
      off = '1';
}

 if( Mode == 'D'){   //modo D segue mesma lógica que A e B
    off = '1';
 }
}
else
{
 digitalWrite(13, LOW);
 off = '0';
 }
```

```
}
}
```

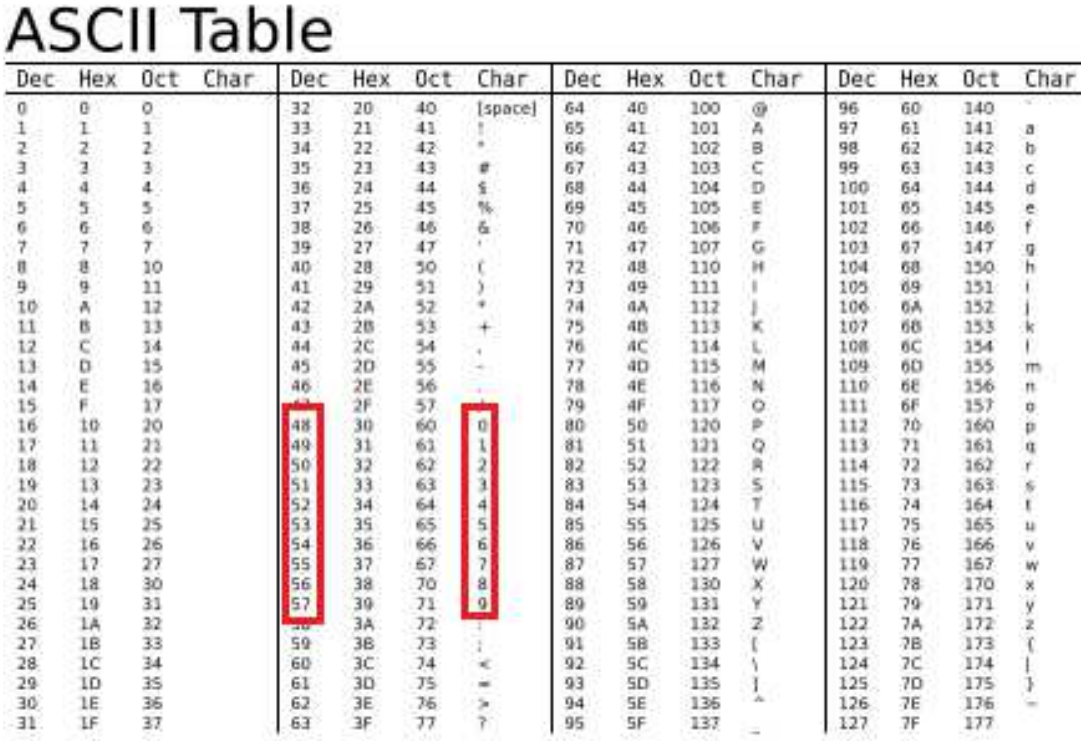
É importante notar que o Arduino está agindo como escravo, ou seja, provendo serviço e não requisitando. O Arduino fica sempre aguardando pela instrução do *software*.

Outro fator que deve ser citado é o da função de capturar o valor da umidade, a função GetStringNumber. Conforme foi mostrado, o sistema envia o caractere A ou B para solicitar modo automático e manual, respectivamente (o modo temporizador envia o modo automático para o Arduino e controla o tempo pelo próprio código, não enviando essa tarefa para o Arduino). No entanto, após enviar A ou B, é enviada a temperatura. A função no Arduino para ler esse dado seria a "bt.read()", no entanto, essa função retornar apenas o primeiro *byte* do serial, não permitindo que um número composto de dois algarismo, por exemplo, 40 fosse lido, foi necessário desenvolver outra forma de leitura. A função leria 4 e repassaria este valor, depois leria 0 e passaria o valor . Para resolver este problema, o seguinte código foi escrito como a seguir:

```
int GetStringNumber()
{
   float value = 0;   // declara variável valor
   while(1)
   {
      char byteBuffer = bt.read();      //lê e armazena os bytes do serial
      if(byteBuffer > -1)         //se o serial for maior que -1 entra na condição
      {
```

