CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB
FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS - FATECS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

ÉDER PEIXOTO MARQUES

A LÓGICA FUZZY APLICADA AO CONTROLE DE TEMPERATURA E UMIDADE

Orientador: Prof. MSc. Luciano Henrique Duque

Brasília
2016

ÉDER PEIXOTO MARQUES

A LÓGICA FUZZY APLICADA AO CONTROLE DE TEMPERATURA E UMIDADE

Trabalho de conclusão de Curso apresentado à Banca examinadora do curso de Engenharia da Computação da FATECS – Faculdade de Tecnologia e Ciências Sociais Aplicadas – Centro Universitário de Brasília como requisito para obtenção do título de Engenheiro da Computação.

Orientador: Prof. MSc. Luciano Henrique Duque

Brasília
2016

ÉDER PEIXOTO MARQUES

A LÓGICA FUZZY APLICADA AO CONTROLE DE TEMPERATURA E UMIDADE

Trabalho de Conclusão de Curso apresentado à Banca examinadora do curso de Engenharia da Computação da FATECS – Faculdade de Tecnologia e Ciências Sociais Aplicadas – Centro Universitário de Brasília como requisito para obtenção do título de Engenheiro da Computação.

Orientador: Porf. MSc. Luciano Henrique Duque

BANCA EXAMINADORA

_____

Prof. Dr. Abiézer Amarília Fernandes
Coordenador do Curso

_____

Prof. MSc. Luciano Henrique Duque
Orientador

_____

Prof. Dr. Camilo Sanchez Ferreira
UniCeub

_____

Profª. Dra. Claudia Patricia Ochoa Diaz
UniCeub

Brasília
2016

AGRADECIMENTOS

Todo trabalho é fruto de um conhecimento coletivo repassado de geração em geração e que permite aos mais novos trilharem novos desafios tomando por base os obstáculos já ultrapassados pelos mais velhos. E não poderia ser diferente com este trabalho, o qual com certeza é fruto de infinitas colaborações.

Agradeço à minha esposa Aline e minhas filhas Emily e Lívia (esta inclusive que nasceu durante a elaboração deste trabalho) por toda a compreensão durante os longos meses em que me dediquei a este projeto.

Agradeço à minha mãe por sempre ter sido a minha principal incentivadora quanto aos estudos, e ao meu pai por ter criado as condições para que eu pudesse frequentar uma faculdade.

Agradeço aos meus irmãos e familiares, para os quais tenho tanto carinho.

Agradeço aos amigos e colegas, em especial ao Edilson, Jessica, Josihel e Renan, dos quais sempre obtive tanta ajuda quando necessitei.

E por fim agradeço ao professor Luciano pela orientação dedicada durante toda a elaboração deste trabalho, ao professor Abiézer pelas informações passadas que sempre foram muito úteis, e a todos os demais professores que de alguma forma contribuíram para que este trabalho fosse realizado.

Obrigado a todos!

Éder Peixoto Marques

CITAÇÃO

"Não se iludam. A ciência não está alicerçada na rocha. A vasta estrutura de suas teorias ergue-se sobre um pântano. É como um edifício sustentado por estacas que mergulham num terreno movediço mas não atingem nenhuma base natural. Muitas questões fundamentais para a ciência continuam em aberto. Se não queremos nos ver reduzidos a meras fatias do conhecimento oficial, precisamos manter abertas também as nossas mentes."

Karl Popper

RESUMO

A necessidade de redução no consumo de energia elétrica e emissão de gases de efeito estufa tem sido uma preocupação mundial, com a criação de normas e leis que exigem melhores desempenhos de equipamentos como automóveis, eletrodomésticos, dentre outros. Dessa forma, os equipamentos passam a ser projetados para que realizem o maior trabalho com o menor consumo energético possível, e, portanto, precisam realizar controles eficientes sobre os recursos disponíveis. Porém, modelar sistemas pode requerer controles muito complicados, devido à dificuldade em se conhecer e entender as relações entre as diversas variáveis envolvidas. Com isso, no intuito de otimizar o uso de recursos e ao mesmo tempo simplificar a modelagem, este projeto propõe o desenvolvimento de um sistema que permita controlar a temperatura e umidade do ambiente onde se localize através da utilização de lógica *fuzzy*. A criação de um sistema de controle visa atingir a primeira premissa descrita, ou seja, um sistema que otimize os recursos. E quanto ao uso de lógica *fuzzy*, essa é utilizada devido ao fato de dispensar o conhecimento de boa parte da modelagem matemática e física da planta a ser controlada, requerendo para tanto a criação de regras por especialistas da área. Portanto, foi desenvolvido de forma satisfatória um sistema que quanto à temperatura é capaz apenas de aumentá-la, porém em relação à umidade é capaz de aumentá-la e reduzi-la.

Palavras-chave: lógica fuzzy; sistema de controle; Arduino®; MATLAB®.

ABSTRACT

The need to reduce the consumption of electric energy and the emission of greenhouse gases has been a worldwide concern, with the creation of norms and laws that require better performance of equipment such as automobiles, home appliances, among others. In this way, the equipment is designed to perform the most work with the lowest energy consumption possible, and therefore it must perform efficient controls over the available resources. However, modeling systems may require very complicated controls, due to the difficulty in knowing and understanding the relation between the variables involved. Thus, in order to optimize the use of resources and, at the same time, simplify the modeling, this project proposes the development of a system that allows controlling the temperature and humidity of the environment where it is located through the use of fuzzy logic. The creation of a control system aims at reaching the first premise described, that is, a system that optimizes the resources. As for the use of fuzzy logic, this is used due to the fact that it dispenses the knowledge of much of the mathematical and physical modeling of the plant to be controlled, requiring the creation of rules by specialists in the area. Therefore, it was developed satisfatorily a system that in terms of temperature is only able to increased it, but in relation to humidity is capable of increase it and decrease it.

Keywords: fuzzy logic; control system; Arduino®; MATLAB®.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

AVR	*Advanced Virtual RISC ou Alf and Vergad RISC*
DAC	*Digital to Analog Converter*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
FIS	*Fuzzy inference system*
GND	*Ground*
IDE	*Integrated Development Environment*
NC	*Normaly Closed*
NO	*Normaly Opened*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
USB	*Universal Serial Bus*
V	Volt
VCC	*Collector supply voltage*

SUMÁRIO

1 INTRODUÇÃO

A partir da máquina a vapor de James Watt, em 1769, houve um acentuado progresso em termos de automação de processos (ROMANO, 2002), intensificando-se ainda mais durante todo o século XX, com os avanços da ciência, em especial da eletrônica.

Em todos esses avanços, os métodos e tecnologias utilizados ainda hoje são baseados profundamente na lógica binária, ou lógica clássica, sendo o filósofo e matemático Aristóteles considerado seu fundador. Na lógica clássica uma declaração é verdadeira ou falsa, não podendo ao mesmo tempo apresentar ambas as características, mesmo parcialmente (CAVALCANTI, 2012).

Porém há uma grande diferença entre a capacidade criativa dos seres humanos comparada à capacidade de solução das máquinas computacionais. Enquanto estas realizam todas suas operações seguindo rigidamente a lógica clássica, aqueles possuem um raciocínio que geralmente é incerto, impreciso e muitas vezes difuso ou nebuloso (SIMÕES, 2007).

Devido à necessidade de haver uma lógica que incorporasse as peculiaridades do raciocínio humano, o matemático Lotfi Zadeh propôs no ano de 1965 a Teoria dos Conjuntos *Fuzzy*, criada para resolver os problemas incompatíveis com a lógica clássica (CAVALCANTI, 2012).

A lógica *fuzzy* tem como principal intenção propiciar um tratamento matemático a certos termos linguísticos subjetivos, como por exemplo quando se quer definir que algo é ou está QUENTE, MORNO ou FRIO. Para tanto, Zadeh baseou-se no princípio de que qualquer conjunto pode ser caracterizado por uma função matemática. Dessa forma, para qualquer termo linguístico que pertença ao conjunto *fuzzy* será necessário associar uma função denominada de função de pertinência (BARROS, 2006).

Essa função de pertinência descreve para cada termo linguístico o grau de pertinência ou não do termo, de forma que a passagem de um extremo a outro seja gradativa e não-abrupta (BILOBROVEC, 2005). Com isso, a lógica *fuzzy* foca no tratamento e descrição dos termos utilizados no sistema, e não especificamente no modelo físico-matemático que o descreve.

Segundo SIMÕES (2007), projetistas costumam encontrar dificuldades ao projetarem sistemas através do emprego de modelagem matemática precisa de uma planta, devido a diversos fatores, dentre eles: fenômenos físicos pouco compreendidos, valores imprecisos dos parâmetros do modelo, problemas com as dimensões reais do sistema em comparação com as

simplificações realizadas e maquetes construídas, e quanto a distúrbios externos que possam influenciar.

Todos esses problemas descritos passam a ser ainda mais acentuados quando o sistema a ser controlado possui comportamento não linear, situação na qual os controladores PID começam a deixar de ser indicados, pois exigem a linearização da planta, podendo mesmo assim apresentarem comportamento não estável.

Portanto, quando sistemas a serem controlados se comportam de forma não linear, quando a descrição matemática dos processos químico-físicos não são bem conhecidas, quando da existência de diversas variáveis de entrada e/ou saída, quando os parâmetros possam apresentar comportamento variante no tempo, ou ainda devido ao aumento da complexidade da planta vir a exigir a adoção de sistemas que empreguem o uso de inteligência artificial, o uso de lógica *fuzzy* passa a ser levada em consideração, pois permite-se "expressar de uma maneira sistemática quantidades imprecisas, vagas e mal definidas" (SIMÕES, 2007, p. 8).

Uma diferença essencial das metodologias convencionais de controle quando comparadas à lógica *fuzzy* é quanto ao que está sendo modelado. Enquanto o objetivo principal das metodologias convencionais é modelar a planta a ser controlada, em sistemas *fuzzy* os comportamentos de operadores humanos passam a ser focalizados. Com isso, muda-se o enfoque dos problemas de controle, pois o sistema passa a buscar o entendimento da planta ao invés de conhecê-la previamente.

No entanto não dever-se-ia então serem substituídos todos os sistemas de controle convencionais por sistemas de controle *fuzzy*? A resposta mais sensata é não, pois existem situações em que controladores convencionais possuirão um custo-benefício muito melhor ao serem comparados com controles inteligentes utilizando *fuzzy*. Portanto, diversas análises devem ser tomadas antes de se decidir qual o método de controle a ser utilizado (SIMÕES, 2007).

Para o presente projeto, o emprego da lógica *fuzzy* se justifica pelo fato do sistema a ser controlado ser não linear, pelas variáveis possuírem interferência entre si, ou seja, a tentativa de controle da temperatura interfere na variação da umidade, e vice-versa. Além disso, existem interferências do meio externo nos parâmetros controlados, dificultando dessa forma o controle da planta.

Com o cenário descrito, desenvolveu-se um sistema de controle de temperatura e umidade que ao mesmo tempo em que seja justificado do ponto de vista didático e acadêmico, também o seja tecnicamente aceitável, pois o mesmo atinge um regime estacionário na maioria dos cenários em que os testes foram realizados.

Para tanto, o sistema de controle desenvolvido no contexto deste trabalho objetiva controlar temperatura e umidade do ambiente conforme os valores de *set point* definidos pelo operador, pois há diversas situações em que o controle dessas variáveis são imprescindíveis ou desejados, como em silos, estufas e aviários, agregando dessa forma ganhos quanto a bem-estar, aumento de produtividade e economia de energia.

Dessa forma, uma central de processamento realizará a leitura de sensores. Esses valores serão então passados para um motor de regras *fuzzy*, o qual definirá quais deverão ser os valores de potência a serem aplicados em cada um dos atuadores de forma que se atinjam os valores desejados.

1.1 Objetivos do Trabalho

1.1.1 Objetivo geral

O objetivo é desenvolver um sistema que controle a temperatura e umidade fazendo uso da lógica *fuzzy*.

1.1.2 Objetivos específicos

- Desenvolver um circuito eletrônico que permita a comunicação com a central de processamento, permitindo dessa forma a leitura dos sensores e envio de sinais aos atuadores;
- Construir um circuito para acionamento do subsistema de aquecimento;
- Construir um circuito para acionamento do subsistema de ventilação;
- Construir um circuito para acionamento do subsistema de umidificação de ar;
- Desenvolver um conjunto de regras *fuzzy* para utilização do sistema de inferência *fuzzy* (FIS) do MATLAB®;
- Modelar sistema de controle no MATLAB para realização da leitura dos sensores e acionamento dos atuadores;
- Construir uma maquete que permita integrar o sistema de controle MATLAB com os circuitos dos sensores e atuadores.

1.2 Metodologia

Para se alcançar os objetivos específicos do projeto, foram implementadas as etapas abaixo descritas. Para melhor compreensão, segue na figura a seguir diagrama esquemático sobre cada etapa.

Figura 1.2-1 - *Diagrama do fluxo de desenvolvimento do projeto.*

Fonte: Elaborado pelo autor.

Primeira etapa: Compreende a pesquisa bibliográfica e estudo sobre lógica *fuzzy*, entendendo sobre o seu funcionamento e como a mesma pode ser aplicada ao sistema de controle e como deverá ser implementada utilizando o MATLAB. Necessário também entender sobre o funcionamento e operação da *toolbox Fuzzy* do MATLAB, pois será utilizada na implementação das regras.

Segunda etapa: Pesquisar e testar os modelos disponíveis de sensores de temperatura e umidade, levando-se em consideração as faixas de valores de atuação do sistema, e de preferência componentes integrados que contenham ambos os sensores. Verificar-se-á também a necessidade de serem utilizados mais de um componente para a obtenção dos valores, de forma a evitar a obtenção de valores que não representem a média da temperatura e umidade no interior do ambiente controlado.

Terceira etapa: Modelar o sistema de inferência *fuzzy* na *toolbox Fuzzy* do MATLAB, levando-se em consideração que serão lidos os valores de temperatura e umidade (*inputs*) e serão acionados atuadores para aquecimento, ventilação e umidificação do ar (*outputs*). Nessa etapa realizam-se os primeiros testes unitários no intuito de verificar se a resposta aos sinais de entrada (*inputs*) é a esperada e se o conjunto de regras implementado é o mínimo suficiente, pois caso contrário o sistema poderá se tornar instável.

Quarta etapa: Desenvolver a interface entre *software* (MATLAB) e *hardware* (sensores, atuadores e *displays*) através da plataforma de prototipagem Arduino®, o qual será utilizado para a realização da comunicação com a Central de Processamento. As principais funcionalidades da interface *software-hardware* é realizar a leitura dos sensores, repassar os dados obtidos à Central de Processamento, receber dessa os valores de potência a serem aplicados nos atuadores e repassá-los aos circuitos responsáveis. A Central de Processamento enviará à interface *software-hardware* valores compreendidos entre 0 (zero) e 1 (um) referentes à potência a ser aplicada a cada atuador. Esses valores serão então convertidos através de um DAC (conversor digital analógico) para valores compreendidos entre 0 (zero) e 5 (cinco) volts e repassados então aos circuitos dos atuadores, os quais converterão para um nível de potência adequado.

Quinta etapa: Pesquisar e testar os modelos disponíveis de atuadores necessários ao sistema de controle, que são: resistência elétrica, umidificador e ventilador. Esses atuadores serão acionados através de circuitos eletrônicos a serem construídos, e serão os responsáveis por alterar os valores das variáveis controladas.

Sexta etapa: Desenvolver os circuitos que conterão os atuadores a serem acionados pela Central de Processamento. Cada circuito será específico para o tipo de funcionamento exigido pelo atuador, recebendo como *input* um sinal compreendido entre zero e cinco volts, sendo zero o menor valor de potência possível, e cinco a máxima potência a ser aplicada pelo atuador. Dessa forma, cada circuito sempre receberá valores compreendidos entre zero e cinco volts, ficando a cargo desse realizar as conversões necessárias para o pleno e correto funcionamento do atuador.

Sétima etapa: Modelar no MATLAB a Central de Processamento. Essa ficará responsável por integrar todas as partes do sistema de controle. Primeiramente a Central realiza a leitura dos sensores através de uma interface *software-hardware* utilizando Arduino. Esses valores são então passados como *input* para o sistema de inferência *fuzzy*. O sistema *fuzzy* realiza então os cálculos e retorna os valores de potência a serem aplicados aos atuadores como *outputs*. Os valores de potência são então passados pela Central aos atuadores através de interface *software-hardware* utilizando Arduino.

Oitava etapa: Desenvolver o circuito de interface com o usuário que estará fixado na maquete para apresentação dos valores de temperatura e umidade, dos respectivos valores de *set point*, botões que permitam ao operador alterar os valores de *set point* e de um botão que permita parar o sistema de controle.

Nona etapa: Realização de testes que permitam verificar o funcionamento de todo o sistema, de modo a validar o conjunto de regras *fuzzy* desenvolvidas, através da geração de gráficos pelo MATLAB. Nessa etapa haverá o auxílio de um climatizador de ar, o qual será utilizado para variar a temperatura e umidade externas ao ambiente. Serão então definidos os valores de *set point* a serem atingidos, e com a geração de gráficos espera-se que a curva obtida seja de primeira ou segunda ordem, com um tempo de estabelecimento (t_s) de poucos segundos, de modo geral não excedendo 5 minutos. Para a calibração e verificação da taxa de erro dos sensores, serão realizadas comparações entre os valores lidos pelos sensores com aqueles obtidos com o uso de instrumentos de termohigrometria comerciais. Dessa forma, será possível aferir o grau de precisão do sistema sensorial.

1.3 Motivação

Sistemas automatizados para controle de equipamentos e ambientes se tornam cada vez mais comuns em indústrias, e aos poucos também nas residências, alçados com os progressos da ciência vistos principalmente nas últimas décadas.

Dessa forma, exige-se de estudantes e profissionais de engenharia conhecimentos dos diversos métodos existentes na área de controle, de sistemas On-Off a PID, permitindo assim que estejam capacitados a determinar o melhor sistema a ser empregado em cada situação.

Portanto, a principal motivação do presente trabalho é a de permitir desenvolver o conhecimento em sistemas de controle através do emprego da lógica *fuzzy*, além de possibilitar

a análise das mais diversas etapas do desenvolvimento de uma solução de engenharia para um determinado problema, onde se exige um casamento entre teoria e prática.

Quanto à motivação de se controlar temperatura e umidade, deve-se ao fato de haverem diversas situações em que essas variáveis se tornam imprescindíveis ou quase que obrigatórias, como por exemplo no armazenamento de grãos em silos, e no bem-estar animal em aviários e estufas.

1.4 Resultados Esperados

Com a implementação do projeto espera-se que seja possível controlar a temperatura e umidade de um ambiente conforme os valores desejados, desde que esses não sejam muitos maiores ou menores que os valores de temperatura e umidade do ambiente externo, de tal modo que a estabilização ocorra em um curto espaço de tempo sem grandes oscilações de seus valores.

Para demonstração, haverá a construção de maquete que representará o ambiente a ser controlado. Essa conterá em seu interior uma fonte de alimentação, circuitos dos sensores e dos atuadores, além dos equipamentos/componentes de cada subsistema (sensores, aquecimento, ventilação e umidificação). Na parte frontal da maquete haverá *displays* para apresentação em dos valores atuais de temperatura e umidade do seu interior, e de botões para controle dos valores de *set point*.

Os circuitos contidos na maquete serão interligados a um computador através do uso da plataforma Arduino. Nesse computador haverá a execução do MATLAB, o qual conterá toda a lógica de controle *fuzzy* e uma interface com o usuário que permita visualizar gráficos contendo os históricos dos dados obtidos e calculados. Dessa forma, o Arduino será a interface entre o *hardware* (circuitos, sensores e atuadores) e o *software* (MATLAB) do sistema de controle.

1.5 Trabalhos Correlatos

No artigo apresentado no XVIII Congresso Brasileiro de Automática, CAVALCANTE et al (2010) propõe um algoritmo de controle preditivo para incubadoras neonatais. Apesar do artigo objetivar controlar equipamento semelhante ao utilizado neste trabalho, o método utilizado é diferente, o qual exige um maior conhecimento do modelo matemático e físico a ser controlado para a obtenção de melhores resultados.

Em artigo de autoria de RASHIDI et al (2003), publicado em conferência realizada pelo IEEE, propõem-se um sistema de controle para sistemas não lineares utilizando-se uma junção entre controlador PID e *fuzzy*. Segundo os autores, o desempenho obtido foi superior a sistemas

baseados apenas em controlador PID. Apesar do uso da lógica *fuzzy*, esse artigo difere do presente trabalho por empregar mais de uma técnica de controle.

Na dissertação para obtenção do título de mestrado, BILOBROVEC (2005) aplica a lógica *fuzzy* em um sistema de controle, gerenciamento e manutenção de silos, em especial no processo de aeração de grãos. A motivação pelo emprego de *fuzzy* na dissertação é semelhante ao uso neste trabalho, pois permite a captura do conhecimento de especialistas, sem a necessidade de um conhecimento profundo da modelagem matemática da planta a ser controlada.

1.6 Estrutura do Trabalho

O presente trabalho está estruturado em capítulos e seções, assim divididos:

No primeiro capítulo, há a introdução ao trabalho proposto, com a apresentação dos objetivos, da metodologia a ser empregada, a motivação e resultados esperados, além da apresentação de alguns trabalhos correlatos.

O segundo capítulo contém o referencial teórico, apresentando todo o embasamento utilizado na confecção do sistema de controle *fuzzy*.

O terceiro capítulo mostrará o desenvolvimento da solução proposta, com a apresentação das técnicas, equipamentos, componentes e ferramentas utilizadas durante todo o trabalho.

O quarto capítulo contém os resultados obtidos nos testes realizados, assim como a descrição dos problemas encontrados durante a realização dos mesmos.

O quinto capítulo apresenta as conclusões do projeto e sugestões para futuros trabalhos e pesquisas.

2 REFERENCIAL TEÓRICO

Este capítulo visa explanar sobre os principais assuntos tratados na proposta de solução do projeto, sendo dividido em três seções. Na primeira, haverá a apresentação sobre lógica *fuzzy*. A segunda destina-se à explicação sobre a ferramenta utilizada para modelagem da lógica de controle, o MATLAB®, em especial as *toolboxes Simulink* e *Fuzzy*. Na última seção explicar-se-á o funcionamento do Arduino®, plataforma de prototipagem utilizada para realizar a integração entre o MATLAB e os circuitos do *hardware* de controle.

2.1 Lógica Fuzzy

A lógica *fuzzy* foi apresentada ao meio científico no ano de 1965, por Lotfi Asker Zadeh, no artigo intitulado *Fuzzy Sets* (RIGNEL, 2011), tomando por base a lógica clássica de Aristóteles e os princípios desenvolvidos por Jan Lukasiewicz.

Como se trata de um vasto assunto, para melhor compreensão essa seção será dividida em: lógica clássica e de Lukasiewicz, termos *fuzzy*, conceitos da teoria de conjuntos *fuzzy*, operações básicas da lógica *fuzzy,* probabilidade versus possibilidade, funções de pertinência, fuzzificação, base de regras, inferência e defuzzificação.

2.1.1 Lógica clássica e lógica de Lukasiewicz

Segundo a lógica clássica, uma proposição poderá assumir somente o valor verdadeiro ou falso, e, necessariamente, terá que assumir um desses dois valores, não sendo possível outras possibilidades. Para tanto, utilizam-se predicados através de termos exatos, como: maior que, par, primo, etc (BILOBROVEC, 2005). Dessa forma, pode-se definir sempre dois conjuntos (também denominados como **crisp**): o conjunto dos elementos que pertencem ou satisfazem a proposição, e o conjunto daqueles que não pertencem ou não satisfazem.