```
                if(byteBuffer >= '0' && byteBuffer <= '9')   //se o dado for número entra na condição
                    value = (value * 10) + (byteBuffer - '0');
                else
                    break;
            }
        }
        return value;   //retorna valor
    }
```

A função lê o dado da serial e armazena na variável bytebuffer, depois entra na condição de se o serial não estiver vazio (o serial sempre retorna um valor, -1 se não estiver passando dado) e verifica se ele é um número. Caso seja um número ele faz a expressão de dado – (subtração) ‘0’. Esta expressão funciona da seguinte forma: pela tabela ASCII, mostrada na Figura 3.8, os caracteres de ‘0’ a ‘9’ são representados pelos números 48 a 58. Para converter o caractere ‘1’, a expressão seria 49 (valor de ‘1’ na tabela ASCII) – 48 (valor de ‘0’ na tabela ASCII) retornando 1 como resultado. Isso resolve o problema de retorno do dado digitado no *software* para representar o valor da umidade.



ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		[illegible]	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		[illegible]	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Figura 3. 8 - Tabela ASCII grifada. (Fonte: Autor, adaptada de ivanicabral.blogspot.com)

Com as implementações acima, o sistema está completo em suas partes e pronto para funcionar. No próximo capítulo, serão apresentados os testes e resultados realizados.

CAPÍTULO 4 TESTES E RESULTADOS ALCANÇADOS

A parte de testes e resultados alcançados foi dividida em duas partes. Primeiramente, a parte dos testes unitários, da captação dos dados de temperatura e umidade, teste de comunicação com *bluetooth*, teste do sensor de nível de água e teste do acionamento do relé. Após a finalização dos testes unitários, realizou-se a segunda parte com a homologação do protótipo.

4.1.1 Captação dos sensores

O teste de captação de temperatura e umidade foi realizado de modo simples com o uso do sensor DHT11. A Figura 4.1 ilustra o recebimento destes dados no monitor do Arduino. Para confirmar a veracidade, o sensor foi movido para a saída de um ar condicionado e podemos ver a temperatura diminuir e a umidade aumentar na Figura 4.2.

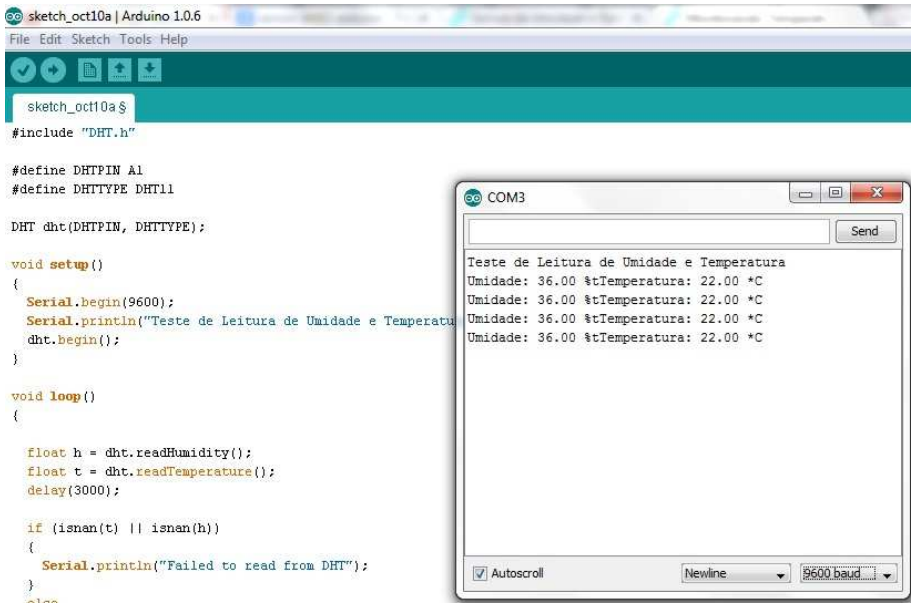


Figura 4. 1 - Teste de captação de umidade e temperatura (fonte: Autor)

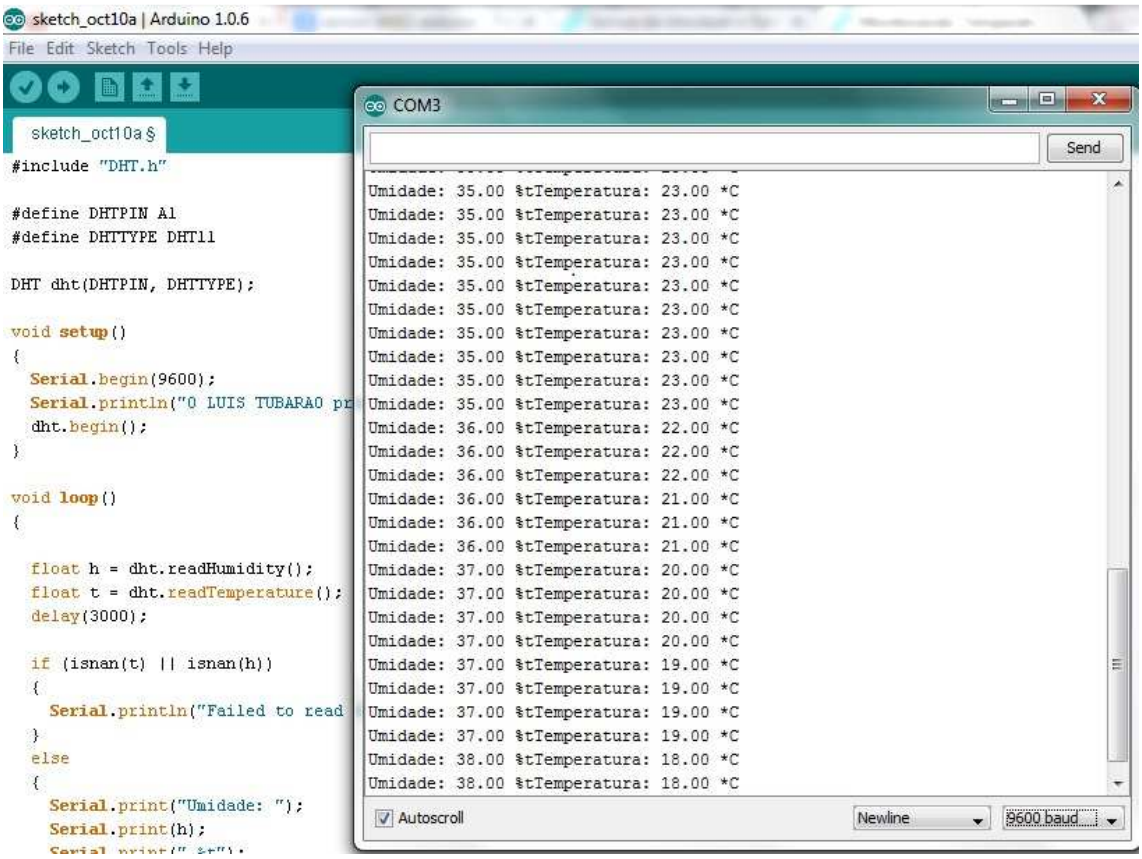


Figura 4. 2 - Variação na umidade e temperatura (Fonte: Autor)

Com estes resultados, foi possível validar o sensor DHT11.

4.1.2 Comunicação com bluetooth

Para o teste da comunicação, foi utilizado o programa Tera Term. O Tera Term é um *software* gratuito que tem a função de emular um programa de terminal. Ele suporta telnet, SSH e conexão com porta serial.

As Figuras 4.3 e 4.4 ilustram o teste de comunicação realizado. Depois de configurado o *bluetooth* no *desktop*, o programa foi executado e a porta serial

escolhida para conectar e realizar os testes. Foi verificado que o alcance do *bluetooth* é de seis metros, confirmando a especificação de seu *datasheet*.

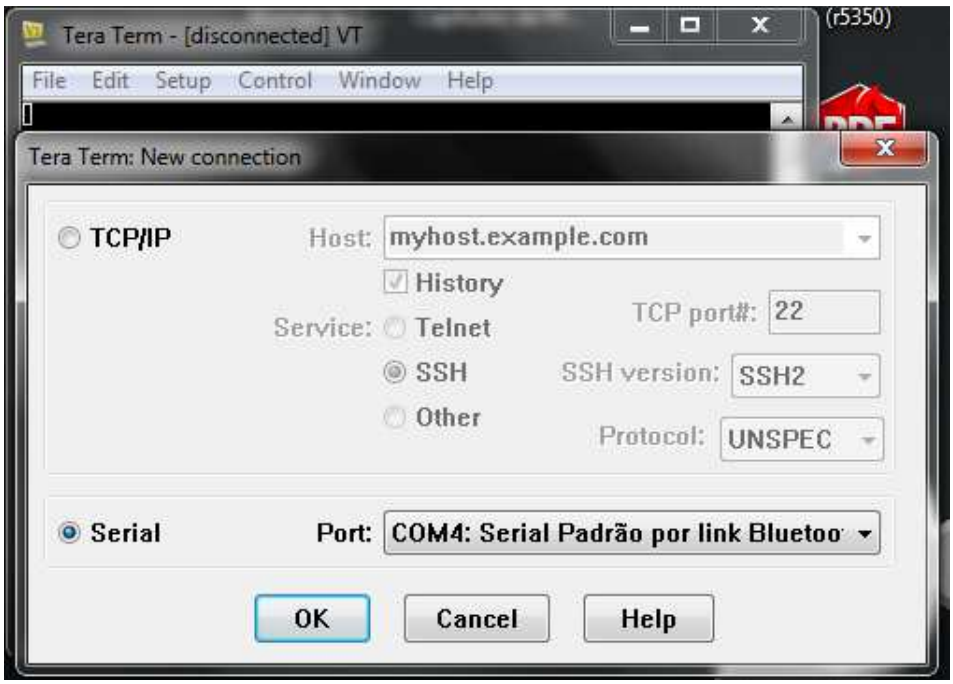


Figura 4. 3 - Iniciando o programa Tera Term para fazer a conexão serial (Fonte: Autor)

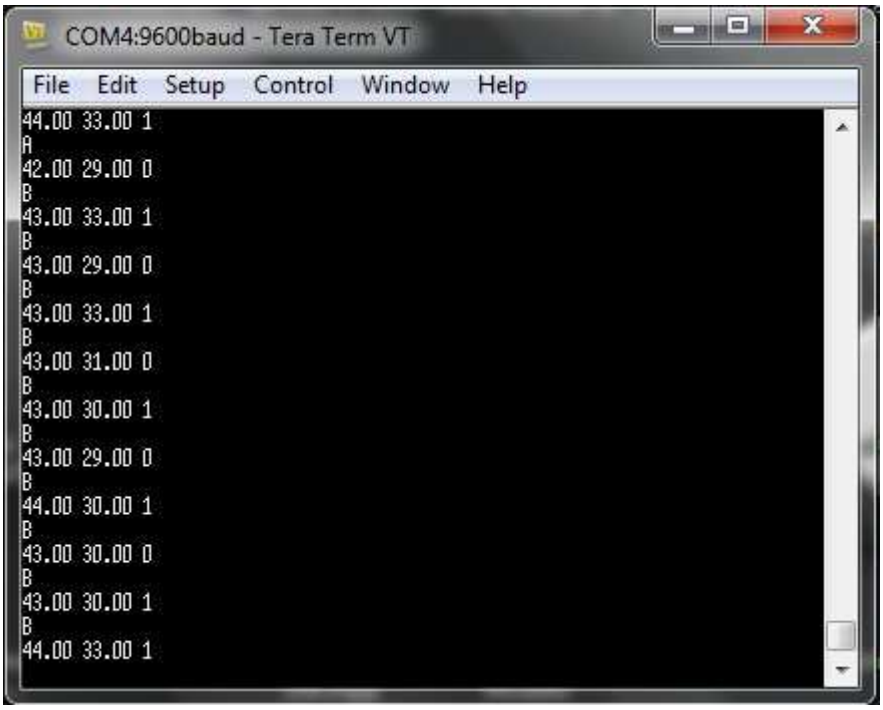


Figura 4. 4 - Terminal com o programa funcionando (Fonta: Autor)

Após obter o resultado positivo da conexão, a comunicação com *bluetooth* foi validada.

4.1.3 Sensor de nível

Para verificar o funcionamento do sensor de nível da água, instalado no repositório do umidificador, foram realizados dois testes utilizando o terminal do programa Tera Term e também a interface.

O primeiro cenário foi testar o nível baixo do repositório. Conforme a Figura 4.5, o repositório foi esvaziado para que o sensor acusasse nível baixo. A Figura 4.5 e 4.6 ilustra a resposta obtida no Tera Term e na interface.

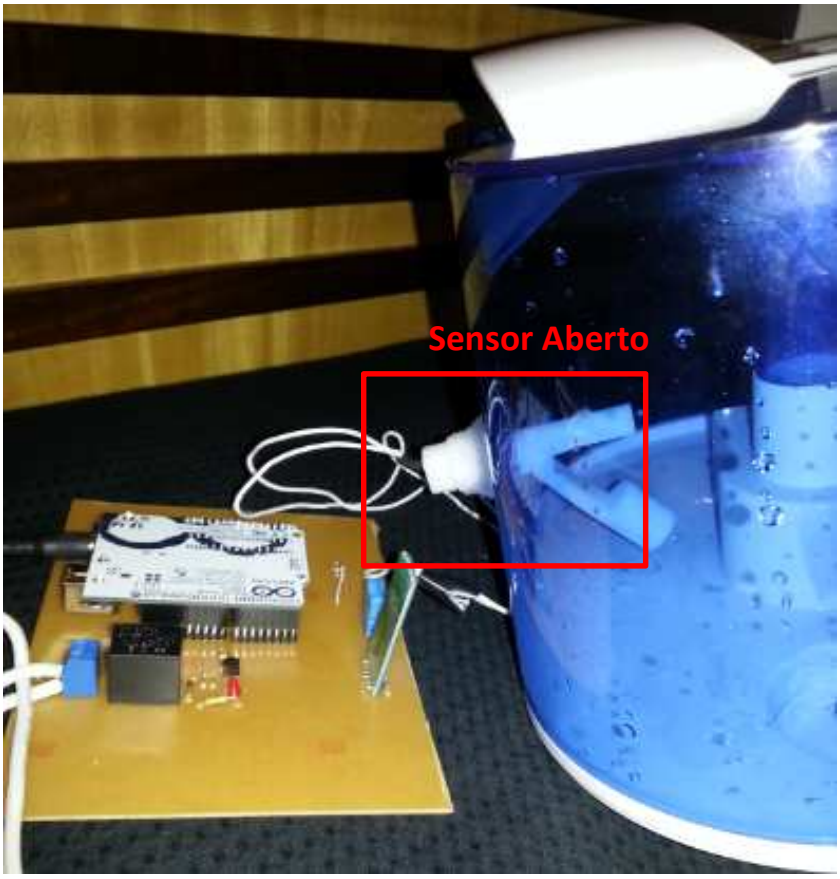


Figura 4. 5 - *Hardware* com sensor aberto. (Fonte: Autor)

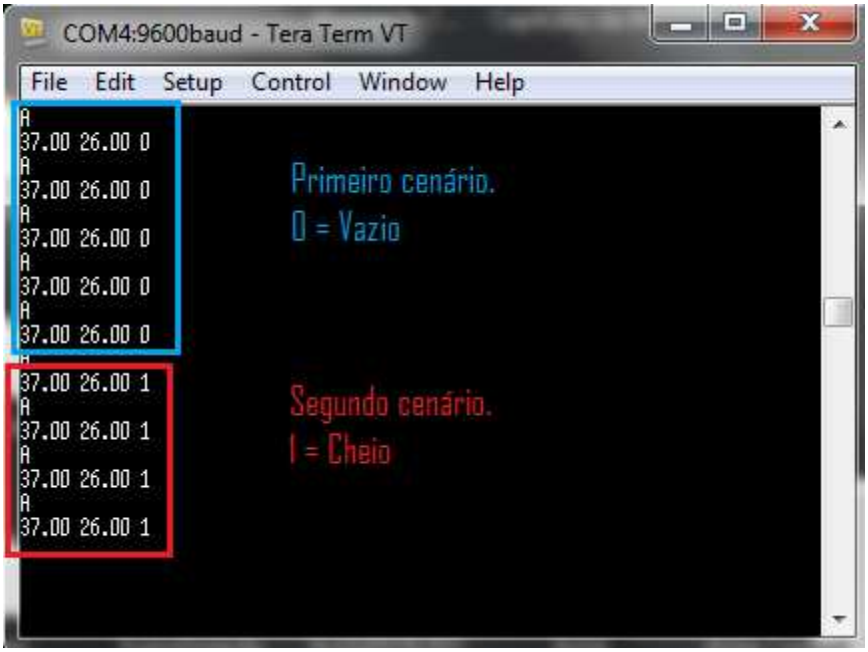


Figura 4. 6 - Respostas dos cenários propostos no Tera Term. (Fonte: Autor)

Figura 4. 7 - Resposta do teste de sensor de nível na interface para estado vazio. (Fonte: Autor)

O segundo cenário foi testar o nível cheio do reservatório. Dessa vez, conforme ilustrado na Figura 4.8, inverte-se o repositório para que o sensor acuse

cheio. Respectivamente, as Figuras 4.6 e 4.9 mostram a resposta no Tera Term e na interface.

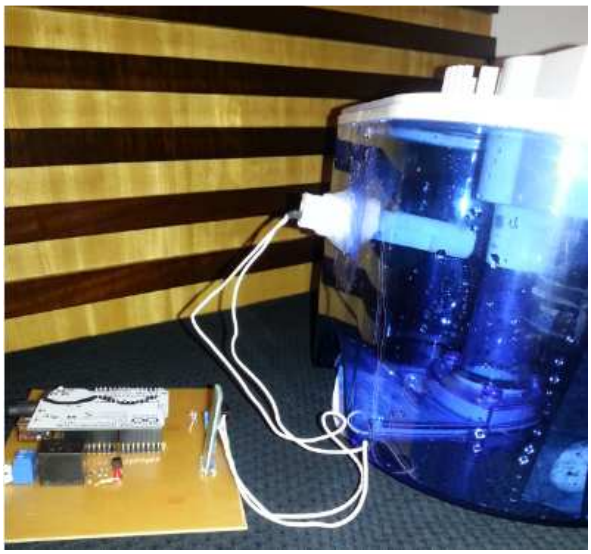


Figura 4. 8 - *Hardware* com sensor fechado. (Fonte: Autor)

Figura 4. 9 - Resposta do teste de sensor de nível na interface para estado cheio. (Fonte: Autor)

Com estes resultados, foi possível vallidar o sensor de nível utilizado no protótipo.

4.1.4 Acionamento do relé

O acionamento do relé para ligar o umidificador deve ser realizado sempre que os modos Automático, Manual ou Temporizador forem escolhidos pela interface. A Figura 4.10 representa o funcionamento do umidificador quando os modos foram acionados.

Figura 4. 10 - Funcionamento do Umidificador. (Fonte: Autor)

Todos os modos foram interpretados corretamente pelo Arduino, ativando e desligando quando solicitado. Após esse resultado, foi possível validar o funcionamento do programa, faltando apenas verificar o desligamento automático.

4.1.5 Desligamento automático

O sistema, como foi descrito nos capítulos anteriores, faz uma verificação na umidade máxima que não causa prejuízo à saúde, que é indicada pela OMS como sendo 80%, no nível de reservatório do umidificador, no temporizador e na