À equação que define quais elementos pertencem a um determinado conjunto e quais não pertencem, denomina-se função característica ou função de pertinência (FP), conforme a definição abaixo:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \qquad (2.1\text{-}1)$$

Pela definição da **Equação (2.1-1)**, a um determinado elemento "x" será associado o valor 1 (um) caso pertença ao conjunto A, senão será associado o valor 0 (zero). Dessa forma, verifica-se que a lógica clássica permite dois estados.

É importante ressaltar que os valores 1 (um) e 0 (zero) da função característica acima não possuem significado numérico direto, pois são apenas símbolos convenientes que permitem distinguir os elementos de um conjunto universo que pertencem ou não a um conjunto A (NICOLETTI, 2013).

Segundo SIMÕES (2007), o principal fundamento da lógica clássica, ou booleana, é que a função de pertinência seja bivalente, ou seja, assuma sempre dois estados possíveis.

Porém, algumas variáveis utilizadas em nosso dia a dia, transmitidas e perfeitamente compreendidas linguisticamente entre interlocutores, têm invariavelmente permanecido fora do tratamento da lógica clássica, pois necessitam distinguir qualificações por meio de graduações com mais de dois estados (BARROS, 2006).

Com essa necessidade de serem tratados casos que exigem mais de dois estados, durante a primeira metade do século XX surgem diversos estudiosos que estendem a lógica de dois valores para lógicas multivaloradas. Dentre estas, cabe destacar a lógica de Lukasiewicz, na qual são definidos três estados. Conforme REZNIK (1997), Jan Lukasiewicz propôs um modelo matemático para o tratamento de imprecisões, onde define-se o valor 1 (um) para o caso verdadeiro, 0 (zero) para falso e ½ para algo que é ao mesmo tempo verdadeiro e falso.

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ \frac{1}{2}, & x \in parcialemente\ a\ A \\ 0, & x \notin A \end{cases} \qquad (2.1\text{-}2)$$

2.1.2 Termos *fuzzy*

Segundo KLIR (1997), utilizam-se no cotidiano diversos termos vagos, porém sem perceber que se tratam de termos *fuzzy,* como quando se fala:

- *O dólar está estável.*
- *O trabalho está parcialmente feito.*

Ou quando se atribui qualidades a um objeto, como "grande", "pequeno", "alto", "baixo", "gordo", "magro", dentre diversos outros.

Todos esses termos são considerados fuzzy, pois referem-se a algo com algum grau de precisão, porém não com a exatidão da lógica clássica.

Quando se fala, “esta sala está quente, abaixe a temperatura do ar-condicionado”, é bem provável que outras pessoas considerem que a mesma sala está com uma temperatura agradável, e outras ainda podem considerar que esteja fria. Isso se deve à imprecisão dos termos QUENTE, AGRADÁVEL e FRIA, pois variam de indivíduo para indivíduo.

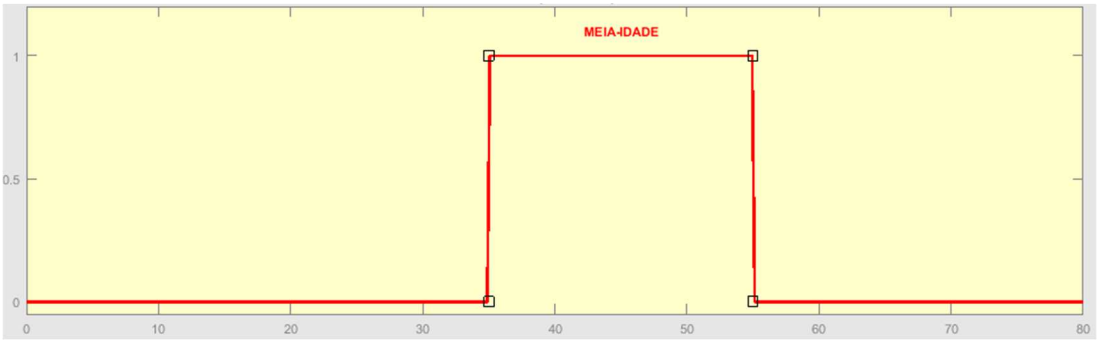
Conforme será visto a seguir, utilizando-se lógica *fuzzy* é possível atribuir valores ou uma função que descreva cada termo linguístico (também denominado por variável linguística ou variável *fuzzy*) a ser utilizado em lógica *fuzzy*. Segundo GONÇALVES (2007), isso permite fornecer uma maneira sistemática de aproximação de fenômenos complexos ou mal definidos.

2.1.3 Conceitos da teoria de conjuntos *fuzzy*

Ao contrário da lógica clássica que caracteriza elementos de um conjunto apenas como zero ou um, verdadeiro ou falso, claro ou escuro, ou qualquer outro termo que caracterize dualidade, a lógica *fuzzy* (alguns autores a chamam de lógica difusa) trata de valores que variam de 0 (zero) a 1 (um)[1], ou seja, um determinado fato pode ser meio verdade 0,5, quase verdade 0,9 ou quase falso 0,1 (SILVA, 2005). Portanto, a lógica *fuzzy* define valores numéricos para palavras ou variáveis linguísticas, de tal forma que seja possível tratá-las matematicamente.

Dessa forma, termos vagos como MEIA IDADE, que pode ser definido como iniciando aproximadamente aos 35 anos e com término em torno dos 55 anos, seria representado na lógica clássica conforme a ***Figura 2.1-1*** (MUKAIDONO, 2001).

Figura 2.1-1 - *Período de meia idade segundo a lógica clássica.*



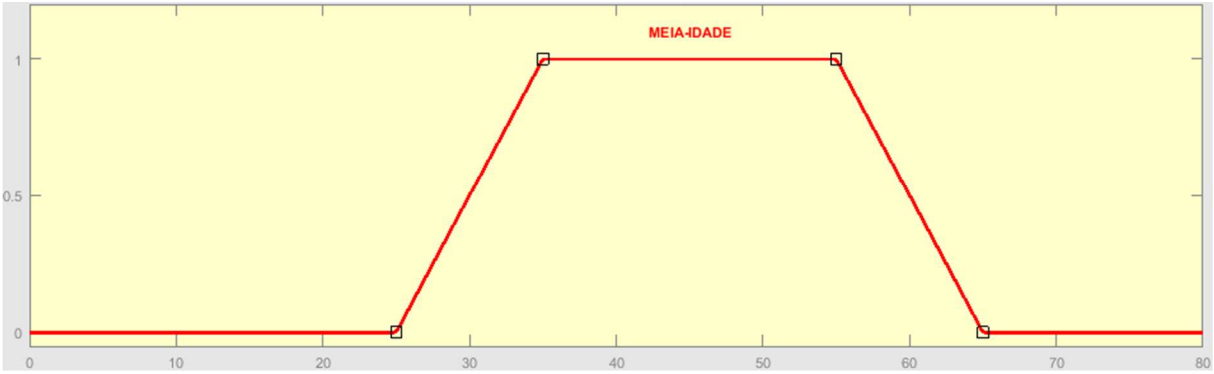
Fonte: Elaborado pelo autor.

1 Utilizado esse intervalo por razões históricas, porém pode ser utilizado qualquer outro.

Porém com o uso da lógica *fuzzy*, a representação gráfica deixa de ser tão abrupta e passa a considerar idades próximas aos 35 e 55 anos, conforme pode ser visto na **Figura 2.1-2** (MUKAIDONO, 2001).

Pela lógica *fuzzy*, indivíduos que possuem de 25 a 65 anos pertencem ao conjunto denominado MEIA IDADE. Porém, o grau de pertinência (ou seja, quanto um indivíduo pertence ou não ao conjunto) de um indivíduo de 25 anos é menor quando comparado a um indivíduo de 30 anos, e que por sua vez é menor quando comparado a alguém de 40 anos. Pela ***Figura 2.1-2*** é possível constatar que o maior grau de pertinência se refere aos indivíduos que possuam de 35 a 55 anos de idade. Quanto aos indivíduos que possuam menos de 25 ou mais de 65 anos, o grau de pertinência é 0 (zero), ou seja, não pertencem ao conjunto MEIA IDADE.

Figura 2.1-2 - *Período de meia idade segundo a lógica fuzzy.*



Fonte: Elaborado pelo autor.

2.1.4 Operações entre conjuntos *fuzzy*

Embora em lógica clássica as operações em conjuntos se resumam basicamente às operações possibilitadas pelos conectivos E, OU e NÃO, na lógica *fuzzy* há diversos operadores disponíveis, os quais se dividem basicamente nas classes denominadas normas-**t** e normas-**s** (SIMÕES, 2007).

Porém neste trabalho serão abordados apenas os operadores intersecção ($A \cap B$), união ($A \ U \ B$) e complemento (A') de conjuntos.

2.1.4.1 Intersecção de conjuntos fuzzy

Seja E o conjunto universo de discurso, x pertencente a E e variando de 0 (zero) a 1 (um), e seja A um subconjunto de B e contido em E. Dessa forma, a intersecção de A com B é denominada $A \cap B$, a qual conterá todos os elementos comuns aos dois conjuntos, A e B (SIMÕES, 2007).

Em *fuzzy*, o vetor de pertinência da intersecção é calculado da seguinte forma:

$$x \in E \quad e \quad x = [0,1]$$
$$\mu_A \cap {}_B(x) = \min[\mu_A(x), \mu_B(x)] \qquad (2.1\text{-}3)$$

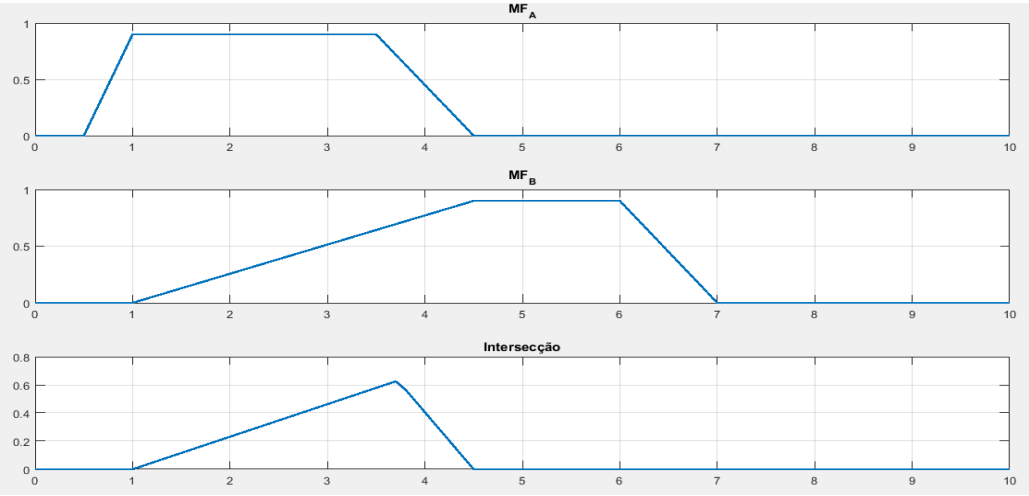
Segue exemplo ilustrando o uso do operador:

E = { x1; x2; x3; x4 } A = { 0,1; 0,6; 0; 0,9 } B = { 0,2; 0; 1; 0,5 }

então $A \cap B$ = { 0,1; 0; 0; 0,5 }

A ***Figura 2.1-3*** exemplifica o operador intersecção aplicado a duas funções de pertinência. Note que o grau que prevalece é sempre o de menor valor entre as duas funções, pois o operador aplica a função **mínimo**.

Figura 2.1-3 - *Exemplo de intersecção entre duas funções de pertinência.*



Fonte: Elaborado pelo autor.

2.1.4.2 União de conjuntos *fuzzy*

Seja *E* o conjunto universo de discurso, x pertencente a *E* e variando de 0 (zero) a 1 (um), e seja *A* e *B* contidos em *E*. Dessa forma, a união de *A* com *B* é denominada *A* U *B*, a qual conterá todos os elementos de *A* e *B* (SIMÕES, 2007).

Em *fuzzy*, o vetor de pertinência da união é calculado da seguinte forma:

$$x \in E \text{ e } x = [0,1]$$
$$\mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)] \qquad (2.1\text{-}4)$$

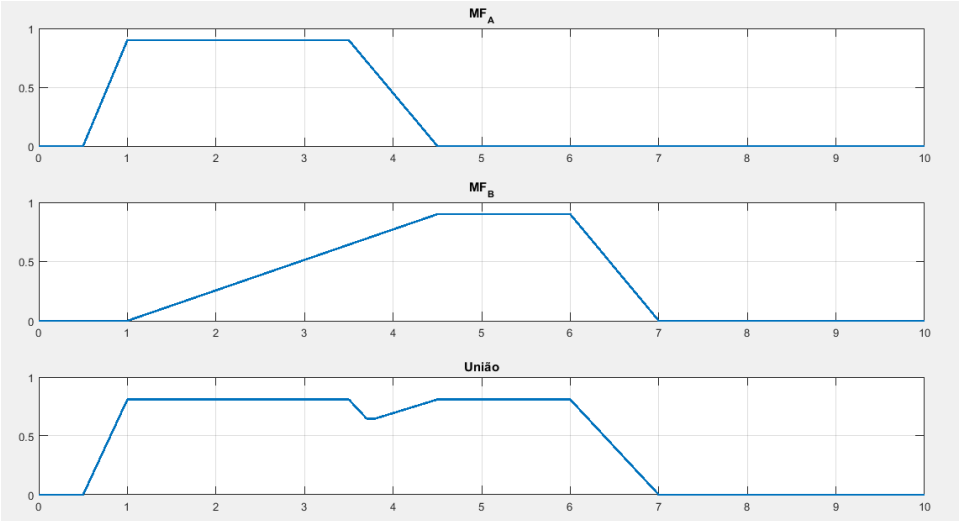
Segue exemplo ilustrando o uso do operador:

E = { x1; x2; x3; x4; x5 } *A* = { 0,8; 0,6; 0; 0; 0,7 } *B* = { 0,2; 0; 0; 1; 0,5 }

então *A* U *B* = { 0,8; 0,6; 0; 1; 0,7 }

A ***Figura 2.1-4*** exemplifica o operador <u>união</u> aplicado a duas funções de pertinência. Note que o grau que prevalece é sempre o de <u>maior</u> valor entre as duas funções, pois o operador aplica a função **máximo**.

Figura 2.1-4 - *Exemplo de união entre duas funções de pertinência.*



Fonte: Elaborado pelo autor.

2.1.4.3 Complemento de um conjunto *fuzzy*

Seja *E* o conjunto universo de discurso e x pertencente a *E*, com *x* variando de 0 (zero) a 1 (um), e seja *A* um subconjunto em relação à *E*. Dessa forma, o complemento de *A* em relação à *E*, é denominado *A'*, o qual conterá todos os elementos de *E* que não sejam membros de *A* (SIMÕES, 2007).

Em *fuzzy*, o vetor de pertinência do complemento é calculado da seguinte forma:

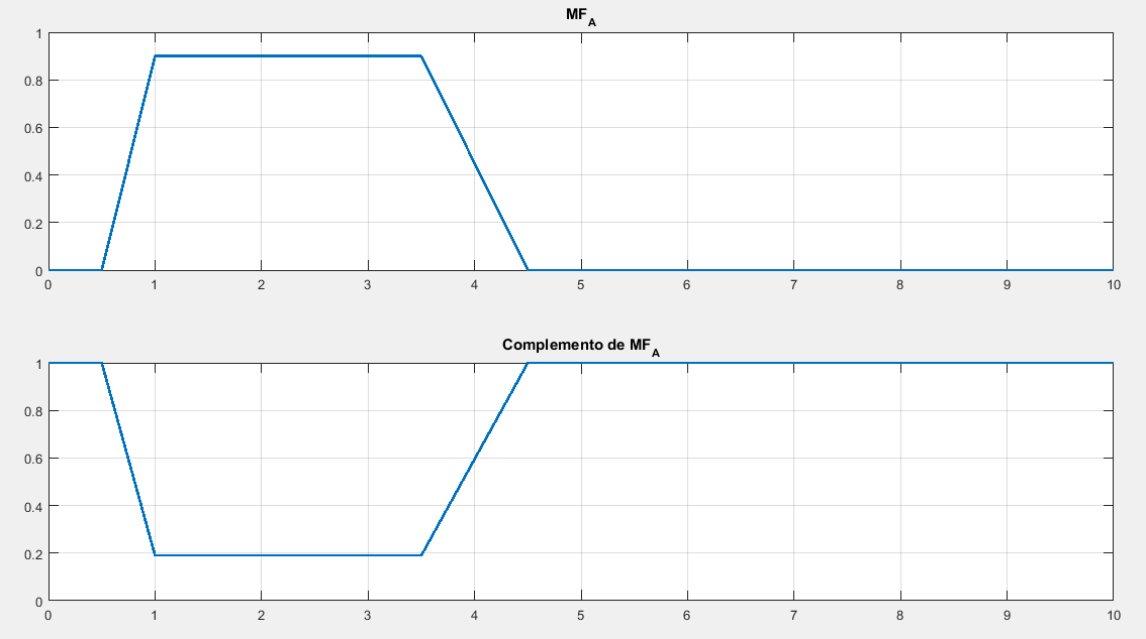
$$x \in E \quad e \quad x = [0,1]$$
$$\mu_{A'}(x) = 1 - \mu_A(x) \qquad (2.1\text{-}5)$$

A seguir segue exemplo ilustrando o uso do operador:

E = { x1; x2; x3; x4 } *A* = { 0,1; 0,6; 0; 0,9 } *A'* = { 0,9; 0,4; 1,0; 0,1 }

A ***Figura 2.1-5*** exemplifica o operador complemento aplicado a uma função de pertinência.

Figura 2.1-5 - *Exemplo do operador complemento aplicado a uma função de pertinência.*



Fonte: Elaborado pelo autor.

2.1.4.4 Probabilidade versus possibilidade

Segundo a teoria de conjuntos *fuzzy*, todos os conjuntos existentes possuem uma imprecisão na definição de seus limites, ao contrário da lógica clássica, onde há uma clara definição (SIMÕES, 2007).

Para os conjuntos fuzzy, a transição da pertinência ou não-pertinência a um determinado conjunto se dá de forma gradual, e não abrupta. Dada essa incerteza sobre a pertinência de determinado elemento estar num determinado conjunto, atribui-se então uma medida da *possibilidade* desse estar no conjunto. Ressalte-se que *possibilidade* não é o mesmo que *probabilidade*, pois o primeiro termo expressa o *grau, qualidade* ou *força* em que o elemento é membro do conjunto, enquanto o segundo expressa a *chance* do elemento ser membro do conjunto (SIMÕES, 2007).

Para facilitar o entendimento, pode-se citar o seguinte exemplo de SIMÕES (2007):

> *De acordo com um relatório de meteorologia, a chance de haver chuva é de 80%, ou seja, essa expressão demonstra a probabilidade de ocorrer chuva. Porém, não indica o grau, qualidade ou força da chuva.*
>
> *Quando se utiliza o conceito de possibilidade, inicialmente se deve criar uma escala de possibilidades, como segue: 1,0 = TEMPESTADE; 0,8 = CHUVA FORTE; 0,6 = CHUVA INTERMITENTE; 0,4 = GAROA; 0,2 = GAROA FINA. Dessa forma, se a possibilidade for 0,4, então haverá uma GAROA.*

2.1.5 Funções de pertinência

Enquanto na lógica clássica a principal propriedade afirma que deva ser atribuído o valor 0 (zero) ou 1 (um) a um determinado elemento, na lógica *fuzzy* deve-se atribuir valores de 0 (zero) a 1 (um) a todos os elementos, ou seja, há infinitas possibilidades de valores, em vez de apenas duas.

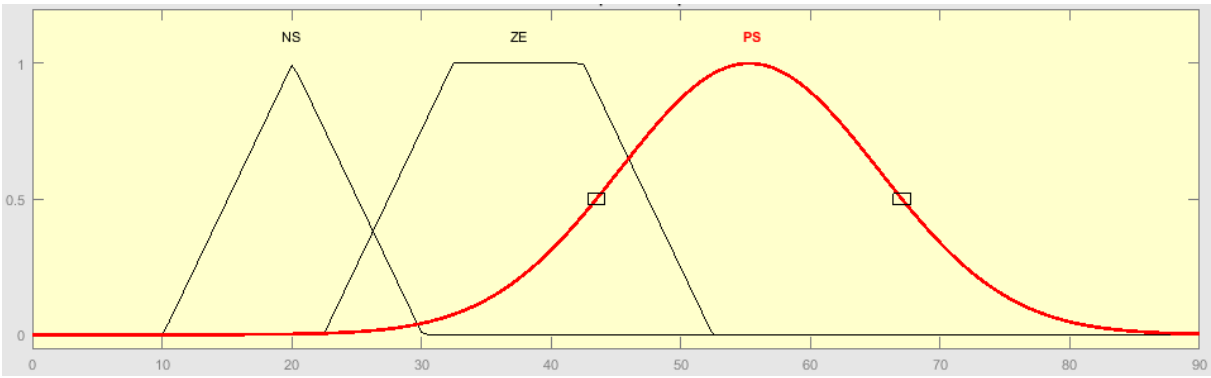
Portanto, há de se atribuir uma função de pertinência (FP) para cada termo *fuzzy* (ver **Seção 2.1.2**) que se queira definir. Essa função pode ser representada através de tabelas, gráficos ou equações.

É importante ressaltar que o universo de discurso de uma variável representa o intervalo numérico de todos os possíveis valores que a mesma possa assumir (SIMÕES, 2007).

Na ***Figura 2.1-6*** há diversos exemplos de funções de pertinência que podem ser atribuídas. Embora essas sejam as funções mais utilizadas, a quantidade e o formato são

escolhidos com base na experiência, na natureza do processo, ou com base em entrevista com especialistas e operadores (SIMÕES, 2007).

Figura 2.1-6 - *Exemplos de funções de pertinência fuzzy.*



Fonte: Elaborado pelo autor.

Não há padrão quanto aos nomes atribuídos aos termos linguísticos, porém esses devem ter algum significado quanto ao universo a que se está modelando. No caso da ***Figura 2.1-6***, foram utilizados nomes comumente usados no inglês, que são: NS (*negative small*), ZE (*zero*) e PS (*positive small*). Outros termos muito utilizados são: NB (*negative big*), NM (*negative medium*), PM (*positive small*) e PB (*positive big*).

O primeiro termo linguístico, da esquerda para a direita, denominado NS (do inglês *negative small*), utiliza a função triangular. Perceba que para valores do conjunto dos números reais abaixo de 10 e acima de 30, o valor *fuzzy* será de 0 (zero). Para o valor 20 será atribuído o valor *fuzzy* 1 (um). E para os demais casos, haverá um valor *fuzzy* correspondente conforme a função triangular.

Para o segundo termo, o ZE (*zero*), utilizou-se a função de pertinência trapezoidal. Essa função é semelhante à triangular, porém possui um topo muito mais largo.