umidade manual repassada pelo usuário. Assim sendo, sempre que alguma das situações acima for atingida, o sistema deve desligar automaticamente.

O primeiro cenário foi uma reconfiguração do código do sistema para que a umidade ideal verificada fosse diminuída para um nível menor do que a umidade do ar do local testado. Ou seja, o sistema não poderia funcionar devido ao atingimento da umidade ‘ideal’ alterada no código.

O segundo cenário foi testar o monitoramento do nível da água. Quando o nível da água estiver baixo, o sistema deixa de funcionar para preservar o umidificador.

O último cenário foi para testar o temporizador. Foi escolhido um tempo pequeno para ser testado o desligamento do sistema quando o tempo se esgotasse.

O sistema desenvolvido obteve sucesso em todos os cenários propostos acima.

O código completo com a programação realizada é apresentado nos Apêndices A, B, C e D.

CAPÍTULO 5 CONCLUSÕES E CONSIDERAÇÕES FINAIS

5.1 Conclusões

O projeto teve como objetivo, criar um sistema remoto para fazer a gestão da umidade do ambiente. O sistema deveria informar a umidade relativa do ar e a temperatura do ambiente. E conforme proposto, agir em três modos diferentes.

Baseado nos resultados dos testes realizados é possível concluir que o objetivo foi alcançado, uma vez que o sistema desenvolvido cumpriu, como esperado, com todos os requisitos propostos.

Conclui-se também que este projeto poderia ser adaptado para um sistema industrial de umidificação, uma vez que o que foi construído é um sistema de controle de acionamento do umidificador. Por último, a principal contribuição deste projeto foi um sistema de monitoramento da umidade do ambiente que foi desenvolvido utilizando softwares gratuitos.

5.2 Sugestões para trabalhos futuros

Embora o projeto esteja funcionando e tenha atingido os objetivos propostos, alguns aprimoramentos podem ser realizados.

O sistema desenvolvido, é limitado pela distância de comunicação do *bluetooth*, considerado local. Uma sugestão de aprimoramento seria disponibilizar o sistema para acionamento e controle remotamente, utilizando-se a internet. Assim, o usuário poderia monitorar e tomar ações a partir dos dados informados sem que a distância seja levada em consideração.

Outra sugestão, como comentado no item 5.1, o sistema pode ser aprimorado para acionar sistemas de umidificação maiores, como os de shoppings e lojas, ou até empresas. Um estudo de viabilidade deve ser realizado devido à necessidade de componentes mais potentes.

Referências

ARDUINO. BuildProcess. **Arduino Build Process**, 16 out. 2014. Disponivel em: <arduino.cc/en/Hacking/Build>. Acesso em: 16 out. 2014.