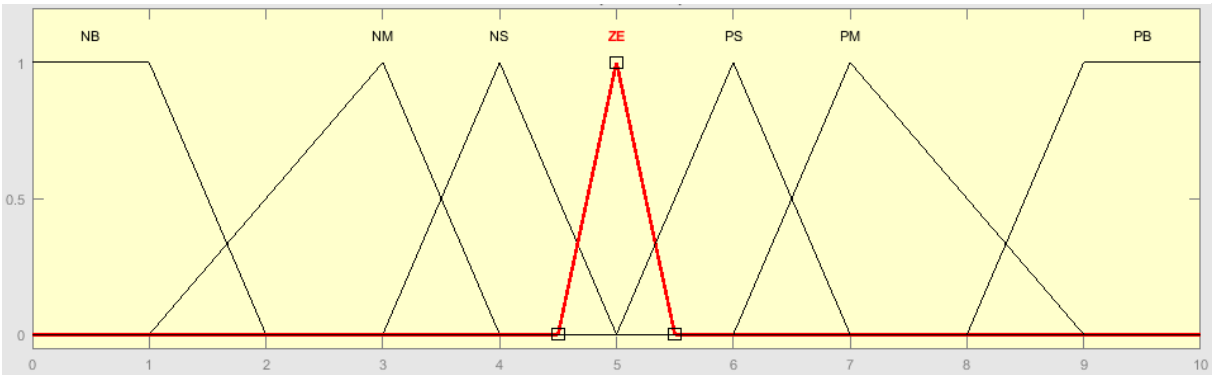
E por último, o termo PS (*positive small*), utilizou-se a função de pertinência Gaussiana, a qual representa uma curva bem suave.

Esses formatos de função são mais frequentemente utilizados devido ao fato de serem facilmente gerados, exigindo pouco custo computacional. Porém, para casos onde o desempenho deva ser o mais suave possível, funções como a Gaussiana são mais indicados. Para os demais casos, as funções triangular e trapezoidal são mais comuns.

A ***Figura 2.1-7*** representa um típico conjunto de funções de pertinência para uma determinada variável. Como necessita-se de um controle mais suave conforme se aproxime do centro, as funções de pertinência ZE, NS e PS são mais estreitas. Conforme se afasta do centro,

as funções passam a ser mais largas. Com isso, haverá uma maior sensibilidade conforme se aproxima do centro.

Figura 2.1-7 - *Funções de pertinência estreitas no centro e mais largas nas extremidades.*



Fonte: Elaborado pelo autor.

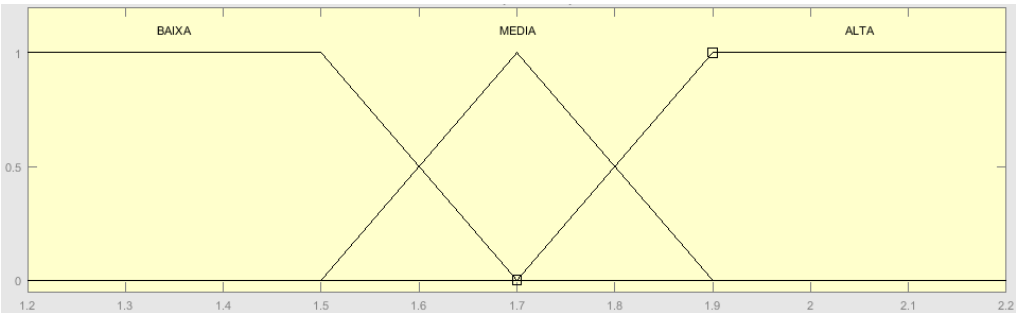
Outro fator que afeta a precisão da variável a ser modelada, é a superposição entre as funções de pertinência especificadas. Conforme SIMÕES (2007), determinou-se experimentalmente que valores de superposição entre 25 e 75% são mais adequados, sendo 50% um compromisso razoável.

Conforme FERNANDES (1997), o tipo e a quantidade de funções de pertinência utilizadas em um determinado sistema dependerão da precisão necessária, estabilidade, facilidade de implementação, manipulação e manutenção da modelagem desenvolvida.

2.1.6 Fuzzificação

A fuzzificação é o processo pelo qual se traduz números reais para o domínio *fuzzy,* através da associação ou cálculo do valor que representa o grau de pertinência da entrada em uma ou mais variáveis *fuzzy* (BILOBROVEC, 2005).

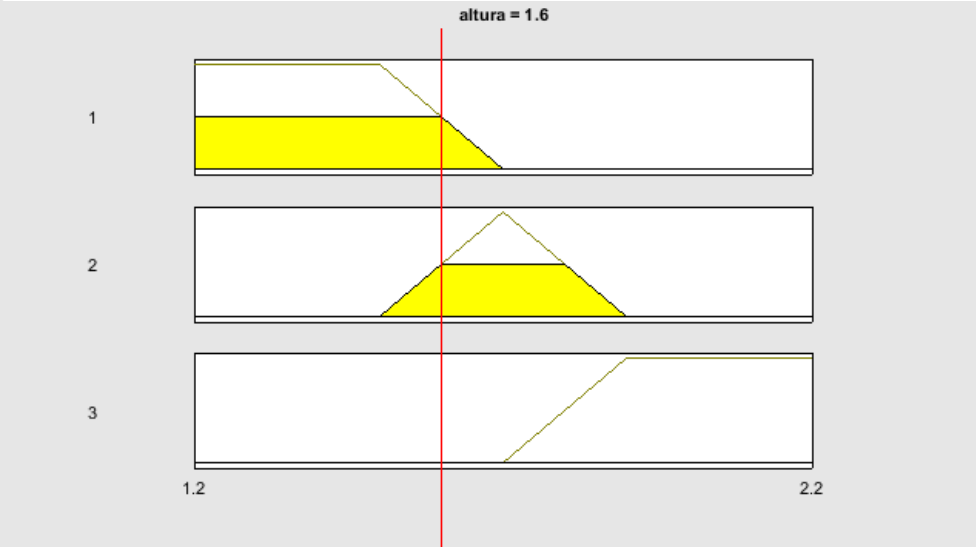
Figura 2.1-8 - *Funções de pertinência para a variável altura.*



Fonte: Elaborado pelo autor.

Tomando como exemplo as funções de pertinência definidas na figura acima, as quais representam a altura de um ser humano, foram definidos três termos: BAIXA, MEDIA e ALTA. Um indivíduo que tenha uma altura inferior a 1,5 m deverá ser considerado de estatura baixa. Alguém que tenha aproximadamente 1,7 m será considerado de estatura mediana. E por último, indivíduos que tenham estatura superior a 1,9 m serão considerados altos. Porém, indivíduos com estatura que estejam compreendidas entre 1,5 m e 1,7 m, e entre 1,7 m e 1,9 m, há um certo grau de imprecisão a respeito de em que grau os qualificar. Para esses casos, haverá um sombreamento entre as funções de pertinência.

Figura 2.1-9 - *Fuzzificação da altura 1,6 m.*



Fonte: Elaborado pelo autor.

Definidas as funções de pertinência, digamos que se queira transpor para o domínio *fuzzy* a altura de 1,6 m. Conforme aparece na ***Figura 2.1-9***, um indivíduo que tenha essa altura, há possibilidade de ser enquadrado tanto como baixo, tanto como de estatura média. Porém, não há possibilidade de ser considerado como alguém de estatura alta. O próximo passo é definir o valor *fuzzy* (de zero a um) da altura especificada, e para tanto, basta verificar qual é o valor no eixo *y* correspondente ao eixo *x*. Conforme se verifica na figura, esse indivíduo será considerado de estatura BAIXA 0,5 e MÉDIA 0,5.

Objetivando obter melhor desempenho computacional no cálculo da fuzzificação, precisa-se definir o menor número possível de funções de pertinência e devem ser geradas através de equações que sejam facilmente computáveis. Por isso a predileção por equações triangulares e trapezóides.

Além do uso de equações para se definir as equações de pertinência, é possível realizar a fuzzificação através do uso de tabelas. Nesse método, haverá o mapeamento de valores discretos para cada uma das variáveis, conforme os quadros 2.1 a 2.3.

Ou seja, através do uso de tabelas, dado um valor de entrada, basta percorrer a tabela para cada função de pertinência e obter o correspondente valor *fuzzy*.

Quadro 2.1-1 - *FP da variável BAIXA*

Valor Real (m)	**Valor *Fuzzy***
1,2	1
1,3	1
1,4	1
1,5	1
1,6	0,5
1,7	0
1,8	0
1,9	0
2,0	0
2,1	0
2,2	0
2,3	0

Quadro 2.1-2 **-** FP da variável MEDIA.

Valor Real (m)	**Valor *Fuzzy***
1,2	0
1,3	0
1,4	0
1,5	0
1,6	0,5
1,7	1
1,8	0,5
1,9	0
2,0	0
2,1	0
2,2	0
2,3	0

Quadro 2.1-3 **-** FP da variável ALTA.

Valor Real (m)	**Valor *Fuzzy***
1,2	0
1,3	0
1,4	0
1,5	0
1,6	0
1,7	0
1,8	0.5
1,9	1
2,0	1
2,1	1
2,2	1
2,3	1

Conforme SIMÕES (2007), quando se utiliza tabelas para definir funções de pertinência, é conveniente realizar o mapeamento de pelo menos 256 valores discretos, para que se obtenha uma precisão razoável.

2.1.7 Base de regras

Considerado o núcleo de todo sistema *fuzzy*, é composto por proposições na forma linguística

Se	x_1	é	A_1	e	x_2	é	A_2	e … e	x_n	é A_n
Então	u_1	é	B_1	e	u_2	é	B_2	e … e	u_m	é B_m

de acordo com informações prestadas por especialistas. Neste momento as variáveis e suas classificações linguísticas devem ser catalogadas e modeladas na forma de funções de pertinência (BARROS, 2006).

Na **Seção 2.1.8**, a qual se segue, será apresentado exemplo demonstrando a aplicação de regras *fuzzy*.

2.1.8 Inferência

A inferência *fuzzy* é responsável por realizar a tradução das proposições da base de regras para um tratamento matemático (BARROS, 2006).

Dos métodos de inferência existentes, o mais largamente utilizado é o método de Mamdani.

Conforme BARROS (2006), o método de Mamdani é baseado na regra de composição de inferência max-min, conforme a seguir:

- Cada regra pode relacionar várias entradas a várias saídas. Uma regra associa cada entrada utilizando o conectivo **E**, através da aplicação da função **mínimo**, como no exemplo a seguir:

  Se temperatura é ALTA **E** há MUITAS pessoas na sala

  Então ar-condicionado no MÁXIMO

- Para relacionar as implicações (ENTÃO) de cada regra, utiliza o conectivo **OU**, através da aplicação da função **máximo**, como no exemplo a seguir:

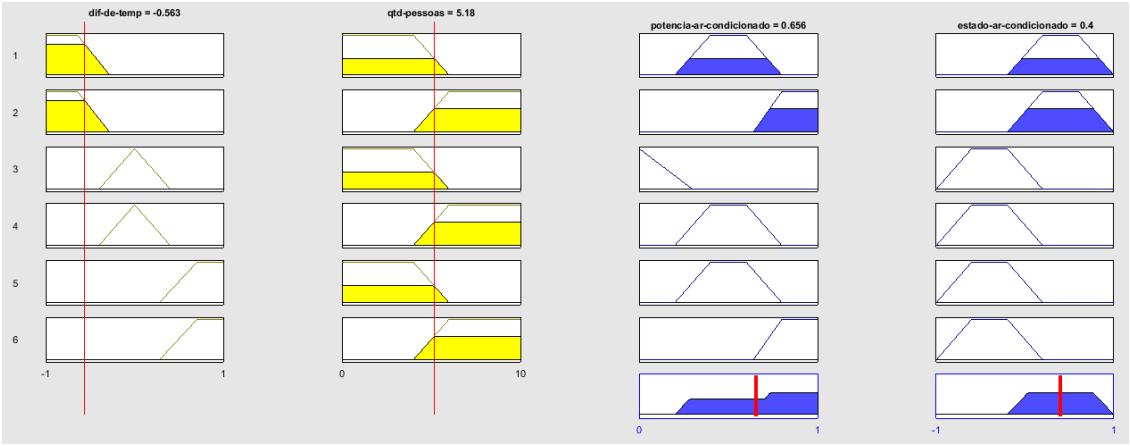
  $regra_1$ **OU** $regra_2$ **OU** $regra_3$ … **OU** $regra_n$

Na **Figura 2.1-10**, no lado esquerdo há a aplicação das proposições de todas as regras, e no lado direito apresenta-se os consequentes da base de regras.

Nesse exemplo, o objetivo é controlar a potência de um sistema de ar-condicionado, sendo possível DESLIGÁ-LO, ligá-lo com potência FRACA ou FORTE, além de informar se deve estar configurado para FRIO ou QUENTE. Há duas variáveis de entrada, uma que informa a diferença de temperatura entre o valor pretendido e a temperatura do ambiente, podendo

assumir os valores de ABAIXO, ZERO ou ACIMA, e a segunda variável especifica a quantidade de pessoas existentes no ambiente (de 0 a 10).

Figura 2.1-10 - *Sistema de ar-condicionado baseado na temperatura e quantidade de pessoas.*



Fonte: Elaborado pelo autor.

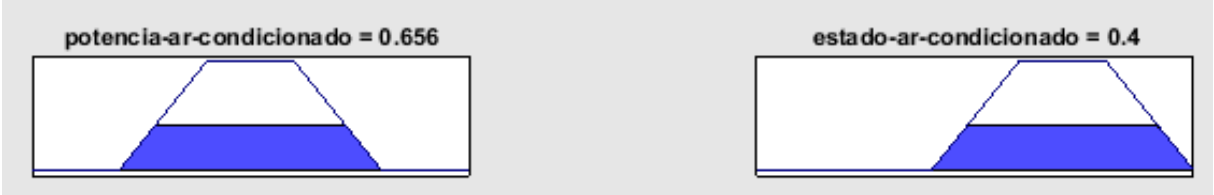
Como o conectivo que o método Mamdani aplica entre as variáveis é o **E**, aplica-se então a função **mínimo** entre as funções de pertinência.

Figura 2.1-11 - *Variável dif-de-temperatura possui maior grau do que a qtd-pessoas para a regra 1.*

Fonte: Elaborado pelo autor.

Perceba na primeira regra, que apesar de quase toda a área da variável **dif-de-temperatura** estar rachurada, o que prevalece na saída é o grau (amplitude) da variável **qtd-pessoas,** conforme ***Figura 2.1-11*** e ***Figura 2.1-12***.

Figura 2.1-12 - *Nos consequentes da regra 1 prevalece o menor (variável qtd-pessoas) grau*

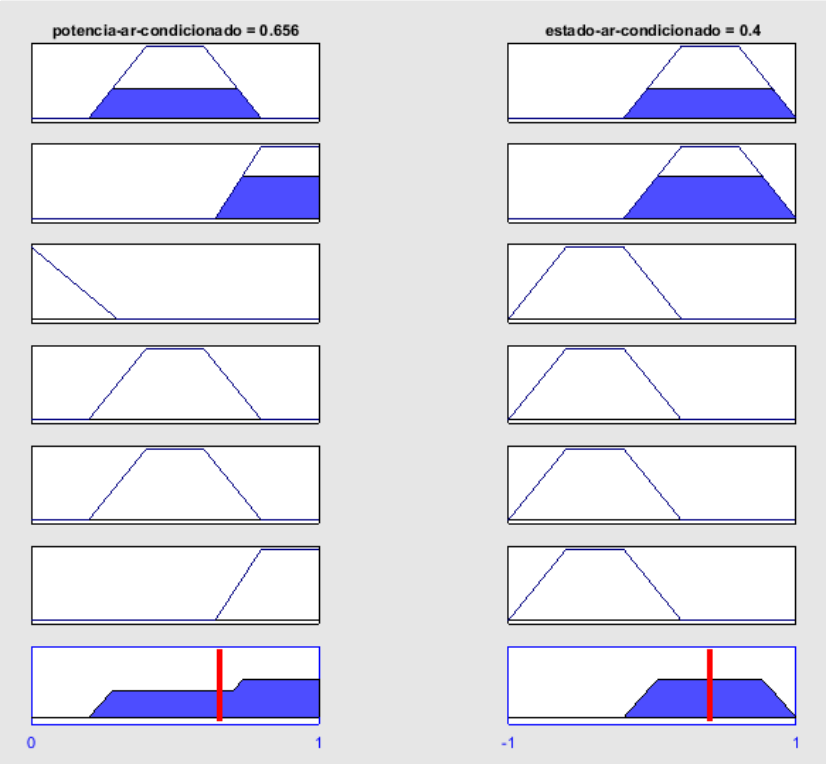


Fonte: Elaborado pelo autor.

Porém, quando se refere a realizar a composição dos consequentes (variáveis de saída), o método Mamdani aplica o conectivo **OU**, que equivale à função **máximo**.

Perceba na ***Figura 2.1-13*** que na variável de saída **estado-ar-condicionado**, o grau de possibilidade que predomina quando se realiza a composição de todas as regras é o da regra 1.

Figura 2.1-13 - *Saída resultante da aplicação de todas as regras.*



Fonte: Elaborado pelo autor.

Quanto à variável de saída **potencia-ar-condicionado**, a sua curva resultante é exatamente a união entre as curvas da primeira e segunda regras, as quais possuem grau maior do que zero.

2.1.9 Defuzzificação

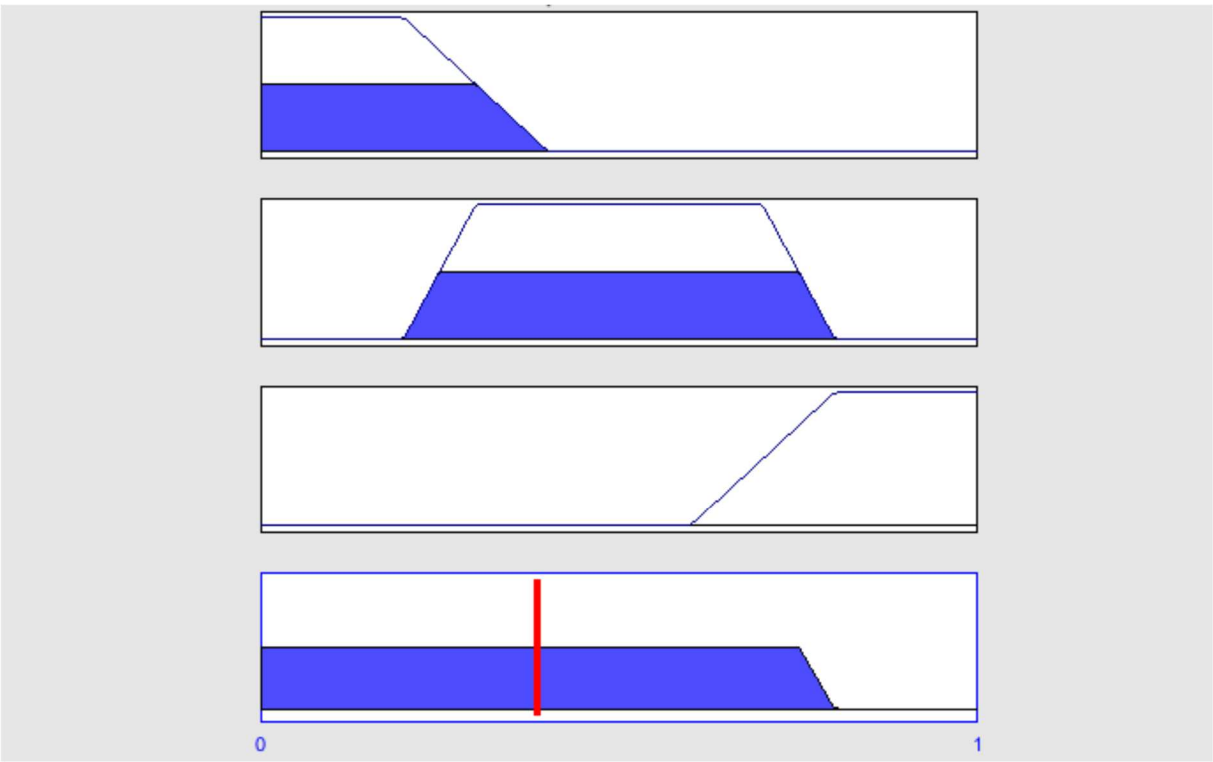
Para um controlador *fuzzy*, o valor recebido na entrada normalmente pertence ao conjunto dos números reais. Como o controlador não entende números reais, realiza-se a fuzzificação para o domínio *fuzzy*. Frequentemente espera-se que a saída também seja um número no domínio dos números reais, e para tanto, necessita-se realizar o processo de deffuzificação (BARROS, 2006).

A deffuzzificação realiza a conversão do valor no domínio *fuzzy* da variável de saída inferida para um valor no domínio dos números reais. Com isso, obtém-se um único número

real que melhor represente a distribuição de possibilidades. Assim, a defuzzificação é a transformação inversa à fuzzificação (SIMÕES, 2007).

Para exemplificar a apresentação da defuzzificação, segue na **Figura 2.1-14** uma possível saída para o exemplo da altura citado no processo de fuzzificação da ***Figura 2.1-9***.

Figura 2.1-14 *- Defuzzificação da altura de 1.6 m, para obtenção de uma poltrona adequada.*



Fonte: Elaborado pelo autor.

Essa figura representa a definição de uma poltrona adequada para um indivíduo de 1,6 m. Há três tipos possíveis de poltrona, classificadas através do uso dos seguintes termos: BAIXA, MÉDIA e ALTA.

O primeiro retângulo da figura representa qual o grau de possibilidade da poltrona BAIXA tendo em vista um indivíduo de 1,6 m. O segundo refere-se a um indivíduo de estatura MÉDIA e o terceiro para alguém considerado ALTO. Para o terceiro caso, o grau de possibilidade é considerado 0 (zero), porém para os demais há valores intermediários, refletindo dessa forma as funções de pertinência da ***Figura 2.1-9***. O quarto retângulo da ***Figura 2.1-14*** representa a resposta do controlador *fuzzy*. O enfoque principal da defuzzificação é exatamente converter essa resposta *fuzzy* para um valor no domínio dos números reais.

Há variados métodos adotados na defuzzificação que podem ser realizados. Aqui serão citados os métodos mais comumente utilizados.

2.1.9.1 Centro de gravidade, centroide ou centro da área (C-o-A)

Método de defuzzificação preferido, apesar de talvez ser o mais complicado, assemelha-se à média aritmética para uma distribuição de frequências de uma dada variável (BARROS, 2006).

A equação utilizada pelo método para valores discretos está apresentada na **Equação (2.1-6)**, onde *x* é o valor a ser obtido no domínio dos números reais.

$$x = \frac{\sum_{i=0}^{N} x_i \, \mu(x_i)}{\sum_{i=0}^{N} \mu\,(x_i)} \qquad (2.1\text{-}6)$$

A variável x_i refere-se aos valores no eixo horizontal da função de pertinência μ. Já a variável $\mu(x_i)$ refere-se ao grau da função de pertinência no ponto x_i, ou seja, o valor no eixo vertical.

2.1.9.2 Centro do máximo (C-o-M)

Semelhante ao método de defuzzificação C-o-A, porém considera apenas os valores de pico de cada função de pertinência. Dessa forma, a área sob cada função de pertinência não desempenha nenhum papel, e apenas os máximos são usados (SIMÕES, 2007).

$$x = \frac{\sum_{i=0}^{N} x_i \mu_i}{\sum_{i=0}^{N} \mu_i} \qquad (2.1\text{-}7)$$

Essa abordagem representa um melhor compromisso para os casos onde há diversas funções de pertinência atuando na saída através de múltiplas regras. Desse modo, caso tenha-se três regras sendo acionadas, e dessas, duas impondo saída ZE e uma para PM, a saída ZE será reforçada (SIMÕES, 2007).

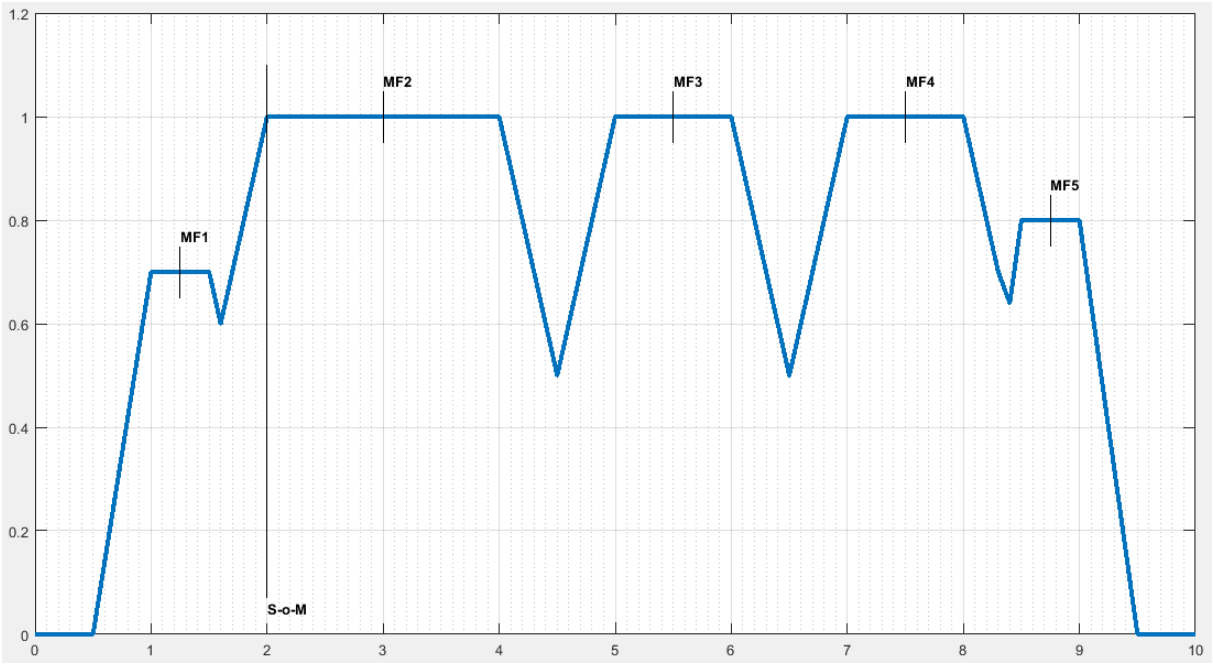
2.1.9.3 Menor dos máximos (S-o-M)

Este método retorna a menor posição no eixo horizontal dentre as funções de pertinência que apresentaram o maior grau de possibilidade, conforme definido na equação abaixo.

$$x = min\big(x(\mu_{i-max})\big) \qquad (2.1\text{-}8)$$

A seguir segue figura exemplificando o método.

Figura 2.1-15 - *Exemplo do método S-o-M.*



Fonte: Elaborado pelo autor.

Pelo exemplo, verifica-se que há 5 funções de pertinência possíveis para a saída, sendo que há três (MF2, MF3 e MF4) com grau de possibilidade igual a 1 (um). Das três funções, a de menor valor no eixo *x* é a MF2. Portanto, o método retorna o valor *2*.

2.1.9.4 Média dos máximos (M-o-M)

Este método retorna a média da posição central no eixo horizontal dentre as funções de pertinência que apresentarem o maior grau de possibilidade, conforme definido na equação abaixo.

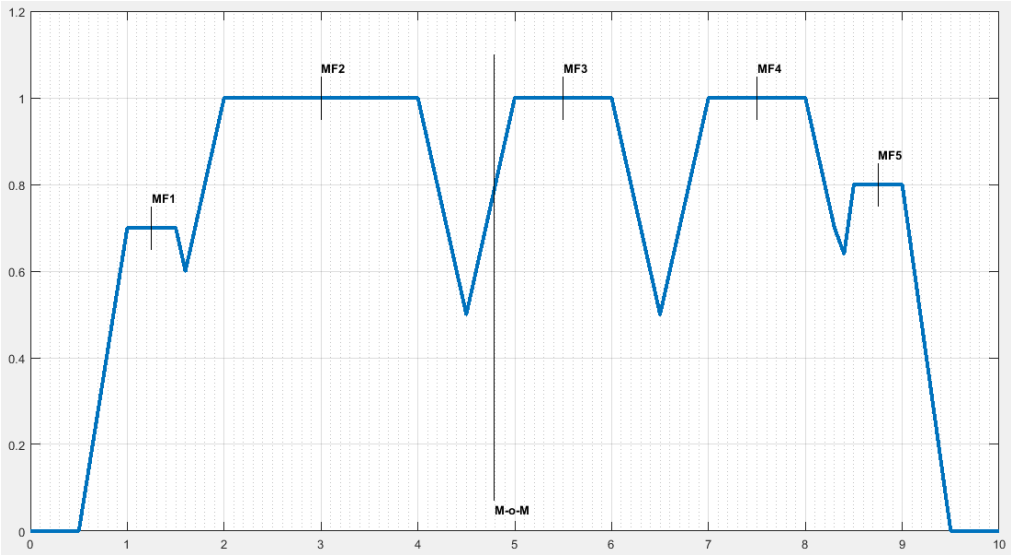
$$x = \frac{1}{N * M} \sum_{i=1}^{N} \sum_{j=x_{min}}^{x_{max}} x_j \left(\mu_{i-max}\right) \qquad (2.1\text{-}9)$$

Inicialmente verifica-se qual o maior grau de possibilidade (eixo vertical) existente para a saída. Após, verifica-se as *N* funções de pertinência que possuem tal grau de possibilidade, somando-se então todos os seus *M* valores (eixo x). E por último, divide-se o resultado da soma pelo produto *N* * *M*.

Semelhante à **Figura 2.1-15**, há 5 funções de pertinência possíveis para a saída, sendo que três apresentam o maior grau de possibilidade dentre todas. Das três funções, obtêm-se o

valor no eixo horizontal de cada função (MF2: de 2 a 4; MF3: de 5 a 6; MF4: de 7 a 8). Somando-se os valores e dividindo-se pela quantidade de segmentos infinitesimais, obtém-se aproximadamente *4,8*. Perceba que se fosse considerado apenas os valores centrais de cada função de pertinência (3, 5,5 e 7,5), o resultado seria *5,3* (16 / 3), porém o método considera a largura de cada função. Como no exemplo a função MF2 possui a maior largura, o valor tende a se aproximar dele.

Figura 2.1-16 - *Exemplo do método M-o-M.*



Fonte: Elaborado pelo autor.

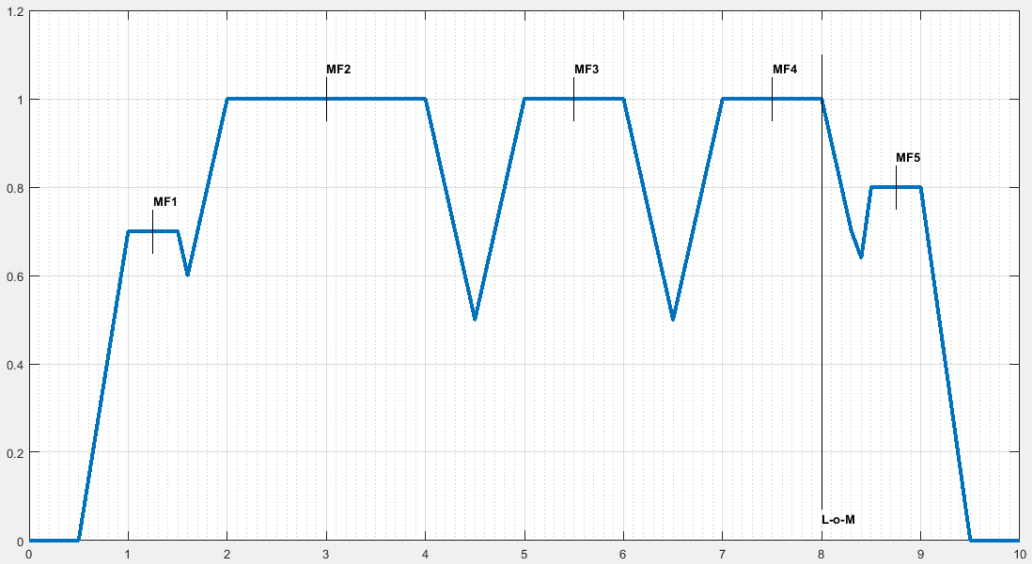
2.1.9.5 Maior dos máximos (L-o-M)

Este método retorna a maior posição no eixo horizontal dentre as funções de pertinência que apresentaram o maior grau de possibilidade, conforme definido na equação abaixo.

$$x = max\left(x(\mu_{i-max})\right) \qquad (2.1\text{-}10)$$

A **Figura 2.1-17** exemplifica a aplicação desse método, na qual, assim como no método S-o-M, há 5 funções de pertinência possíveis para a saída, sendo que há três (MF2, MF3 e MF4) com graus de possibilidade iguais ao máximo. Das três funções, a de maior valor no eixo *x* é a MF4. Portanto, o método retorna o valor *8*.

Figura 2.1-17 - *Exemplo do método L-o-M.*



Fonte: Elaborado pelo autor.

2.1.10 Modelagem utilizando lógica *fuzzy*

Até o momento foram apresentadas as várias partes que constituem a lógica *fuzzy*, porém não se comentou como essas se inter-relacionam.

Nesta seção será apresentado um modelo simplificado para exemplificar como ocorre a modelagem *fuzzy*.

O problema a ser utilizado visa verificar se um determinado banco deve ou não oferecer uma operação de empréstimo a determinado cliente que esteja com saldo negativo. Como o modelo visa ser simples, serão utilizadas poucas variáveis de entrada e apenas uma variável de saída. Segue a relação das variáveis e os respectivos termos linguísticos que serão utilizados.

Variáveis de entrada e respectivos termos linguísticos:

saldo-médio-ultimos-meses:

- ABAIXO
- NEUTRO
- ACIMA

historico-pagamentos-dividas-anteriores

- NAO-PAGA-EM-DIA
- NEUTRO

PAGA-EM-DIA

Variável de saída:

deve-oferecer-emprestimo?

NAO-OFERECER

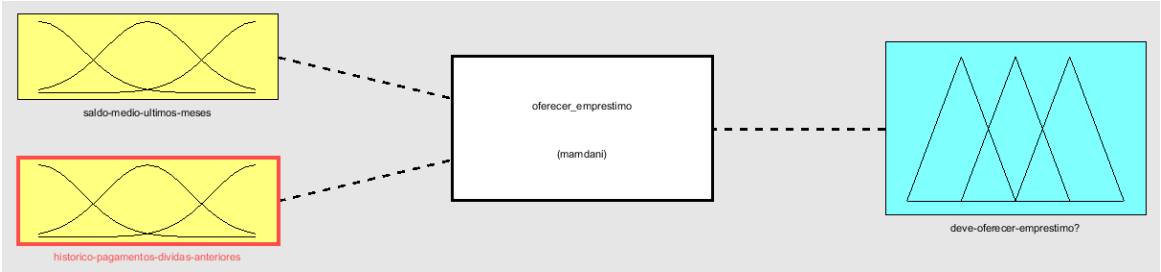
OFERECER-COM-RESSALVA

OFERECER

Para facilitar a exemplificação, todos os valores escalares das variáveis de entrada recebidos pelo módulo de fuzifficação estarão compreendidos entre -1 e 1, e os valores de saída a serem retornados pelo módulo de defuzzificação estarão compreendidos entre 0 e 1.

O método de inferência a ser aplicado será o Mamdani e todas as funções de pertinência serão modeladas com o uso de funções do tipo trapézio. A *defuzzificação* será realizada através do método Centro da Área (C-o-A).

Figura 2.1-18 - *Definição das variáveis de entrada, método de inferência e variável de saída.*



Fonte: Elaborado pelo autor.

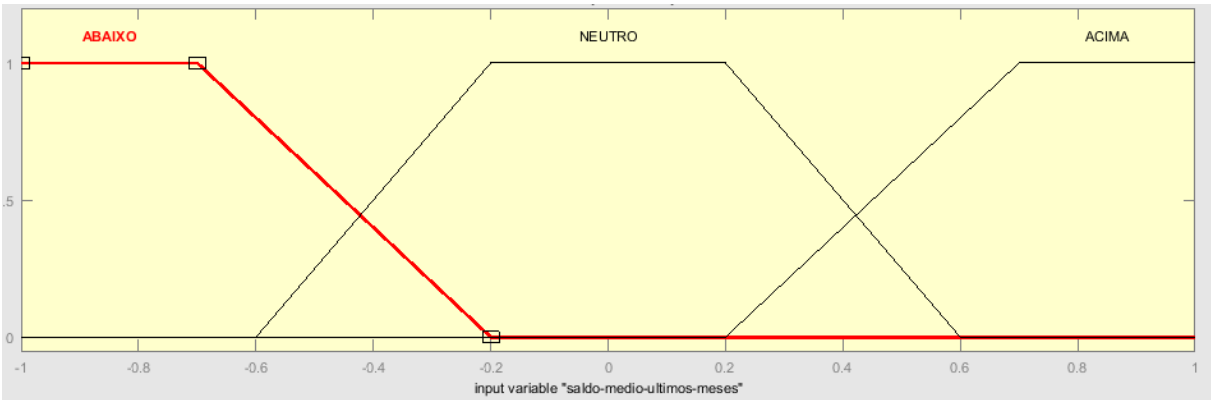
A **Figura 2.1-18** apresenta as duas variáveis de entrada, qual o método de inferência aplicado e a variável de saída. Na figura abaixo constam os parâmetros utilizados pelos métodos de inferência e defuzzificação.

Figura 2.1-19 - *Parâmetros utilizados pelos métodos de inferência e defuzzificação.*

Fonte: Elaborado pelo autor.

Na ***Figura 2.1-20*** é possível verificar as funções de pertinência para cada um dos termos linguísticos utilizados pela variável de entrada “saldo-medio-ultimos-meses”.

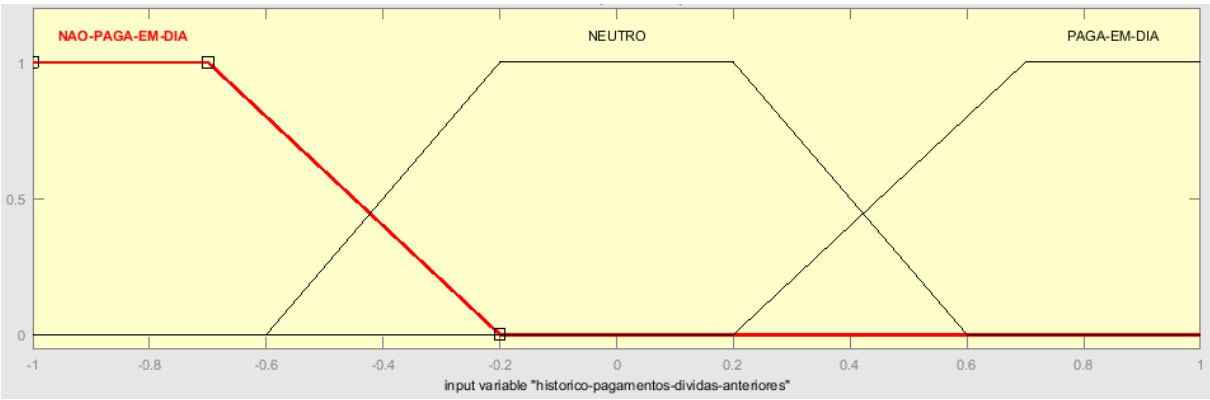
Figura 2.1-20 - *Funções de pertinência de cada termo linguístico da variável “saldo-medio-ultimos-meses”.*



Fonte: Elaborado pelo autor.

Na próxima figura verificam-se as funções de pertinência utilizadas pelos termos linguísticos da variável de entrada “historico-pagamentos-dividas-anteriores”.

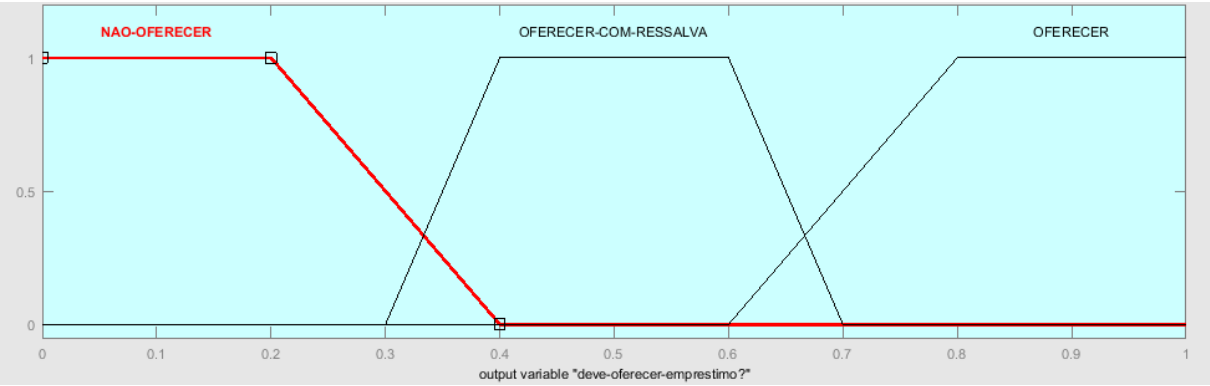
Figura 2.1-21 - *Funções de pertinência de cada termo linguístico da variável de entrada “historico-pagamentos-dividas-anteriores”.*



Fonte: Elaborado pelo autor.

E por último, na **Figura 2.1-22** seguem as funções de pertinência de cada termo linguístico utilizado pela variável de saída.

Figura 2.1-22 - *Funções de pertinência de cada termo linguístico da variável de saída.*



Fonte: Elaborado pelo autor.

Até o momento, definiram-se as variáveis de entrada e seus respectivos termos linguísticos a serem utilizados pelo módulo de fuzzificação, o método de inferência a ser utilizado, a variável de saída e seus termos linguísticos a ser utilizado pelo módulo de defuzzificação, as funções de pertinência a serem utilizadas pelos módulos de fuzzificação e defuzzificação, e o método para que o módulo de defuzzificação possa converter números *fuzzy* em números escalares.

Por último, é necessário especificar o conjunto de regras que serão tratadas pelo módulo de inferência Mamdani, que segue na figura a seguir.

Figura 2.1-23 - *Conjunto de regras a serem tratadas pelo módulo de inferência.*

```
1. If (saldo-medio-ultimos-meses is ABAIXO) and (historico-pagamentos-dividas-anteriores is NAO-PAGA-EM-DIA) then (deve-oferecer-emprestimo? is NAO-OFERECER) (1)
2. If (saldo-medio-ultimos-meses is ABAIXO) and (historico-pagamentos-dividas-anteriores is NEUTRO) then (deve-oferecer-emprestimo? is OFERECER-COM-RESSALVA) (1)
3. If (saldo-medio-ultimos-meses is ABAIXO) and (historico-pagamentos-dividas-anteriores is PAGA-EM-DIA) then (deve-oferecer-emprestimo? is OFERECER) (1)
4. If (saldo-medio-ultimos-meses is NEUTRO) and (historico-pagamentos-dividas-anteriores is NAO-PAGA-EM-DIA) then (deve-oferecer-emprestimo? is OFERECER-COM-RESSALVA) (1)
5. If (saldo-medio-ultimos-meses is NEUTRO) and (historico-pagamentos-dividas-anteriores is NEUTRO) then (deve-oferecer-emprestimo? is OFERECER) (1)
6. If (saldo-medio-ultimos-meses is NEUTRO) and (historico-pagamentos-dividas-anteriores is PAGA-EM-DIA) then (deve-oferecer-emprestimo? is OFERECER) (1)
7. If (saldo-medio-ultimos-meses is ACIMA) and (historico-pagamentos-dividas-anteriores is NAO-PAGA-EM-DIA) then (deve-oferecer-emprestimo? is OFERECER-COM-RESSALVA) (1)
8. If (saldo-medio-ultimos-meses is ACIMA) and (historico-pagamentos-dividas-anteriores is NEUTRO) then (deve-oferecer-emprestimo? is OFERECER) (1)
9. If (saldo-medio-ultimos-meses is ACIMA) and (historico-pagamentos-dividas-anteriores is PAGA-EM-DIA) then (deve-oferecer-emprestimo? is OFERECER) (1)
```

Fonte: Elaborado pelo autor.

Cada uma das regras tem a seguinte forma:

SE $variável_1$ é <TERMO_LINGUÍSTICO> **E** $variável_2$ é <TERMO_LINGUÍSTICO>
ENTÃO $variavel_{saída}$ é <TERMO_LINGUÍSTICO>

Para exemplificar o método de inferência de Mamdani, serão realizados cálculos com alguns valores informados para as variáveis de entrada, conforme quadro abaixo:

Quadro 2.1-4 - *Situações simuladas para o modelo implementado.*

Situação	Variável de Entrada 1	Variável de Entrada 2	Regras Acionadas	Termos de Saída Acionados	Valor Variável de Saída
1	-0,50	0,50	2, 3, 5, 6	OFERECER-COM-RESSALVA OFERECER	0,732
2	0,00	-0,75	4	OFERECER-COM-RESSALVA	0,500
3	-0,75	-0,50	1 e 2	NAO-OFERECER OFERECER-COM-RESSALVA	0,268

Para a primeira situação exemplificada foi informado um saldo médio para os últimos meses ABAIXO de neutro e que o cliente está entre NEUTRO e PAGA-EM-DIA. Com esses parâmetros o método de inferência ativou quatro regras e acionou os termos de saída OFERECER-COM-RESSALVA e OFERECER. O valor defuzzificado foi de 0,732. Portanto, provavelmente seria possível oferecer um empréstimo para o cliente.

Na segunda situação, apesar do cliente possuir nos últimos meses um saldo NEUTRO, o seu histórico é de alguém que NAO-PAGA-EM-DIA. Nesse caso o método de inferência ativou o termo OFERECR-COM-RESSALVA, e o valor retornado pela defuzzificação foi de 0,500.

Quanto à terceira situação, há um cenário de um cliente que possui saldo ABAIXO de neutro nos últimos meses e possui um histórico de que, ou NAO-PAGA-EM-DIA, ou é NEUTRO quanto aos pagamentos. O método de inferência acionou então or termos NAO-OFERECER e OFERECER-COM-RESSALVA, com um valor retornado pela defuzzificação de 0,268.

2.2 MATLAB®

O MATLAB é um software da empresa *The MathWorks, Inc.*, e seu nome refere-se à junção dos termos ***MAT****rix* e ***LAB****oratory*, pois baseia-se profundamente no uso de matrizes.

Amplamente utilizado em universidades e faculdades de diversos cursos, como matemática, ciências e, especialmente, engenharias. Na indústria, alcançou o *status* de ferramenta de pesquisa, projeto e desenvolvimento. Seu pacote padrão contém ferramentas comuns a diversas áreas, como por exemplo o editor de interfaces gráficas GUIDE, além de disponibilizar ferramentas adicionais, denominadas genericamente *toolboxes*. As *toolboxes*

visam resolver problemas específicos, como processamento de sinais, cálculos simbólicos, sistemas de controle, lógica *fuzzy*, dentre outros (GILAT, 2012).

Dentre exemplos de utilização da ferramenta, pode-se citar o uso no projeto e lançamento de sondas espaciais e veículos autônomas da NASA, como *Sojourner*, *Spirit* e *Opportunity*, onde utilizaram *toolboxes* de Redes Neurais, Processamento de Sinais, Processamento de Imagens, PDE (equações diferenciais parciais), sistemas de controle, etc (PALM, 2013).