ARDUINO. Policy. **So you wat to make an Arduino**, 16 out. 2014. Disponivel em: <arduino.cc/en/Main/Policy>. Acesso em: 16 out. 2014.

ASCENCIO, A. F. G. **Fundamentos da programação de computadores**. [S.l.]: Pearson, 1999.

BANZI, M. **Primeiros Passos com o Arduino**. São Paulo: Novatec Editora Ltda., 2011.

BRAIN, K. N. E. M. O interior de um Umidificador. **HSW**, 2010. Disponivel em: <casa.hsw.uol.com.br/umidificadores3.htm>. Acesso em: 20 out. 2014.

ELETRONICS, O. Opiron. **Opiron**, 03 mar. 2013. Disponivel em: <www.opiron.com/portfolio/todos-sobre-los-sensores-dht11-dht22-by-opiron-2>. Acesso em: 18 out. 2014.

ESTADO, A. Estadão. **Estadão**, 24 ago. 2010. Disponivel em: <http://www.estadao.com.br/noticias/geral,umidade-relativa-do-ar-cai-para-7-em-brasilia,599796>. Acesso em: 15 out. 2014.

FARRER, H. et al. **Algoritmos Estruturados**. Rio de Janeiro: Editora Guanabara KOOGAN, 1989.

HUGNEY. Tutorial Módulo Bluetooth. **HU INFINITO**, 2014. Disponivel em: <www.huinfinito.com>. Acesso em: 2014.

ICOS. Sensores de Nível. **Icos**, out. 2014. Disponivel em: <www.icos.com.br/SensorDeNivel/>. Acesso em: 20 out. 2014.

LAGES, Â. D. M. G. E. N. A. D. C. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: EDITORA, 1988.

LEMOS, M. Arduino: Conheça essa plataforma de hardware livre e suas aplicações. **Fazedores**, 17 nov. 2013. Disponivel em: <blog.fazedores.com/arduino-conheca-esta-plataforma-de-hardware-livre-e-suas-aplicacoes>. Acesso em: 16 out. 2014.

MONK, S. **Programação com Arduino**. Porto Alegre: Bookman Editora Ltda., 2013.

MORIMOTO, C. Guia do Hardware. **Guia do Hardware**, 2004. Disponivel em: <http://www.hardware.com.br/termos/transistor>. Acesso em: 31 out. 2014.

PÍCCOLO, H. L. **Estruturas de dados**. [S.l.]: [s.n.], 2000.

SANTOS, D. M. D. Relê - Eletrônica. **Infoescola**, mar. 2012. Disponivel em: <www.infoescola.com/eletronica/rele/>. Acesso em: 20 out. 2014.

SANTOS, F. Terra. **Saúde Terra**, 08 18 2011. Disponivel em: <http://saude.terra.com.br/bem-estar/baixa-umidade-do-ar-exige-cuidados-com-a-saude-veja-dicas,54583f04c2f27310VgnCLD100000bbcceb0aRCRD.html>. Acesso em: 15 out. 2014.

SAVITCH, W. J. **C++ Absoluto**. [S.l.]: Pearson, 2004.

TESKE, D. QT Blog. **Qt Creator**, 6 25 2009. Disponivel em: <blog.qt.digia.com/blog/2009/06/25/qt-creator-12-released>. Acesso em: 27 out. 2014.

WEATHERLY, K. Science. **Science How Stuff Works**, 2008. Disponivel em: <science.howstuffworks.com/dictionary/metereological-terms/question651.htm>. Acesso em: 20 out. 2014.

APÊNDICE A – PROGRAMA DO ARDUINO

Programa utilizado para carregar o microcontrolador e fazer a comunicação com o software.