Devido à extensa quantidade de recursos disponíveis no MATLAB, este trabalho se concentrará na apresentação e explicação breve das *toolboxes* ***Simulink*** e ***Fuzzy Logical Toolbox***.

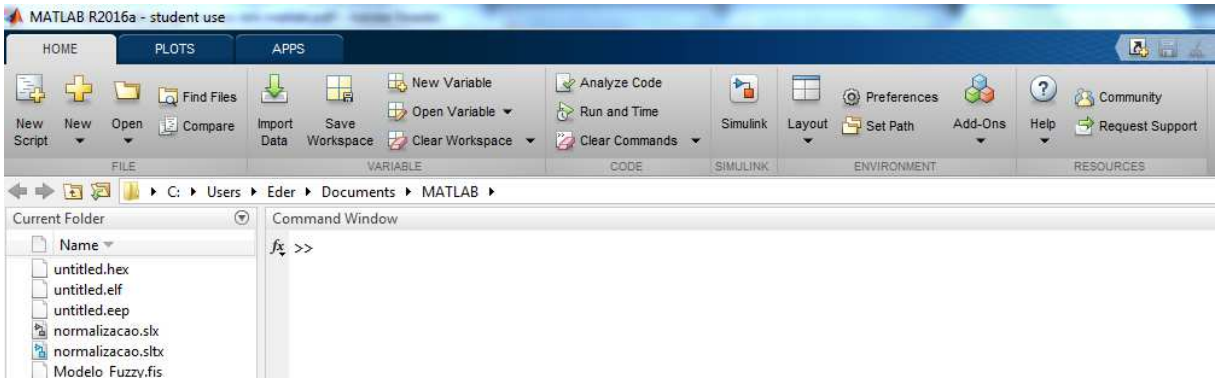
Todos os exemplos e explicações apresentados serão baseados na versão do estudante 2016, *release* a (MATLAB R2016a – Student Use).

2.2.1 Área de trabalho do MATLAB

Ao iniciar o MATLAB, aparecerá sua área de trabalho, a qual contém uma barra de menus, barra de ferramentas, lista de arquivos correntes e uma janela de comandos, conforme a **Figura 2.2-1**.

O MATLAB apresenta o *prompt* (“>>”) para indicar que está pronto para receber comandos. Para entrar com algum comando, posicione o cursor logo após o *prompt*, e digite o comando.

Figura 2.2-1 - *Área de trabalho do MATLAB.*



Fonte: Elaborado pelo autor.

2.2.2 Simulink

A *toolbox* Simulink foi construída para ser utilizada no MATLAB, sendo amplamente aplicada na indústria para modelar sistemas complexos e processos que são difíceis de serem modelados com um simples conjunto de equações. Possui uma interface gráfica com o usuário que utiliza diversos elementos, denominados blocos, na criação da simulação de sistemas dinâmicos, os quais frequentemente envolvem testes do tipo *hardware-in-the-loop* (PALM, 2013). É possível através da interface gráfica posicionar, redimensionar, rotular, especificar parâmetros e interconectar os blocos, no intuito de descrever complexos sistemas.

2.2.2.1 Acessando a *toolbox* Simulink

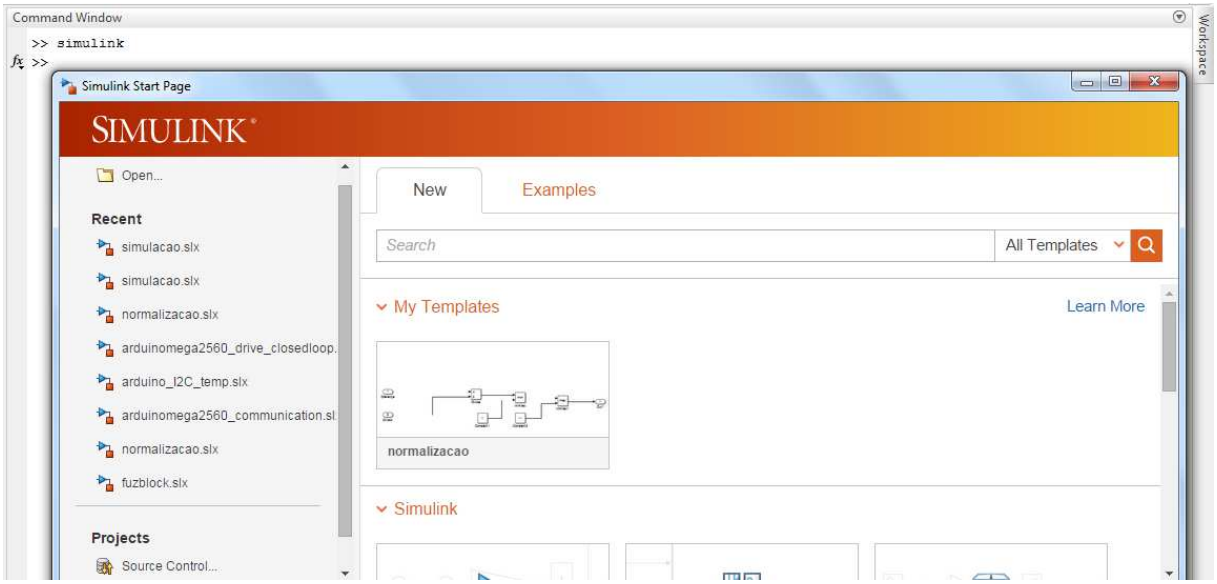
Para acessar a *toolbox* Simulink, inicie o MATLAB, e no *prompt* digite "`simulink`". Será aberta a janela conforme ***Figura 2.2-2***. Nessa janela, serão apresentados os arquivos de modelagem abertos recentemente, botão ("Open...") para abrir um arquivo salvo no computador, opção para criar modelagem a partir de um tipo ou exemplo, e diversos exemplos de aplicação.

Abra um arquivo de modelagem existente ou crie um novo para que seja aberta a janela de modelagem do Simulink, conforme ***Figura 2.2-3***.

Utilize o botão *Library Browser* ( ) para abrir o navegador de bibliotecas do Simulink, o qual contém diversos blocos que podem ser utilizados na simulação.

Para definir o tempo total de execução da simulação, utilize a *input* "*Simulation stop time*" (10.0), localizada na barra de ferramentas.

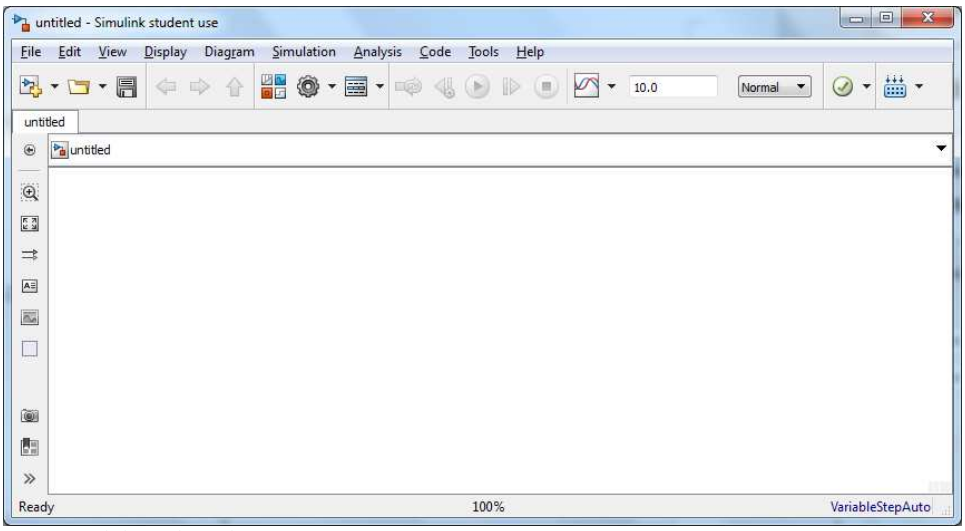
Figura 2.2-2 - *Janela inicial do Simulink.*



Fonte: Elaborado pelo autor.

O botão da barra de ferramentas “Model Configuration Parameters” () permite alterar diversas configurações do Simulink, como momento inicial (*Start time*) e final (*Stop time*) da simulação, nomes de variáveis de *input* originadas do *workspace* do MATLAB, especificação de *hardware* caso necessite de integração do Simulink com algum *hardware* externo, como o Arduino, dentre muitas outras configurações.

Figura 2.2-3 - *Janela Model do Simulink.*



Fonte: Elaborado pelo autor.

2.2.2.2 Diagramas de simulação

Os modelos desenvolvidos no Simulink baseiam-se na construção de diagramas através do uso de blocos, os quais estão localizados no *Library Browser*, conforme ***Figura 2.2-4***.

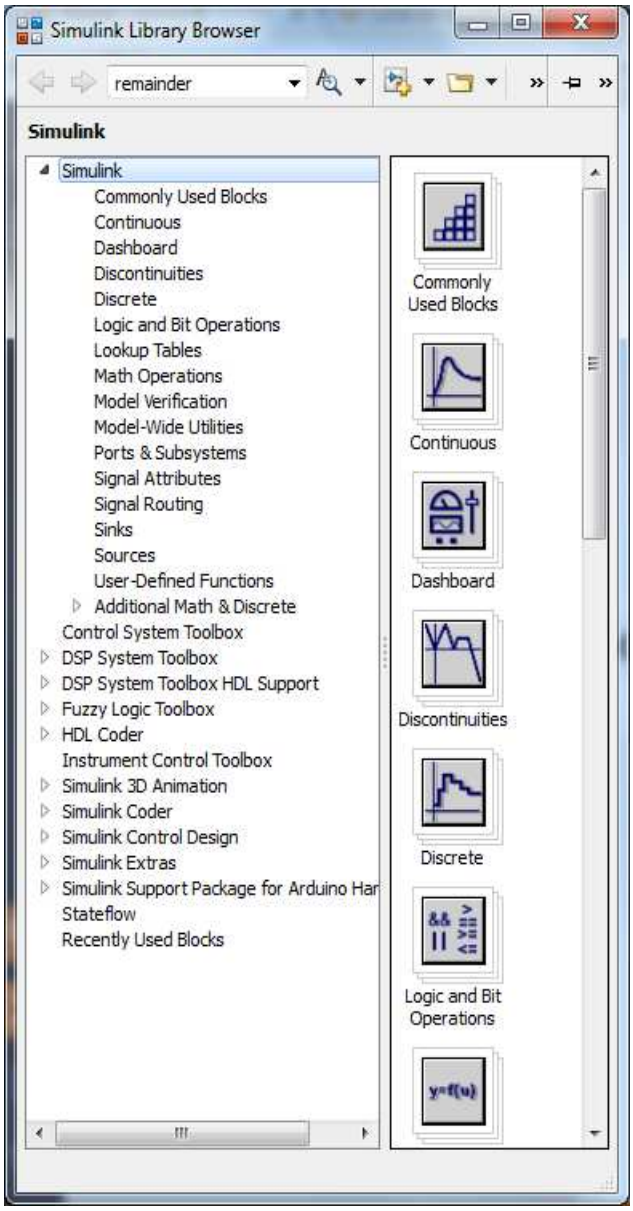
Para utilizar um bloco do *Library Brower*, clique e arraste para a janela *Model* do Simulink. Com o bloco na janela *Model*, é possível redimensionar o seu tamanho. Diversos blocos permitem também a atribuição de um nome. Outros exigem a digitação de parâmetros para que possam ser utilizados.

Outra forma de incluir blocos no diagrama é clicando em algum lugar vazio da janela *Model*, e mantendo o *mouse* parado por alguns segundos. Aparecerá então a imagem de uma lupa. Clique-a e digite o nome do bloco a ser inserido. Aparecerá uma lista de blocos que satisfazem o nome especificado.

Interligue os blocos clicando nas setas que os circundam e arrastando-as até outro bloco, de forma que aparecerá uma seta desde o bloco de origem até o bloco de destino. Dê duplo clique na seta de interligação para nomeá-la.

Após construído o diagrama, utilize o botão da barra de ferramentas *Run* ( ) para executar a simulação.

Figura 2.2-4 - *Library Browser.*



Fonte: Elaborado pelo autor.

Abaixo segue quadro contendo exemplos de blocos usados em diagramas.

Quadro 2.2-1 - *Exemplos de blocos do Library Browser do Simulink.*

Bloco	Descrição
	Elemento de entrada, o qual permite informar um valor constante.
	Distribui o sinal de entrada para dois barramentos distintos.
	Une dois sinais de barramentos distintos para um mesmo sinal em um único barramento.
	Realiza a soma de dois sinais para uma única saída.
	Equivalente ao bloco “*Sum*”.
	Realiza o produto de dois valores.
	Realiza a divisão de dois valores.
	Multiplica o sinal de entrada pelo valor especificado.
	Sinal de entrada.
	Sinal de saída.
	Apresenta o sinal em um simulador de osciloscópio.
	Representa uma modelagem, a qual é construída dentro do bloco.
	Operador de integração.
	Operador de derivada.
	Bloco utilizado para executar o sistema *fuzzy* especificado como parâmetro.

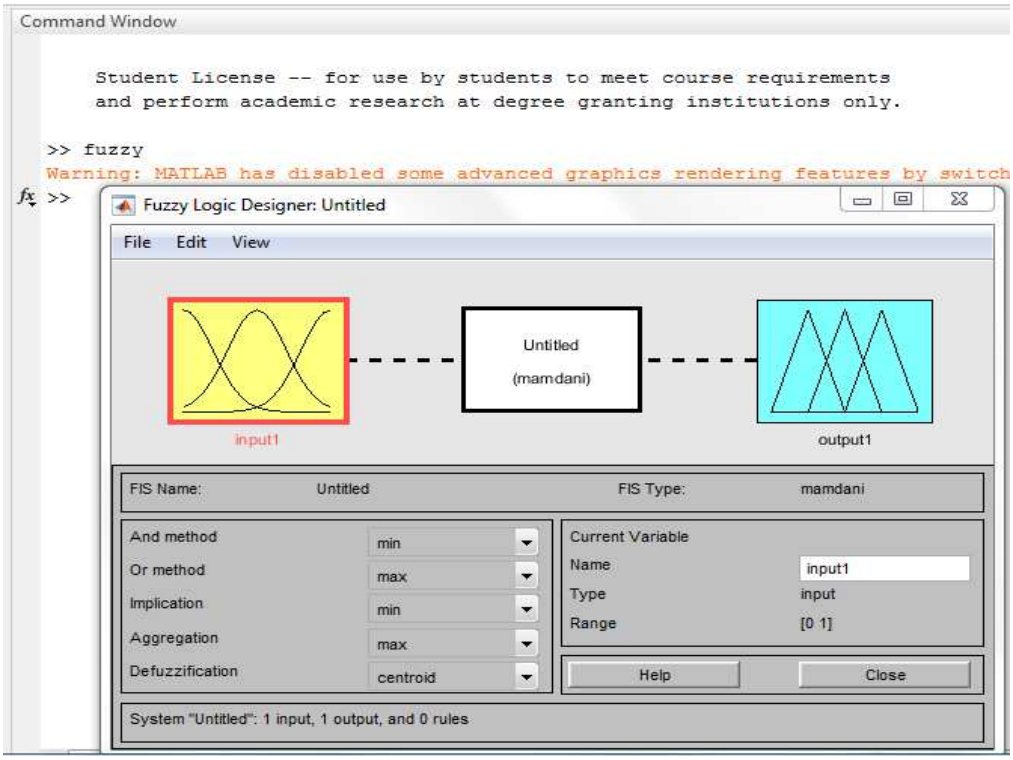
2.2.3 Fuzzy Logic Toolbox

A *toolbox* Fuzzy destina-se a modelar sistemas que envolvam lógica *fuzzy*, possuindo uma interface gráfica com o usuário que permite criar modelos utilizando os métodos de inferência Mamdani e Sugeno (AMENDOLA, 2005). Neste trabalho, porém, será abrangido apenas o método Mamdani, o qual é utilizado na solução proposta, e algumas das principais funcionalidades.

2.2.3.1 Acessando a *Fuzzy Logic Toolbox*

Para acessar a *toolbox* Fuzzy, inicie o MATLAB, e no *prompt* digite "`fuzzy`". Será aberta a janela conforme ***Figura 2.2-5***, onde por padrão apresenta-se um modelo simples contendo uma *input* e uma *output*, Mamdani como método de inferência e o método centroide para defuzzificação. Cada variável contém três funções de pertinência, as quais utilizam a função triangular e intervalo de domínio de 0 (zero) a 1 (um).

Figura 2.2-5 - *Janela inicial da Fuzzy Logic Toolbox.*

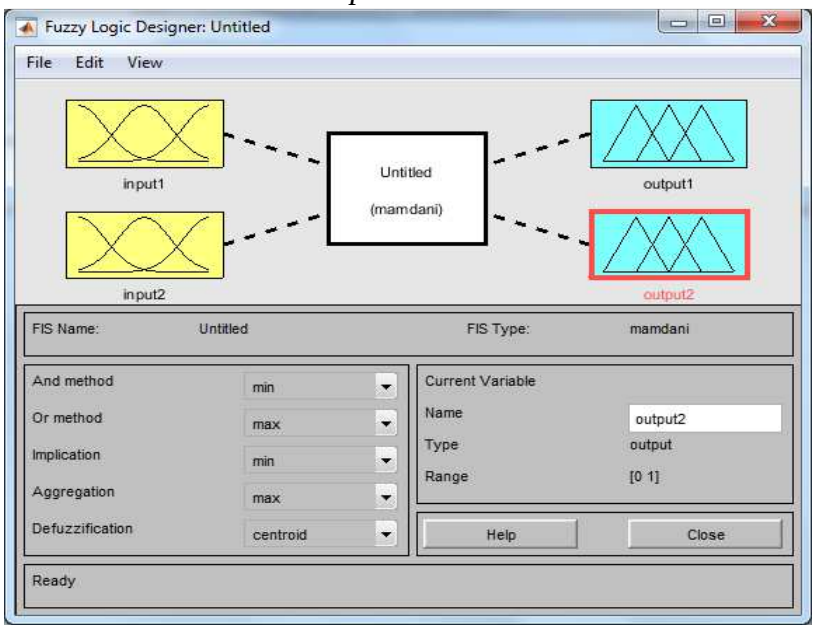


Fonte: Elaborado pelo autor.

2.2.3.2 Incluir e Editar Variáveis de entrada (*input*) e saída (*output*)

Para adicionar uma variável, acesse o menu “Edit”, depois em “Add Variable...”, e clique no tipo de variável que se queira incluir (“Input” ou “Output”). Após criada a variável, essa será adicionada à tela inicial da *toolbox*, conforme figura a seguir. Para renomeá-la, clique no respectivo retângulo e altere o nome no campo “Name”.

Figura 2.2-6 - *Janela inicial da toolbox após a inclusão de mais duas variáveis.*

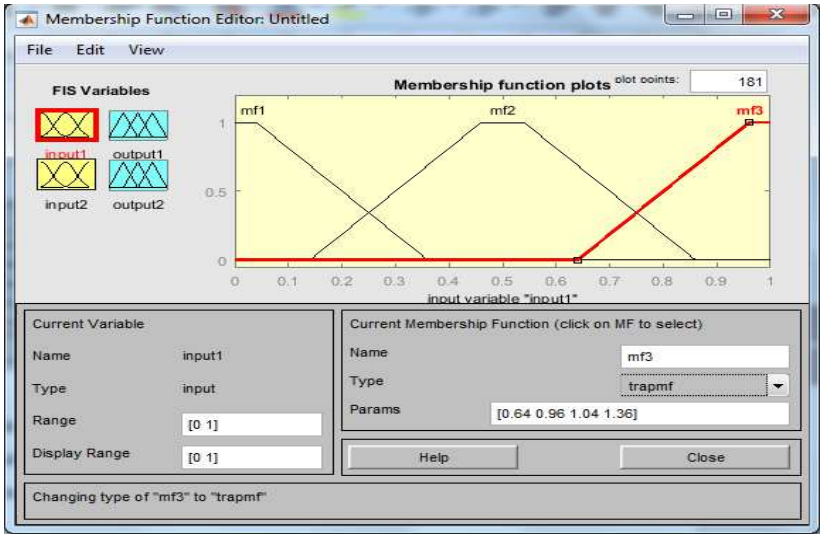


Fonte: Elaborado pelo autor.

Para editar os parâmetros, clique duas vezes no retângulo da variável. Será aberta a janela do *Membership Function Editor*, conforme ***Figura 2.2-7***. Nessa janela, clique no retângulo da variável que se queira consultar ou editar. Serão apresentados os intervalos de domínio (“Range”) e apresentação (“Display Range”) da variável e suas funções de pertinência, as quais representam os termos linguísticos tratados pela lógica *fuzzy*. Clique ao longo da função de pertinência plotada no gráfico para que sejam apresentados seus dados. É possível alterar o nome (“Name”), tipo (“Type”) e parâmetros de plotagem da função (“Params”).

Caso necessário incluir ou remover uma função de pertinência, clique no menu “Edit” e depois em alguma das seguintes opções: “Add Mfs...” (para adicionar diversas funções de pertinência de mesmo tipo), “Add Custom MF...” (para adicionar uma função de pertinência), “Remove Selected MF” (remove a função de pertinência selecionada no gráfico) ou “Remove All Mfs” (remove todas as funções de pertinência da variável selecionada).

Figura 2.2-7 - *Janela "Membership Function Editor", para editar funções de pertinência.*



Fonte: Elaborado pelo autor.

2.2.3.3 Remover Variáveis de entrada (*input*) e saída (*output*)

Para remover uma variável, selecione-a e acesse o menu "Edit" e depois clique em "Remove Selected Variable".

2.2.3.4 Editar Regras *Fuzzy*

Para editar as regras utilizadas pelo método de inferência *fuzzy*, abra o menu "Edit" da tela inicial da *toolbox*, e depois clique em "Rules...", sendo aberta a janela "Rule Editor", conforme figura a seguir.

No método de inferência Mamdani todas as regras seguem a seguinte forma:

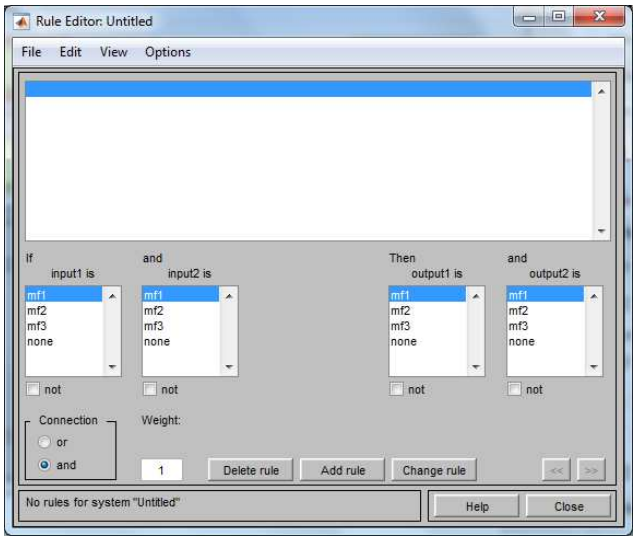
```
SE      variável1 é <TERMO_LINGUÍSTICO> <CONECTIVO> variável2 é <TERMO_LINGUÍSTICO>
ENTÃO   variavel1_saída é <TERMO_LINGUÍSTICO> E variavel2_saída é <TERMO_LINGUÍSTICO>
```

Para elaborar as regras, selecione a função de pertinência desejada para cada variável de entrada e o respectivo conectivo ("AND" ou "OR"). Por padrão o conectivo utilizado para se avaliar a condição SE ("if") é o E ("AND"), porém é possível alterar para OU ("OR"). Marque então as funções de pertinência que serão ativadas na saída. Após, clique no botão "Add rule" para adicionar a regra na lista de regras.

Para deletar uma regra, selecione-a e clique no botão "Delete rule".

Se necessário alterar uma regra, selecione-a, altere as funções de pertinência a serem verificadas ou ativadas para cada variável e clique no botão "Change rule".

Figura 2.2-8 - *Janela "Rule Editor", para edição das regras do método de inferência.*



Fonte: Elaborado pelo autor.

2.2.3.5 Testar Fuzzificação e Defuzzificação

Após definido o método de inferência e defuzzificação, as variáveis de entrada, saída e respectivas funções de pertinência, e o conjunto de regras, a *toolbox* permite testar valores de entrada no intuito de verificar o comportamento das variáveis de saída. Para tanto, na janela inicial acesse o menu "View" e clique em "Rules", sendo então aberta a janela "Rule Viewer", conforme ***Figura 2.2-9***.

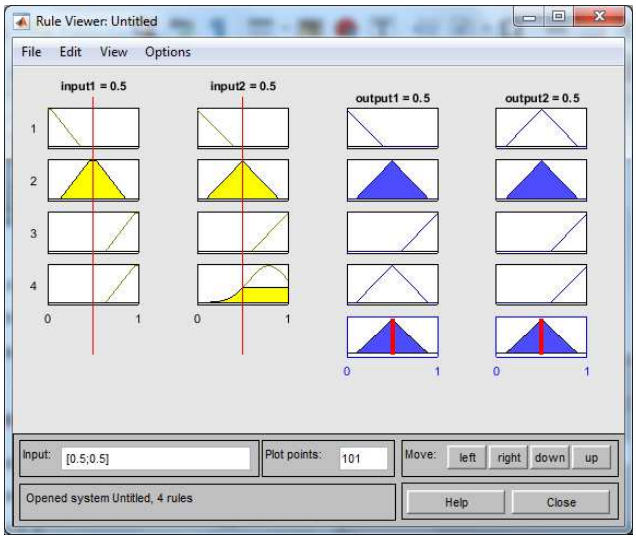
Na janela "Rule Viewer" é possível digitar valores para cada uma das variáveis de entrada no campo "Input" ou então arrastar a barra vertical vermelha com o *mouse*.

Definidos os valores de entrada, a *toolbox* aciona a fuzzificação, processo no qual converte-se o número escalar para um número *fuzzy* com base nas funções de pertinência.

Após, o método de inferência verifica quais regras serão consideradas VERDADEIRAS (na figura, corresponde ao rachurado em amarelo para cada variável de entrada). Definidas as regras pelo método de inferência, acionam-se as variáveis de saída (na figura, corresponde ao rachurado em azul para cada variável de saída) e o método de defuzzificação calcula os graus de pertinência para cada variável.

Por último, o método de inferência calcula os valores de saída da lógica *fuzzy* com base nos valores obtidos pela defuzzificação de cada regra e variável de saída.

Figura 2.2-9 - *Janela "Rule Viewer", para testar a fuzzificação e defuzzificação.*

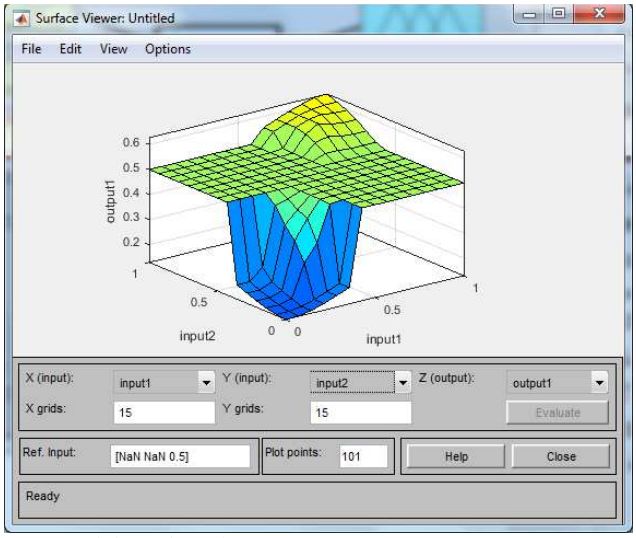


Fonte: Elaborado pelo autor.

Uma forma muito comum de verificar um sistema *fuzzy* é através da geração de gráficos tridimensionais, no qual é possível verificar o valor de saída de uma dada variável de saída conforme os valores de duas variáveis de entrada.

Para gerá-los na *toolbox*, acione o menu "View" na janela inicial, e depois clique em "Surface". Será aberta a janela "Surface Viewer", conforme figura a seguir.

Figura 2.2-10 - *Janela "Surface Viewer", para geração de gráficos do sistema.*



Fonte: Elaborado pelo autor.

Selecione as variáveis para as quais o gráfico será gerado através das combos "X (input)", "Y (input)" e "Z (output)".

2.3 Arduino®

Criado em 2005 na Itália por um grupo de pesquisadores, o Arduino® é uma plataforma eletrônica *open source* que contempla tanto *hardware* quanto *software*, com o objetivo de facilitar a criação de projetos eletrônicos (ARDUINO, 2016).

Existem diversos tipos de placas Arduino disponíveis, cada uma com características e finalidades distintas, como por exemplo: Arduino Nano, Arduino Uno, Arduino Mega, Arduino Due, etc.

Além das placas Arduino convencionais, há também placas denominadas *shields* que foram desenvolvidas para algum propósito específico e que permitem serem encaixadas em algum tipo de Arduino, como: ETHERNET, para conexão em rede de computadores; GSM, para conexão a redes de telefonia móvel; TFT TOUCH DISPLAY, para uso de *displays* LCD *touch*; WIFI, para conexão a redes sem fio; dentre outras.

Como na solução proposta por este projeto será utilizada a placa Arduino UNO, todas as especificações técnicas informadas a seguir se basearão nela.

2.3.1 Arduino UNO

A placa Arduino UNO, apresentada na figura abaixo, é uma das mais utilizadas, sendo que muitas das *shields* são construídas para que sejam encaixadas nesta placa.

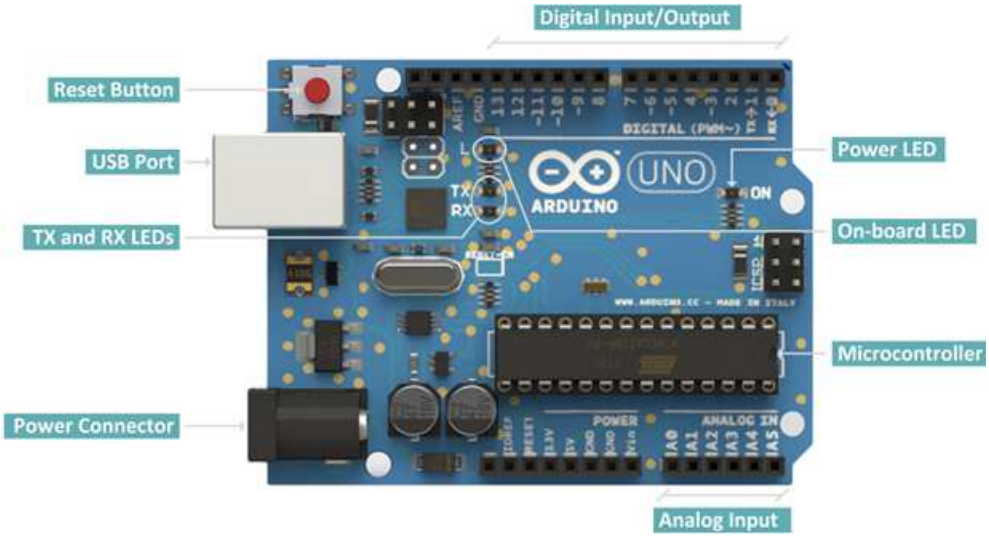
Possui um microcontrolador ATmega328P da marca Atmel®, sendo esse o componente responsável por armazenar e executar o código-fonte. Trata-se de um microcontrolador CMOS de baixa potência, com instruções de 8 bits, arquitetura RISC, maioria das instruções executadas em um único ciclo de *clock*, memória *flash* de 32 KB, memória RAM de 2 KB e memória EEPROM de 1 KB. A memória *flash* permite guardar o código-fonte a ser executado, enquanto a memória RAM armazena os dados e instruções durante a execução do programa e a memória EEPROM permite persistir informações obtidas durante a execução do programa, as quais poderão ser recuperadas futuramente.

Para que o microcontrolador possa executar as instruções, há um cristal que oscila na frequência de 16 MHz.

Para a gravação de programas no microcontrolador e alimentação da placa, há disponível um conector USB 2.0 tipo B.

O botão de *reset* permite reiniciar a execução do programa existente no microcontrolador.

Figura 2.3-1 - *Placa Arduino UNO. Vista superior.*



Fonte: http://resources.intenseschool.com/

O conector Jack (*Power Connector*) possibilita alimentar a placa Arduino sem o uso de um computador conectado à USB.

Os leds TX e RX indicam a transmissão de dados pela porta serial USB.

O led "Power LED" indica se a placa está ligada ou não a uma fonte de alimentação.

O "On-board LED" está conectado ao pino 13 da placa, podendo ser utilizado.

Há 14 pinos digitais para entrada e saída de dados, numerados de 0 a 13, os quais permitem ler ou gravar níveis de tensão de 0V e 5V.

Além dos pinos digitais, há seis pinos analógicos, numerados de A0 a A5. Esses pinos permitem realizar a leitura e gravação de níveis de tensão de 0V a 5V.

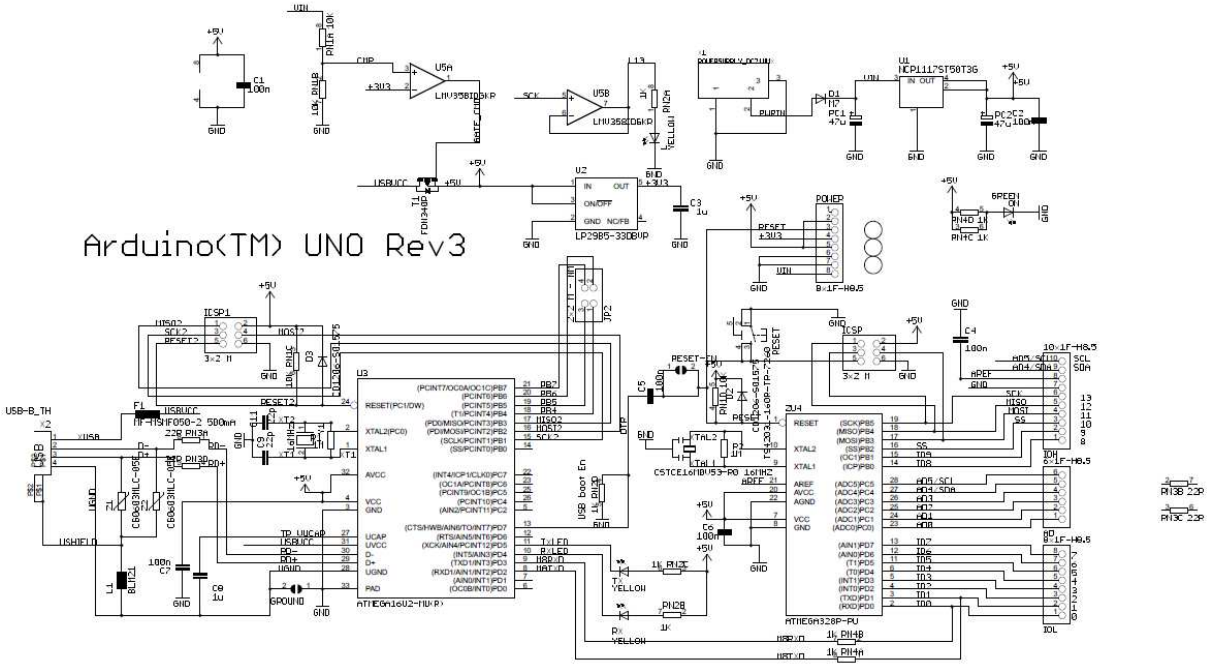
Possui também um conjunto de seis pinos para conexões que utilizam o protocolo ICSP.

E caso seja necessário alimentar algum outro circuito através da placa Arduino, há um pino de saída de 3,3V e outro de 5V, além de um pino de *ground* (0V).

2.3.2 Esquema Elétrico do Arduino UNO

Como trata-se de plataforma *open source*, o esquema elétrico das placas Arduino são divulgadas em seu *site*, sendo permitida a sua reprodução. A seguir segue figura do esquema elétrico do Arduino UNO.

Figura 2.3-2 *- Esquema elétrico do Arduino UNO.*



Fonte: http://www.arduino.cc

Para melhor entendimento, o esquema elétrico será dividido em blocos, tendo em vista suas funcionalidades. O primeiro bloco a ser visto será o responsável pela alimentação do Arduino, após o responsável pela comunicação via USB entre o Arduino e um computador, e por último o bloco responsável pelo processamento de toda a lógica de programação implementada e gravada no Arduino.

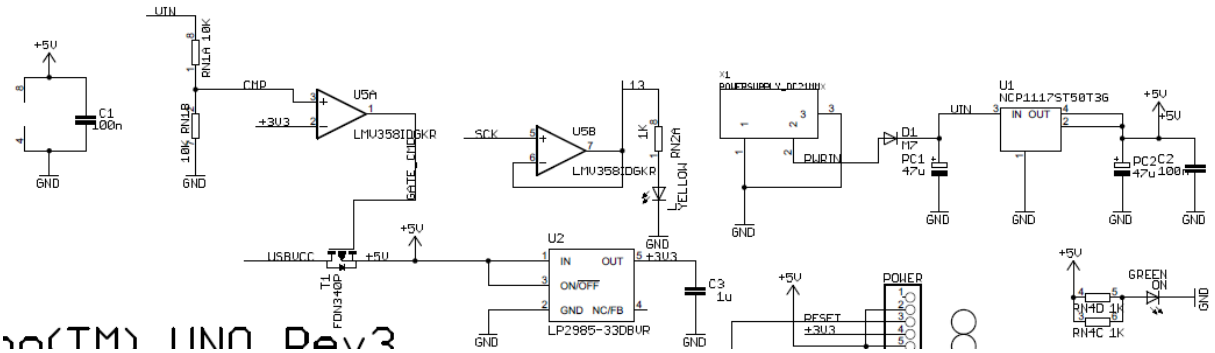
2.3.2.1 Bloco funcional: alimentação

Bloco que representa a alimentação externa do Arduino através de conector Jack (componente PWRIN), o qual por padrão possui um nível de tensão de 12V. Porém, o Arduino trabalha com nível de tensão menor, portanto há um regulador de tensão de 5V. Há outro regulador para os casos em que se necessite trabalhar com nível de 3,3V.

Como é possível alimentar também o Arduino através do uso de conector USB (componente USBVCC), haveria conflito de fontes caso houvesse a ligação através dos dois conectores. Para evitar, há um amplificador operacional (componente U5A) configurado no modo de comparação. Na sua porta inversora há a recepção de um sinal de 3,3V, e na porta não-inversora há o recebimento de metade da tensão de V_{in} (equivale à tensão do conector Jack, normalmente 12 V, subtraído da queda de tensão no diodo D_1). Dessa forma, o amplificador

compara os dois sinais, e caso a tensão na porta não-inversora seja maior que a tensão na porta inversora, a corrente drenada será do Jack. Caso contrário, drenará corrente da USB.

***Figura 2.3-3** - Bloco funcional de alimentação.*



Fonte: http://www.arduino.cc

Esse mesmo amplificador (U5B) também é o responsável por acender o *LED* (YELLOW) utilizado pelo pino 13 do Arduino. Nessa situação, o amplificador verifica se há algum sinal no barramento SCK. Se houver, o *LED* é aceso.

E por último há o *LED* (GREEN) indicador de ON, o qual recebe um sinal de 5 V. Um divisor de corrente é implementado através dos resistores RN4C e RN4D.

2.3.2.2 Bloco funcional: processador USB

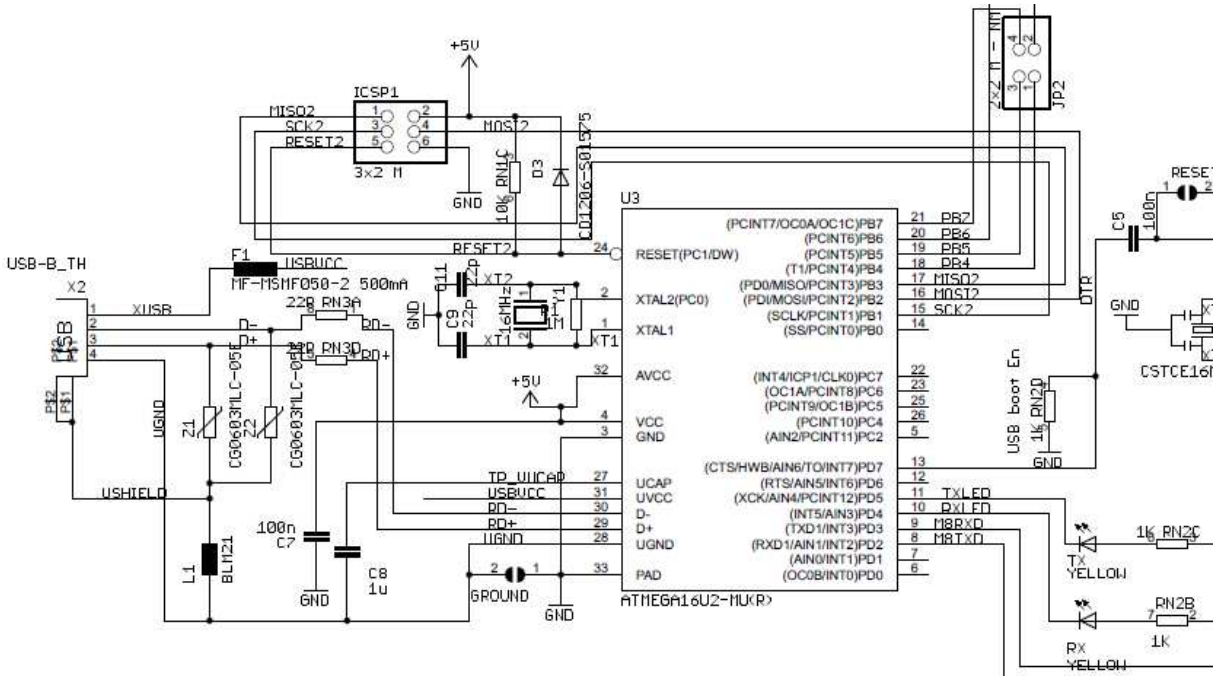
Bloco que representa a conexão com um computador através do conector USB, para tráfego de dados. O conector USB também é utilizado como fonte de alimentação quando não há conexão via conector Jack.

Este bloco possui como principal componente o microcontrolador ATMEGA16U2, da Atmel®, o qual "é necessário pois o processador principal do Arduino não suporta conexão direta com uma porta USB. Dessa forma, o processador converte os dados da USB do computador para um sinal serial (UART) que possa ser lido pelo processador principal" (CIRCUITAR).

Os pulsos de *clock* do microcontrolador são gerados através do oscilador (Y1) de cristal numa frequência de 16 MHz. Para o pleno funcionamento do oscilador, há o uso de dois capacitores de 22 picofarads (C9 e C11) e de um resistor de 1 MΩ (R1).

Este bloco possui também um conjunto de seis pinos os quais permitem a gravação do *bootloader* do microcontrolador no momento da fabricação da placa.

***Figura 2.3-4** - Bloco funcional do processador USB.*



Fonte: http://www.arduino.cc

Para proteção da porta USB por motivo de sobrecarga de corrente, há o fusível F_1, no qual se utiliza um termistor. Dessa forma, quando drenada uma corrente superior a 500 mA, o termistor eleva o valor da resistência, aumentando dessa forma a oposição à passagem de corrente.

Para proteger os pinos de dados (D- e D+) da USB, há dois varistores (Z_1 e Z_2), os quais têm redução na sua resistência quando submetidos a um excesso de tensão.

Há também um ferrite (L_1) para filtrar sinais de alta frequência.

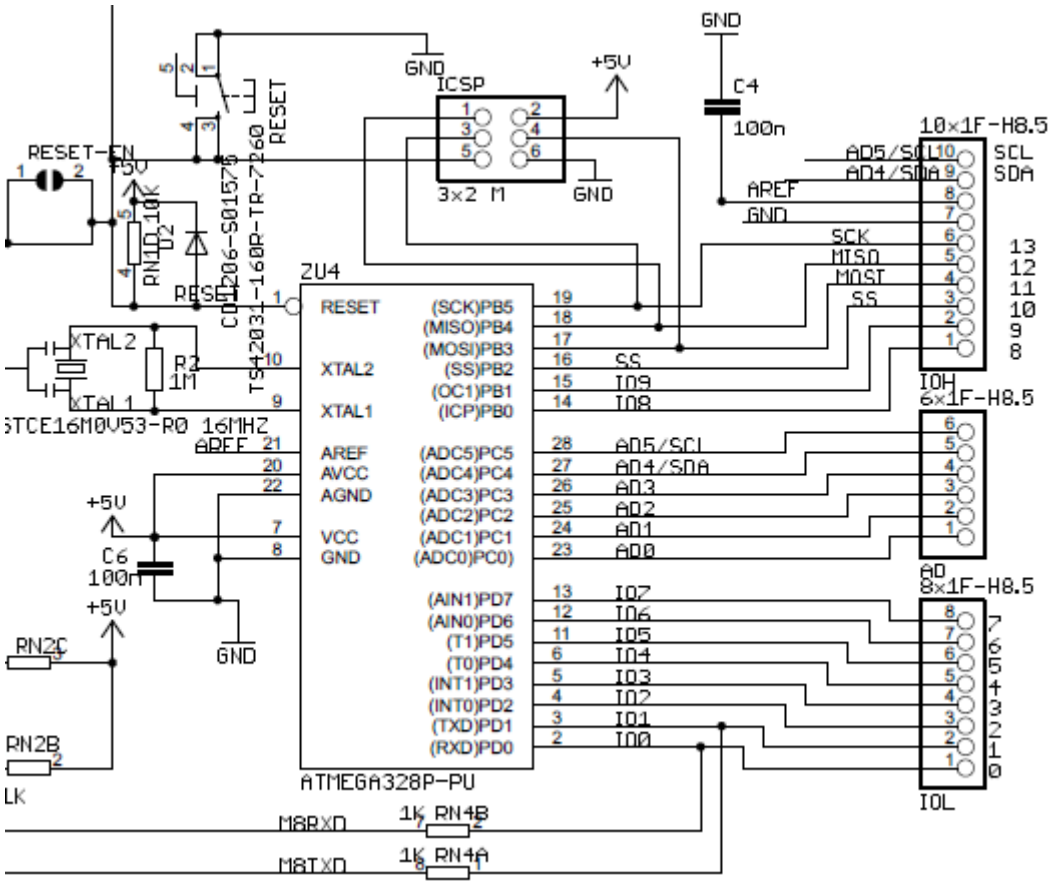
2.3.2.3 Bloco funcional: processador principal

Neste bloco há o principal microcontrolador utilizado pelo Arduino, o ATMEGA328P, da Atmel.

Esse microcontrolador é responsável pela execução do código-fonte gravado, além de estar interligado com praticamente todos os pinos da placa.