```
#include <SoftwareSerial.h>
#include "DHT.h"

#define DHTPIN A1
#define DHTTYPE DHT11
#define RxD 10
#define TxD 12
#define IN 5

DHT dht(DHTPIN, DHTTYPE);

int Humidity;
char Mode;
char off = '0';
int GetStringNumber();
int RELE = 8;
```

```
SoftwareSerial bt(RxD,TxD);


void setup(){
  pinMode(RxD, INPUT);
  pinMode(TxD, OUTPUT);
  pinMode(IN, INPUT);
  pinMode(RELE, OUTPUT);
  dht.begin();
  bt.begin(115200);

}

void loop(){

      float h = dht.readHumidity();
      float t = dht.readTemperature();
      delay(50);

      if (isnan(t) || isnan(h))
      {
      bt.println("Failed to read from DHT");
      }
```

```
else{

if ( off == '0'){

char Mode;
if (bt.available()){

Mode = bt.read();
bt.println(Mode);
}

delay(50);

if( Mode == 'A'){
    Humidity = GetStringNumber();
    digitalWrite(RELE, HIGH);
    bt.print(h);
    bt.print(" ");
    bt.print(t);
    bt.print(" ");
    bt.println(digitalRead(IN));
```

```
    if ( h = 80.00 || digitalRead(IN) = 0 )
      off = '1';
}

if( Mode == 'B'){
    Humidity = GetStringNumber();
    digitalWrite(RELE, HIGH);
    bt.print(h);
    bt.print(" ");
    bt.print(t);
    bt.print(" ");
    bt.println(digitalRead(IN));
    if ( h >= Humidity || digitalRead(IN) = 0)
      off = '1';
}

 if( Mode == 'D'){
   Humidity = GetStringNumber();
   off = '1';
 }
}
else
```

```
      {
       digitalWrite(RELE, LOW);
       off = '0';
       }
}
}
int GetStringNumber()
{
   float value = 0;
   while(1)
   {
      char byteBuffer = bt.read();
      if(byteBuffer > -1)
      {
         if(byteBuffer >= '0' && byteBuffer <= '9')
            value = (value * 10) + (byteBuffer - '0');
         else
            break;
      }
   }  return value; }
```

APÊNDICE B – HEADER INTERFACE USUÁRIO

Código do cabeçalho do programa utilizado para a interface.

```
#ifndef PROPATOMAIN_H
#define PROPATOMAIN_H

#include <QMainWindow>
#include <QSerialPort>
#include <QThread>
#include <QTimer>

/**
 * Esta é a declaração da classe propatoMain na interface gráfica.
 * Serve apenas para fazer o link do código com o arquivo ui.
 */
namespace Ui {
class propatoMain;
}

/**
 * Pré-definição das classes propatoMain e propatoThread de forma a poder fazer a referência cruzada desta com a classe Worker.
```

```
 */
class propatoMain;
class propatoThread;


/**
 * Classe Worker, que serve para rodar as funções da thread que executa o loop de leitura e envio para o Arduino.
 * Possui um link para a interface gráfica de modo a poder chamar funções da mesma e a porta serial.
 * O método doWork() executa o loop.
 * A função readSerialData serve para organizar a leitura a partir do Arduino.
 */
class Worker : public QObject
{
    Q_OBJECT

public:
    propatoMain* myMain;
    propatoThread* myThread;
    QSerialPort* myPort;
    Worker();
    ~Worker();
```

```
    int doBreak;


public slots:
    void doWork();
    QString readSerialData();
};


/**
 * Classe propatoThread, que serve para isolar o objeto Worker em uma thread separada da interface gráfica.
 * Já possui um worker interno que é criado e destruído junto com a thread.
 * A função run() é chamada quando a thread é iniciada com o comando start().
 */
class propatoThread : public QThread
{
    Q_OBJECT


public:
    propatoMain* myMain;
    Worker* myWorker;
```

```
public:
    propatoThread();
    ~propatoThread();
    void run();
};


/**
 * Classe propatoMain, que possui a interface gráfica principal do programa.
 * Possui um link para a definição da interface gráfica (ui) e uma propatoThread para ser executada.
 * Também guarda de forma simples os valores de modo de operação e umidade.
 * Possui slots para cada botão clicado na interface (automático, manual, temporizador e desligar).
 */
class propatoMain : public QMainWindow
{
    Q_OBJECT


public:
    explicit propatoMain(QWidget *parent = 0);
    ~propatoMain();
```

```
public:
    Ui::propatoMain* ui;
    propatoThread* backThread;
    QTimer* myTimer;

public:
    char mode;
    double humidity;
    bool timerStatus;

private slots:
    void on_manButton_clicked();
    void on_autoButton_clicked();
    void on_timeButton_clicked();
    void on_offButton_clicked();
}; #endif // PROPATOMAIN_H
```

APÊNDICE C – MAIN INTERFACE USUÁRIO

Código utilizado para fazer o link do programa principal com a interface.

```
#include "propatomain.h"
#include <QApplication>

/**
 * Código que inicia o programa chamando a interface principal.
 */
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    propatoMain w;
    w.show();

    return a.exec();
}
```

APÊNDICE D – PROGRAMA PRINCIPAL INTERFACE USUÁRIO

Código principal utilizado para rodar a thread da interface.

```
#include "propatomain.h"
#include "ui_propatomain.h"
#include <iostream>
#include <QSerialPortInfo>

/**
 * Construtor da classe Worker.
 * Verifica as portas seriais disponíveis e abre a primeira.
 * Depois configura os parâmetros de comunicação.
 * Os warnings são mensagens para o caso de ocorrer algum erro.
 */
Worker::Worker()
{
    doBreak = 0;
    myPort = new QSerialPort;
    myPort->moveToThread(this->thread());
    if (!QSerialPortInfo::availablePorts().isEmpty())
    {
        myPort->setPort(QSerialPortInfo("COM4"));
```

```
if(!myPort->open(QIODevice::ReadWrite))
{
    qWarning("Unable to open serial port!");
}
if(!myPort->setParity(QSerialPort::NoParity) ||
        !myPort->setStopBits(QSerialPort::OneStop) ||
        !myPort->setDataBits(QSerialPort::Data8) ||
        !myPort->setFlowControl(QSerialPort::NoFlowControl ) ||
        !myPort->setBaudRate(QSerialPort::Baud9600) ||
        !myPort->setDataTerminalReady(true))
{
    qWarning("Unable to configure serial port!");
    if (myPort->isOpen())
    {
        myPort->close();
    }
}
if(myPort->error() != QSerialPort::NoError)
{
    qWarning("Unknown error!");
    if (myPort->isOpen())
    {
```

```
                myPort->close();
            }
        }
    }
    else
    {
        qWarning("No device connected!");
    }
}

Worker::~Worker()
{
    myPort->close();
    delete myPort;
}

/**
 * Código do loop principal.
 * Espera alguns segundos, então escreve os parâmetros para o Arduino.
 * Depois captura as mensagens e manda comandos para a interface.
 * Os comandos estão em estruturas de invokeMethod() porque este código não roda na mesma thread que a interface.
 */
```