A conexão com a porta USB ocorre através dos pinos de TXD (transmissão/escrita de dados) e RXD (recepção/leitura de dados), sendo intermediado pelo microcontrolador ATMEGA16U2.

Figura 2.3-5 *- Bloco funcional do processador principal.*



Fonte: http://www.arduino.cc

Da mesma forma que o ATMEGA16U2, possui também um oscilador de 16 MHz, porém de material cerâmico.

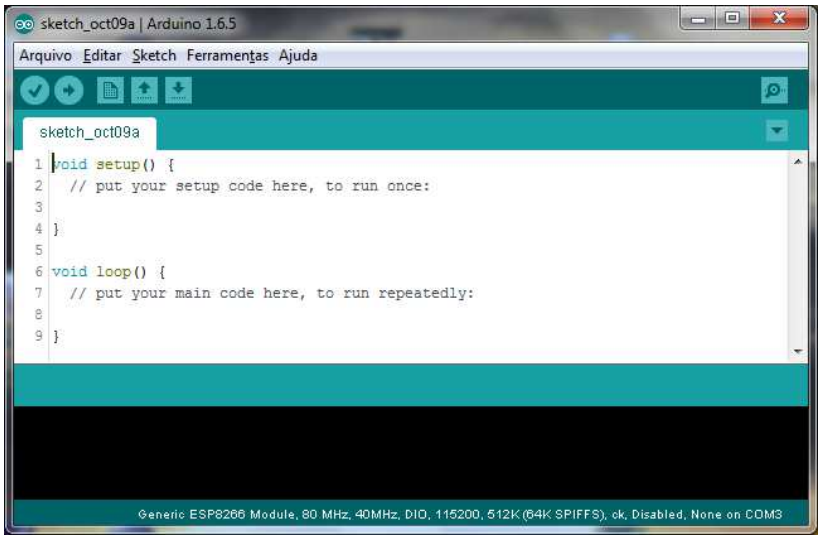
2.3.3 IDE de Desenvolvimento

A plataforma Arduino possui uma IDE (*Integrated Development Environment*) para desenvolvimento do código-fonte a ser gravado e executado na placa Arduino, a qual permite a escrita em linguagem C/C++.

Basicamente, todo programa a ser executado no Arduino deve possuir duas funções, que são: *setup* e *loop*.

A função *setup* é o ponto de partida de todo programa executado no Arduino. Desse modo, normalmente possui o código-fonte necessário para configurar e permitir a execução do programa, como configurar pinos e velocidades de execução da serial, por exemplo.

Figura 2.3-6 - *IDE para desenvolvimento do código-fonte.*



Fonte: Elaborado pelo autor.

Após a execução da função *setup*, o microcontrolador executa a função *loop*. Como o próprio nome informa, essa função executa em *loop*, ou seja, quando se atinge a sua última linha de código o microcontrolador a executa novamente, reiniciando dessa forma a sua execução. Para que a função finalize, é necessário que o Arduino seja reinicializado ou desligado. Em caso de *reset*, o microcontrolador encerrará a função *loop* e executará a função *setup* novamente.

3 DESENVOLVIMENTO DO TRABALHO PROPOSTO

Este capítulo destina-se ao desenvolvimento do trabalho proposto, o qual está dividido em diversas etapas, conforme exposto anteriormente na ***Figura 1.2-1***.

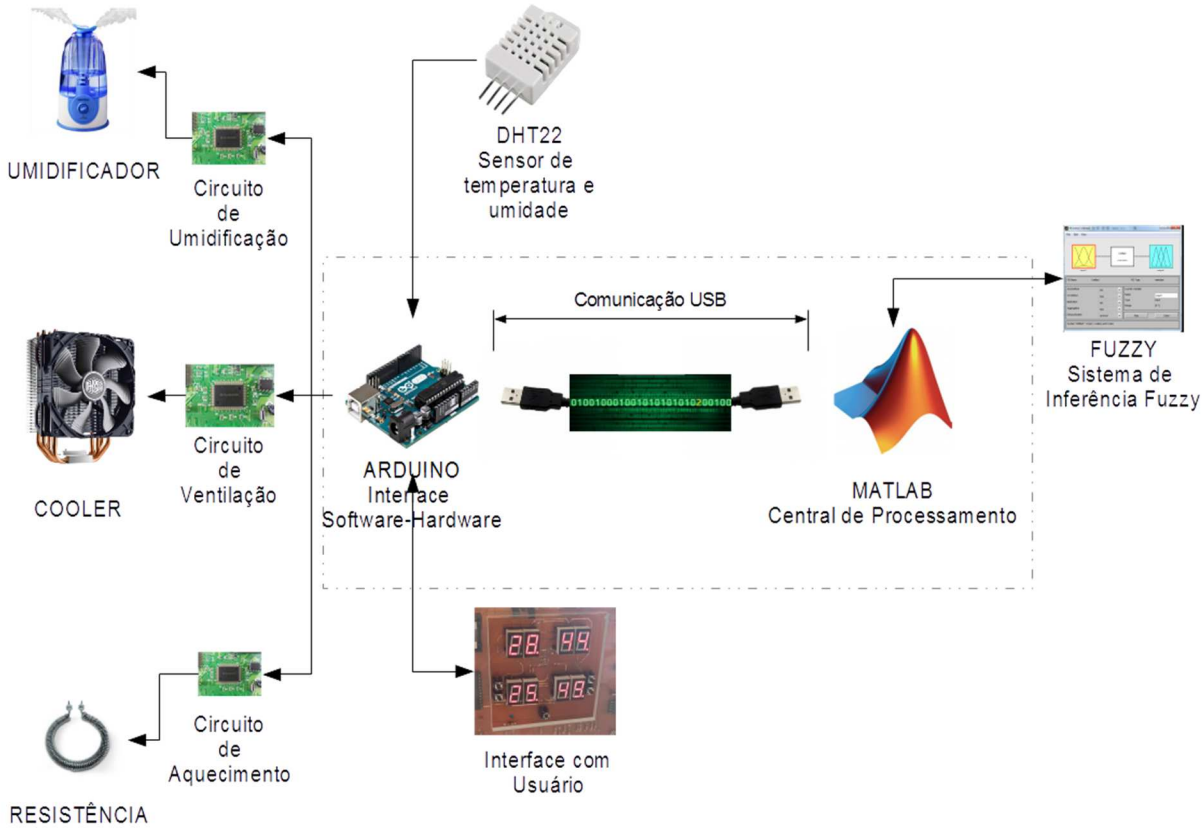
Inicialmente o projeto se detém na realização de pesquisa bibliográfica (etapa 1) sobre lógica *fuzzy*, sistemas de controle que a utilizam e sistemas de controle de termohigrometria. Obtido o conhecimento necessário, permite-se desenvolver o conjunto de regras *fuzzy* (etapa 2) que serão responsáveis por controlar o sistema. Após, realiza-se a pesquisa (etapa 3) e o desenvolvimento de circuito responsável por conter os sensores e comunicar-se com o MATLAB (etapa 4). Nova pesquisa é necessária para definir os atuadores (etapa 5) a serem utilizados e o desenvolvimento dos circuitos que os conterão (etapa 6) e que serão responsáveis por permitir a comunicação entre estes e o circuito da maquete. Desenvolve-se então a Central de Processamento (etapa 7) na ferramenta MATLAB, a qual obtém os valores lidos pelos sensores através de comunicação *serial* com o circuito da interface Software-Hardware, realiza a fuzzificação dos mesmos e os repassa ao Sistema de Inferência *Fuzzy*, o qual por sua vez retorna os valores de potência a serem aplicados nos atuadores. Para tanto, necessita-se desenvolver o circuito da Interface Software-Hardware (etapa 8), o qual será responsável por interligar os sensores e atuadores à Central de Processamento do MATLAB. E por último, são realizados testes (etapa 9) que validem o sistema de controle.

3.1 Componentes e funcionalidades do projeto

Nesta seção serão descritas todas as partes integrantes do sistema desenvolvido e suas funcionalidades, conforme apresentado na **Figura 3.1-1.**

Na figura, as setas indicam o sentido da comunicação entre as partes, nomes em letras maiúsculas o tipo de componente utilizado e em letras minúsculas a funcionalidade. Para cada funcionalidade, haverá um detalhamento de como foi desenvolvido e como interage com as demais.

***Figura 3.1-1** - Componentes e funcionalidades do sistema de controle.*



Fonte: Elaborado pelo autor.

3.1.1 Central de Processamento

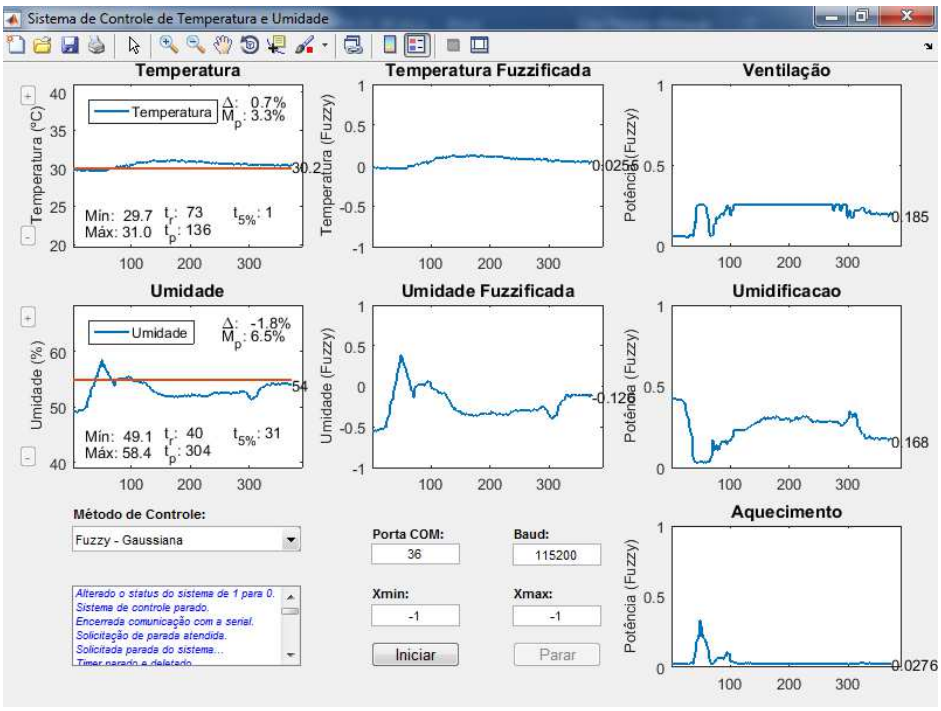
A Central de Processamento executa um programa no MATLAB (vide código-fonte completo no **Anexo 1.1**), o qual obtém os valores necessários, os repassa ao sistema de inferência *fuzzy* e informa à Interface Software-Hardware os valores de potência a serem aplicados nos atuadores. Esse programa possui um *timer* (declarado na função `startSistemaControle`) que periodicamente "acorda" e solicita à Interface Software-Hardware os dados do sistema, até que o usuário pressione o botão *STOP* na Interface com Usuário existente na maquete ou pela Interface com Operador no MATLAB.

A **Figura 3.1-2** apresenta a Interface com Operador do MATLAB com o programa da Central de Processamento em execução. Ao iniciar o sistema de controle apresenta-se na console do MATLAB o método de inferência utilizado (mamdani), quantidade de entradas (2), nomes das entradas (temperatura e umidade), quantidade de saídas (3), nomes das saídas (ventilador, umidificador e resistência), quantidade de regras *fuzzy* (49), métodos dos

operadores AND, OR, implicação e agregação utilizados pelo módulo de inferência (min, max, min e max, respectivamente) e o método de defuzzificação utilizado (centróide).

Obtido os dados do sistema *fuzzy*, o programa realiza a limpeza de variáveis utilizadas, solicita a conexão via *serial* na porta e velocidade especificadas pela Interface com Operador, cria o *timer* e o inicializa, passando então a plotar os gráficos a serem apresentados ao operador.

Figura 3.1-2 – *Interface do MATLAB com o operador.*



Fonte: Elaborado pelo autor.

Após o operador pressionar o botão `STOP`, o programa realiza a parada do *timer* e a sua deleção (função `stopSistemaControle`), solicita a parada de todo o sistema de controle, solicita que a conexão com a *serial* seja fechada, liberando desse modo o recurso.

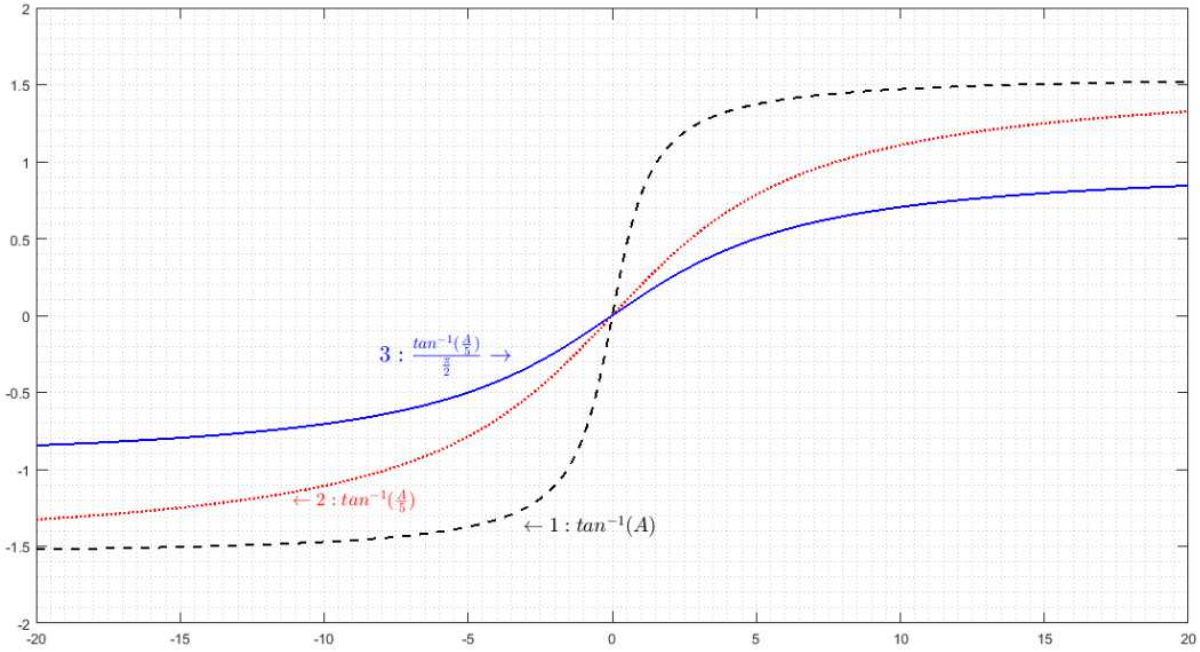
Toda vez que o *timer* "acorda" (função `TimerCallback`) o programa solicita à Interface Software-Hardware os valores de *set point* da temperatura e umidade, os valores da temperatura e umidade e se o operador pressionou ou não o botão STOP (função `realizarLeituras`).

$$fuzz = \frac{\tan^{-1}\left(\frac{(valor\ -\ setPoint)}{5}\right)}{\frac{\pi}{2}} \qquad (3.1\text{-}1)$$

Porém, os valores de temperatura e umidade precisam ser convertidos para que sejam úteis ao sistema de inferência *fuzzy*. Como o sistema *fuzzy* foi modelado para entender valores de -1 a +1, aplica-se a **Equação (3.1-1)**, de tal modo que se o valor da temperatura for igual ao respectivo *set point*, a função retornará zero. A **Figura 3.1-3** demonstra passo-a-passo o funcionamento da **(3.1-1**, onde quanto menor for o valor da temperatura em relação ao *set point*, mais próximo de -1 retornará a função (função `fuzzificacao`). De forma simétrica, quanto maior for a temperatura em relação ao *set point*, mais próximo de +1 retornará a função. O mesmo cálculo é realizado para a umidade.

Realizada as conversões da temperatura e umidade, aciona-se o sistema *fuzzy* informando como entrada os respectivos valores (função `calcularPotencias`). Como resposta, retornam-se os valores (de 0 a 1) de potência a serem aplicados nos atuadores.

Figura 3.1-3 - *Gráfico demonstrando cada etapa da equação de conversão dos valores.*



Fonte: Elaborado pelo autor.

Após obtem-se os valores de potência anteriores, pois somente haverá a transmissão à Interface Software-Hardware caso exista mudança de valores, minimizando assim o tráfego no barramento *serial*. Armazenam-se então os novos valores de potência. Realizam-se algumas verificações e plotam-se os gráficos.

Por último há o envio das potências, desde que tenha ocorrido alteração.

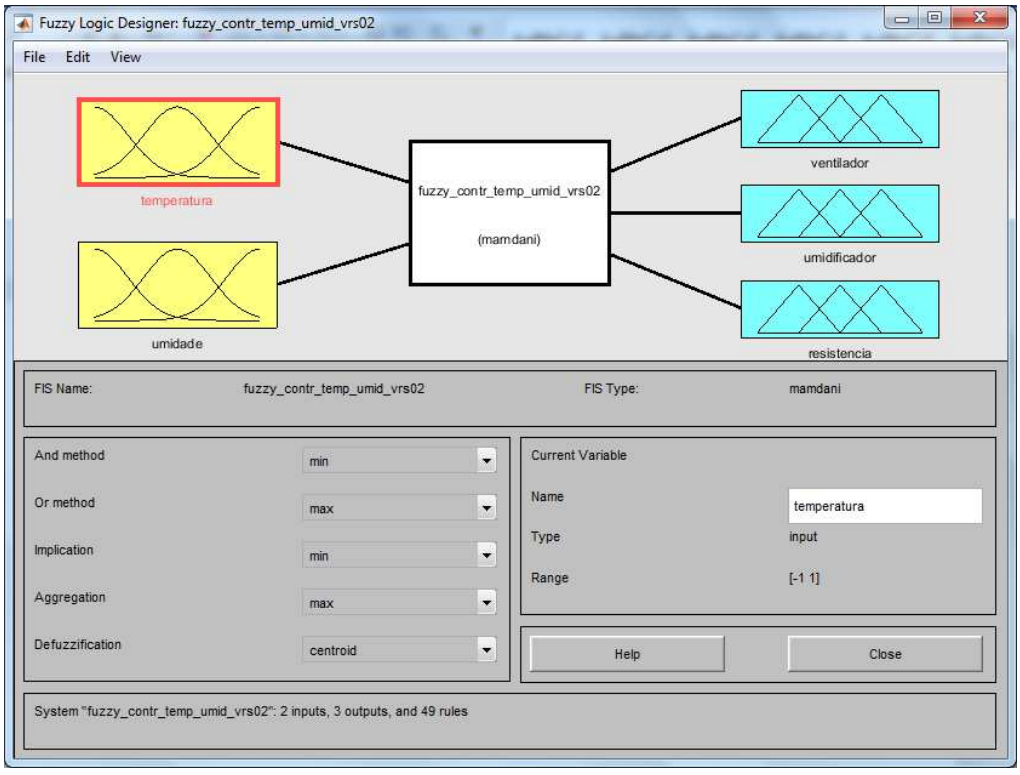
Na **Figura 3.1-2** vê-se os gráficos apresentados ao operador referentes à temperatura e umidade lidas (primeira coluna), respectivos valores convertidos (segunda coluna) e as

potências calculadas (terceira coluna) pelo sistema de inferência *fuzzy*. Além disso, é possível especificar um outro tipo de método de controle a ser aplicado (foram implementados os métodos **Fuzzy – Gaussiana**, **Fuzzy – Trapezoidal** e **On/Off**), informar o número e velocidade da porta COM de comunicação *serial*, valores inicial e final do eixo **x** dos gráficos, além de ser apresentado um *log* da execução do sistema, e um botão para iniciar ou pausar a aplicação e outro para parada total do sistema.

3.1.2 Sistema de inferência fuzzy

A modelagem do sistema *fuzzy* foi desenvolvida na *toolbox Fuzzy* do MATLAB. As primeiras informações prestadas foram as variáveis de entrada (temperatura e umidade), de saída (ventilador, umidificador e resistência), o método de inferência utilizado (mamdani) e o método de defuzzificação (centróide), conforme **Figura 3.1-4**.

***Figura 3.1-4** - Informações básicas do modelo fuzzy.*

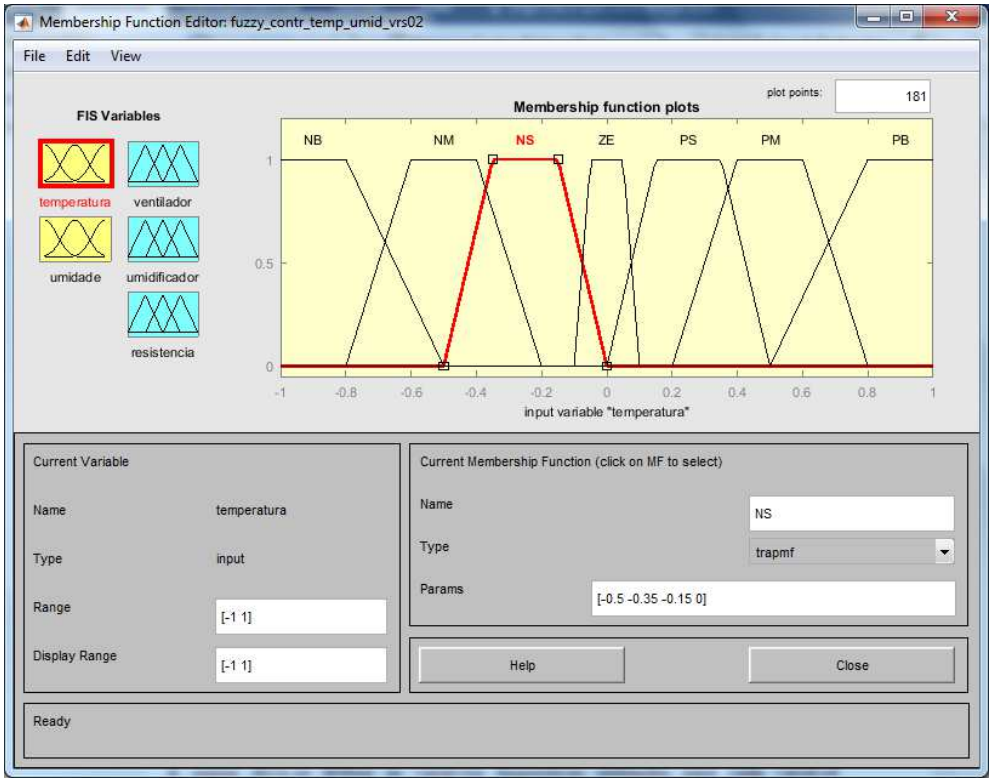


Fonte: Elaborado pelo autor.

A seguir definiu-se os termos linguísticos utilizados para cada variável. Normalmente utilizam-se mnemônicos para se nomear as variáveis.

Para as variáveis de entrada “temperatura” e “umidade” foram definidos sete termos, conforme **Figura 3.1-5**: NB (*negative big*), NM (*negative medium*), NS (*negative small*), ZE (*zero*), PS (*positive small*), PM (*positive medium*) e PB (*positive big*).

Figura 3.1-5 *- Termos linguísticos e funções de pertinência da variável “temperatura”.*



Fonte: Elaborado pelo autor.