```
void Worker::doWork()
{
    QThread::msleep(5000);
    QString myString;
    myString.clear();
    char myData[1000];
    if(myPort->waitForReadyRead(5000))
    {
        myPort->read(myData, 1000);
        myString.append(myData);
        qDebug(myString.toStdString().data());
    }
    while(doBreak == 0)
    {
        QThread::msleep(2500);
        qDebug("write 1 (mode):");
        qDebug(QString(myMain->mode).toStdString().data());
        myPort->write(QString(myMain->mode).toStdString().data(), 2);
        myPort->waitForBytesWritten(1000);
        QThread::msleep(700);
        qDebug("read 1 (echo):");
        readSerialData();
```

```
                QThread::msleep(500);
                qDebug("write 2 (humidity):");
                qDebug(QString::number(myMain->humidity).toStdString().data());
                myPort->write(QString::number(myMain-
>humidity).append('\n').toStdString().data(), 5);
                myPort->waitForBytesWritten(1000);
                QThread::msleep(700);
                qDebug("read 2 (h t l)");
                myString = readSerialData();
                QStringList humTemp = myString.split(" ");
                if (!humTemp.isEmpty() && humTemp.length() == 3)
                {
                    QMetaObject::invokeMethod(
                        myMain->ui->humLcd,
                        "display",
                        Qt::QueuedConnection,
                        Q_ARG(double, humTemp.at(0).toDouble())
                    );
                    QMetaObject::invokeMethod(
                        myMain->ui->tempLcd,
                        "display",
                        Qt::QueuedConnection,
                        Q_ARG(double, humTemp.at(1).toDouble())
```

```
);


if (humTemp.at(2).toInt() == 0)
{
    QMetaObject::invokeMethod(
        myMain->ui->levelBar,
        "setValue",
        Qt::QueuedConnection,
        Q_ARG(int, 20)
    );
}
else
{
    QMetaObject::invokeMethod(
        myMain->ui->levelBar,
        "setValue",
        Qt::QueuedConnection,
        Q_ARG(int, 80)
    );
}
myPort->write("\n");
readSerialData();
```

```
            myPort->clear();
        }
    }
}


/**
 * Código para ler os dados do Arduino.
 * Mostra no debug o comportamento da leitura.
 * O código espera por até 5000 milissegundos para executar cada leitura.
 */
QString Worker::readSerialData()
{
    QString myString;
    myString.clear();
    char myData[1000];
    if(myPort->waitForReadyRead(5000))
    {
        myPort->readLine(myData, 1000);
        myString.append(myData);
        qDebug(myString.toStdString().data());
    }
    else
```

```
    {
        myString.clear();
        qWarning("Read timeout");;
    }
    return myString;
}

/**
 * Construtor padrão da propatoThread com controle do Worker interno.
 */
propatoThread::propatoThread()
{

}

/**
 * Destrutor padrão da propatoThread com controle do Worker interno.
 */
propatoThread::~propatoThread()
{
    myWorker->doBreak = 1;
    if (myWorker != NULL)
```

```
    {
        delete myWorker;
    }
}

/**
 * Código a ser executado quando a propatoThread inicia.
 * É apenas um chamado para a função do Worker.
 * Esta estrutura garante que o Worker roda em uma thread separada.
 */
void propatoThread::run()
{
    myWorker = new Worker();
    myWorker->myMain = this->myMain;
    myWorker->doWork();
}

/**
 * Construtor da janela principal.
 * Seta os parâmtros iniciais de umidade e modo.
 * Também inicia a thread de controle do Arduino.
 */
```

```
propatoMain::propatoMain(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::propatoMain)
{

    this->humidity = 80;
    this->mode = 'A';
    this->timerStatus = false;
    this->myTimer = new QTimer();

    ui->setupUi(this);

    backThread = new propatoThread();
    backThread->myMain = this;
    backThread->start();
}

/**
 * Destrutor da janela principal.
 * Encerra a thread de controle do Arduino.
 * Não está funcionando e não sei por que.
 */
```

```
propatoMain::~propatoMain()
{
    delete ui;
    delete myTimer;
    backThread->exit();
    backThread->wait();
}

/**
 * Comandos da seleção do botão de modo automático;
 */
void propatoMain::on_autoButton_clicked()
{
    this->timerStatus = false;
    this->myTimer->stop();
    this->mode = 'A';
    this->humidity = 80;
}

/**
 * Comandos da seleção do botão de modo manual;
 */
```

```
void propatoMain::on_manButton_clicked()
{
    this->timerStatus = false;
    this->myTimer->stop();
    this->mode = 'B';
    this->humidity = this->ui->humBox->value();
}

/**
 * Comandos da seleção do botão de modo temporizador;
 */
void propatoMain::on_timeButton_clicked()
{
    if (this->timerStatus = false)
    {
        connect(this->myTimer, SIGNAL(timeout()), this, SLOT(on_offButton_clicked()));
        this->myTimer->setSingleShot(true);
        this->myTimer->start(abs(this->ui->timeEdit->time().msecsTo(QTime())));
    }
    this->timerStatus = true;
    this->mode = 'A';
```

```
    this->humidity = 80;
}


/**
 * Comando do botão de desligar;
 */
void propatoMain::on_offButton_clicked()
{
    Worker myWorker;
    this->timerStatus = false;
    this->myTimer->stop();
    this->mode = 'D';
    this->humidity = 0;
}
```