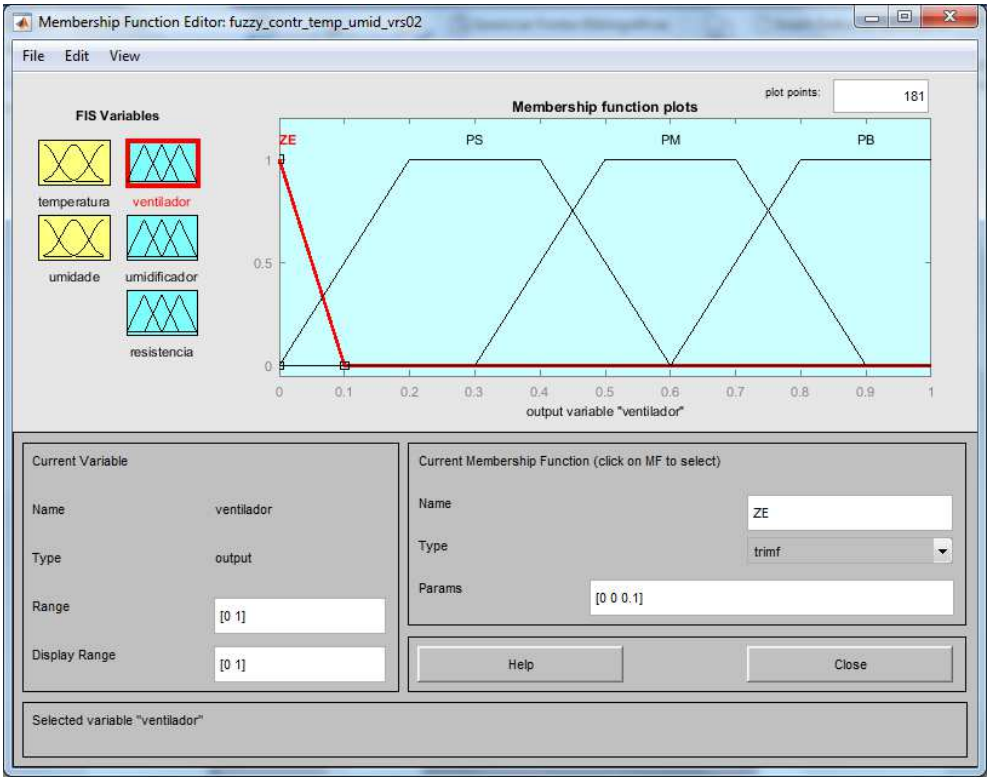
Para cada termo criou-se uma função de pertinência trapezoidal (ou Gaussiana no caso da implementação do método **Fuzzy - Gaussiana**), sendo que quanto mais próximo de zero, mais estreita a base, e quanto mais próximo dos extremos (-1 e +1), mais larga.

O mesmo procedimento foi realizado para as variáveis de saída, porém com uma quantidade menor de termos, conforme **Figura 3.1-6**: ZE (*zero*), PS (*positive small*), PM (*positive medium*) e PB (*positive big*), pois as potências não possuem valores negativos, apenas positivos. Utilizou-se funções de pertinência trapezoidais (ou gaussiana no caso da implementação do método **Fuzzy - Gaussiana**), exceto para o termo ZE, onde se utilizou uma função triangular.

Como pode-se verificar, todas as variáveis *fuzzy*, sejam as de entrada, sejam as de saída, possuem valor no eixo da ordenada compreendido de ZERO a UM. Ou seja, um valor ZERO significa que não há pertinência em relação ao termo linguístico que estiver sendo avaliado. No

extremo oposto, o valor UM significa que há pertinência total. Esses dois casos extremos referem-se exatamente à lógica clássica, também conhecida por lógica booleana.

Figura 3.1-6 *- Termos linguísticos e funções de pertinência da variável "ventilador".*



Fonte: Elaborado pelo autor.

Para os valores intermediários, significa que há um certo grau de pertencimento.

Definidos os métodos de inferência e de defuzzificação, as variáveis a serem utilizadas, os termos linguísticos e suas respectivas funções de pertinência, criou-se então o conjunto de regras. Como foram utilizados sete termos para cada variável de entrada, houve então a necessidade de serem elaboradas 49 regras (7 vezes 7), as quais contemplam todas as combinações possíveis.

Essas regras são criadas a partir do conhecimento que se têm sobre a planta a ser controlada, quanto à relação e influência entre as variáveis de entrada, análise dos gráficos tridimensionais gerados pelo sistema *fuzzy* e ajustes finos realizados a partir de testes.

Na **Tabela 3.1-1** é possível verificar o conjunto de regras que foram definidas para o método **Fuzzy - Trapezoidal**, as quais no método de inferência do tipo “mamdani” seguem um padrão de condição (*if*) e causa (*then*). Para obtenção do arquivo gerado pela toolbox, vide

Anexo 2.1. No **Anexo 2.2** consta a especificação do método **Fuzzy – Gaussiana**, o qual é utilizado quando se requer um comportamento mais suave e preciso do sistema.

Na primeira coluna da tabela estão listados todos os termos linguísticos da variável de entrada “temperatura”, e na primeira linha os termos linguísticos da variável “umidade”. Nas demais células consta os valores *fuzzy* a serem aplicados nos atuadores, que são respectivamente “ventilador”, “umidificador” e “resistência”. Por exemplo, os valores de potência que devem ser atribuídos a cada atuador para a situação de haver uma “temperatura” *NB* e “umidade” *PM* é de *PM*, *ZE* e *PB*, respectivamente para o ventilador, umidificador e resistência.

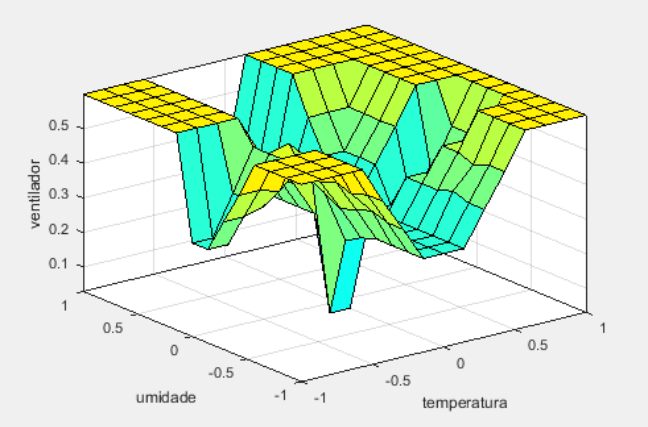
Verifique pela tabela que a única situação em que se aplicará potência *ZE* a todos os atuadores é quando as variáveis de entrada atingem o *set point*, ou seja, valor *ZE*.

***Tabela 3.1-1** - Conjunto de regras.*

<table>
<tr><th>Umidade
Temperatura</th><th>NB</th><th>NM</th><th>NS</th><th>ZE</th><th>PS</th><th>PM</th><th>PB</th></tr>
<tr><td>NB</td><td>PM PB PB</td><td>PM PM PB</td><td colspan="2">PS PS PB</td><td>PM PS PB</td><td colspan="2">PM ZE PB</td></tr>
<tr><td>NM</td><td>PM PB PM</td><td>PM PM PM</td><td colspan="2">PM PS PM</td><td colspan="3">PM ZE PM</td></tr>
<tr><td>NS</td><td>PS PB PS</td><td>PS PM PS</td><td>ZE PS PS</td><td>PS PS PS</td><td colspan="3">PS ZE OS</td></tr>
<tr><td>ZE</td><td rowspan="2">PS PB ZE</td><td rowspan="2">PS PM ZE</td><td>ZE PS ZE</td><td>ZE ZE ZE</td><td colspan="3">PS ZE ZE</td></tr>
<tr><td>PS</td><td rowspan="2">PS PS ZE</td><td colspan="2">PS ZE ZE</td><td colspan="2">PM ZE ZE</td></tr>
<tr><td>PM</td><td rowspan="2">PM PB ZE</td><td rowspan="2">PM PM ZE</td><td rowspan="2">PM PS ZE</td><td colspan="3">PM ZE ZE</td></tr>
<tr><td>PB</td><td>PM PS ZE</td><td colspan="3">PM ZE ZE</td></tr>
</table>

Uma forma de se analisar o conjunto de regras é através da análise de gráficos tridimensionais, os quais são gerados pela *toolbox*. Na **Figura 3.1-7** consta o gráfico gerado para o “ventilador”, com base nas duas variáveis de entrada.

***Figura 3.1-7** - Gráfico 3D para o "ventilador" a partir do método **Fuzzy - Trapezoidal**.*

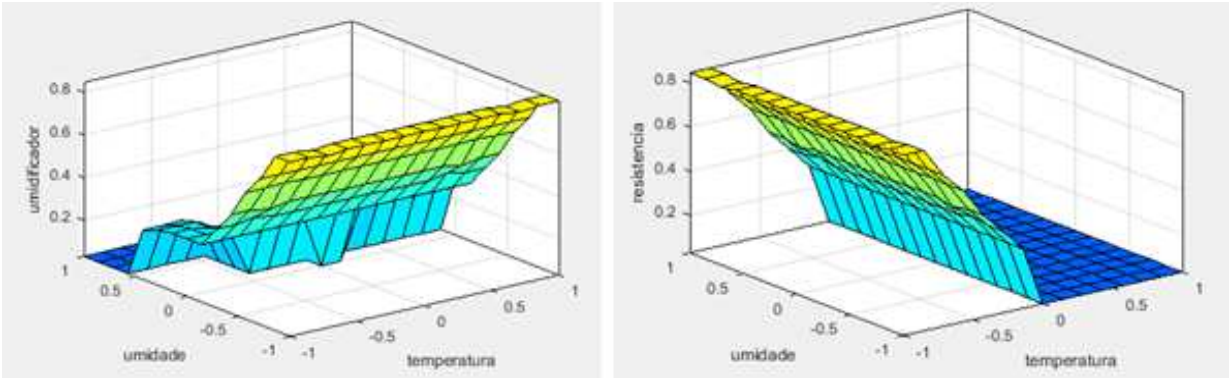


Fonte: Elaborado pelo autor.

Como se pode verificar pelo gráfico, quando o valor da temperatura e umidade estão próximos de ZERO, o valor da potência aplicada pelo sistema de ventilação deve ser ZERO. Porém quando a temperatura ou a umidade se aproximam de seus extremos (-1 ou +1), a potência tende ao máximo (valor relativo igual a UM).

Na **Figura 3.1-8** constam os gráficos tridimensionais gerados para a umidificação e aquecimento. Ao contrário do sistema de ventilação, esses atuadores são acionados somente quando há valores baixos, ou seja, a potência do umidificador tenderá ao máximo quando o valor relativo da umidade estiver próximo de -1, e a resistência tenderá ao máximo quando a temperatura relativa estiver próxima de -1.

***Figura 3.1-8** - Gráficos para o "umidificador" (a) e "aquecimento" (b), a partir do método **Fuzzy - Trapezoidal**.*



Fonte: Elaborado pelo autor.

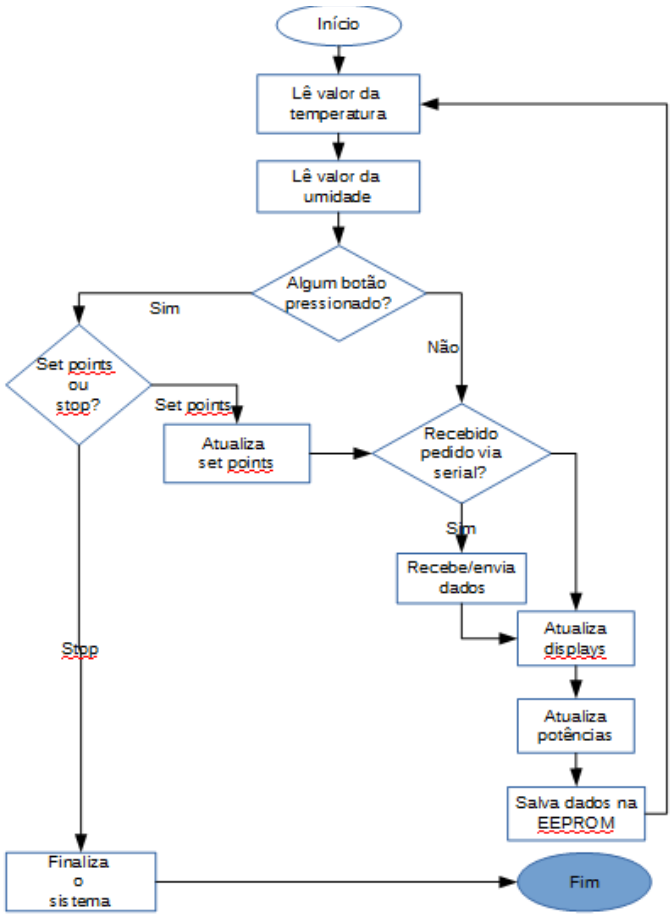
(a) **(b)**

3.1.3 Interface S*oftware-Hardware*

A interface entre o *software* MATLAB, *hardwares* dos circuitos atuadores e sensores e com o usuário se dá pelo desenvolvimento de solução através da plataforma de prototipagem Arduino.

No Arduino foi desenvolvido *software* que realiza a leitura da temperatura e umidade através do componente de termohigrometria DHT22 (também é possível o uso do DHT11, o qual possui menor custo, porém menor precisão), verifica se algum botão foi pressionado pelo operador pela interface da maquete, atualiza os dados apresentados nos *displays*, atualiza as potências a serem aplicadas nos atuadores e atualiza os dados armazenados na memória não volátil do Arduino. Essa descrição está resumida no fluxograma apresentado na **Figura 3.1-9**.

***Figura 3.1-9** - Fluxograma da execução do programa de interface entre hardware e software.*



Fonte: Elaborado pelo autor.

No **Anexo 3.1** é possível verificar todo o código fonte do programa armazenado no Arduino para execução. No **Anexo 3.2** constam as constantes utilizadas pelo programa. Abaixo segue descrição dos principais pontos do programa do **Anexo 3.1**.

As primeiras linhas do código declaram as bibliotecas utilizadas (`Arduino.h`, `string.h`, `EEPROM.h`, `DHT.h`, `LedControl.h` e `EepromNumber.h`), códigos externos utilizados (`Constantes.c`), objetos necessários (`lc` e `dht`) e diversas variáveis globais necessárias.

No método `setup` do Arduino, o qual é executado toda vez que o microcontrolador é ligado ou reinicializado, são inicializados os displays de interação com o operador (`initDisplays`), declarados os modos de uso dos pinos (`initPinos`), inicializada a serial (`initSerial`), o sensor de termohigrometria (`initSensorTermohigrometria`) e os *set points* (`initSetPoints`).

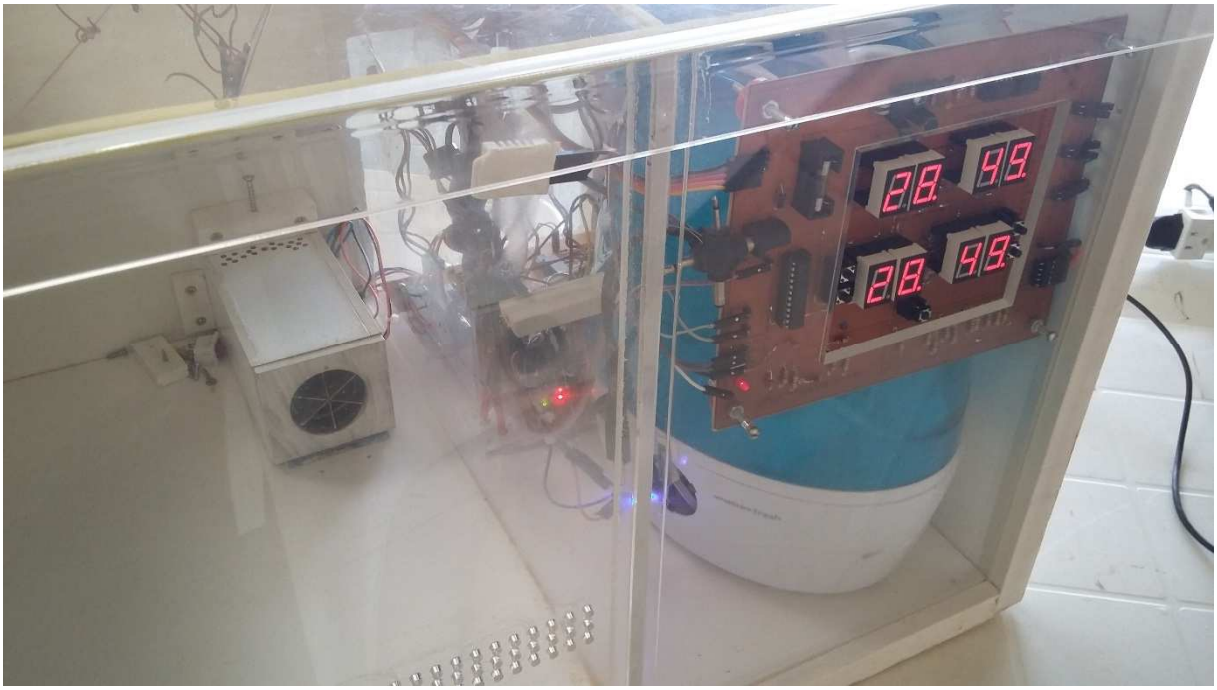
Após o `setup`, o Arduino executa infinitamente e ciclicamente a função `loop`. Nessa função realiza-se primeiramente a leitura dos sensores de temperatura e umidade (`dht.readTemperature` e `dht.readHumidity`), verifica se algum botão foi pressionado (`checkBotaoPressionado`), se há alguma solicitação por parte do MATLAB de transmissão de dados via serial (`checkSerial`), atualiza os displays (`updateDisplays` e `updateDisplaysSetPoint`), atualiza os valores a serem aplicados nos atuadores conforme definidos pelo Sistema de Inferência *Fuzzy* e atualiza na memória não-volátil os valores de set point (`updateEeprom`) para que possam ser obtidos na próxima vez que o Arduino for reinicializado.

3.1.4 Circuito da interface com o operador

Para que o operador possa interagir com o sistema foi criado um circuito que apresenta os valores atuais da temperatura e umidade dentro do ambiente a ser controlado, os valores de *set point* da temperatura e umidade desejados e botões que permitam alterar esses valores de *set point*. Esse circuito foi fixado na parte frontal da maquete, conforme **Figura 3.1-10**.

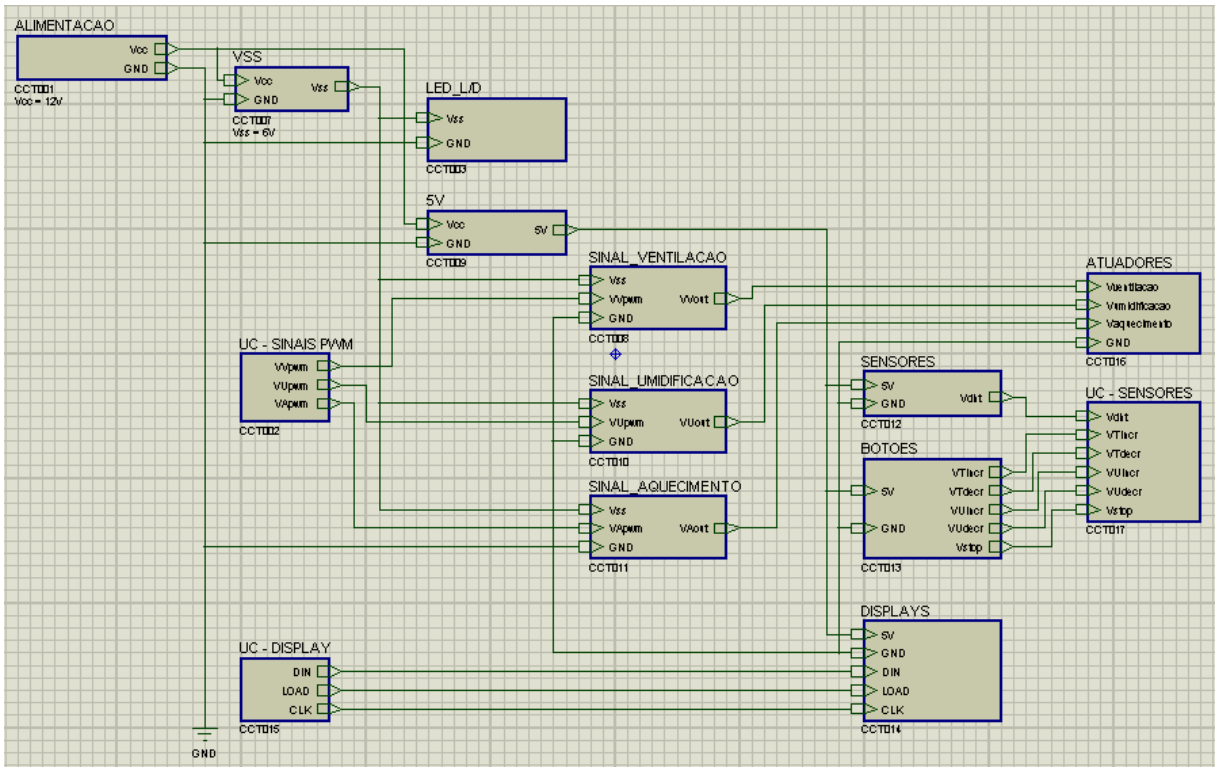
Fonte: Elaborado pelo autor.

Figura 3.1-10 *- Parte frontal da maquete mostrando o circuito de interface com o operador.*



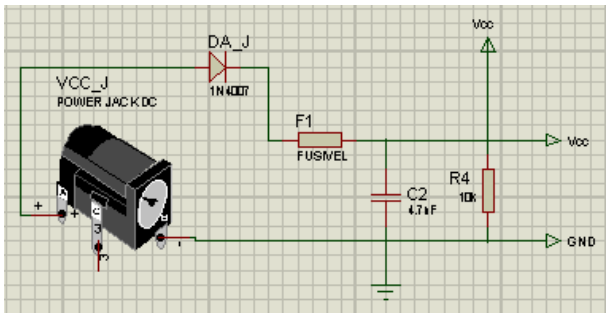
Na **Figura 3.1-11** apresenta-se o diagrama de blocos demonstrando o funcionamento do circuito. Cada retângulo do diagrama representa um subcircuito do circuito principal, conforme descrição a seguir.

***Figura 3.1-11** - Diagrama de blocos do circuito de interação com o operador.*



Fonte: Elaborado pelo autor.

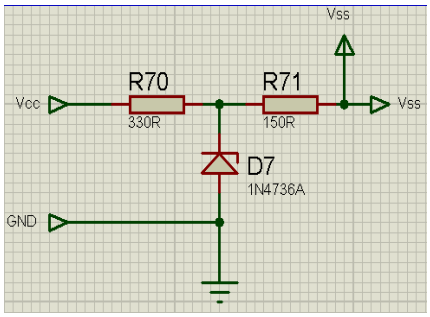
***Figura 3.1-12** - Subcircuito ALIMENTACAO.*



Fonte: Elaborado pelo autor.

O subcircuito `ALIMENTACAO` é responsável pelos conectores de alimentação da placa, o qual deve ser conectado a uma fonte de 12V.

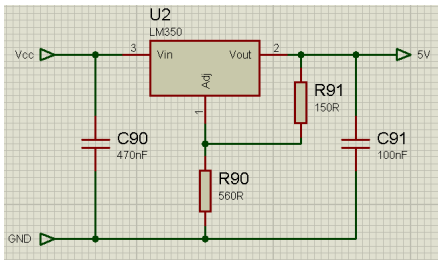
***Figura 3.1-13** - Subcircuito VSS.*



Fonte: Elaborado pelo autor.

O subcircuito `VSS` reduz o nível de tensão para 6V através do diodo zêner `D7`, o qual é utilizado pelos amplificadores operacionais.

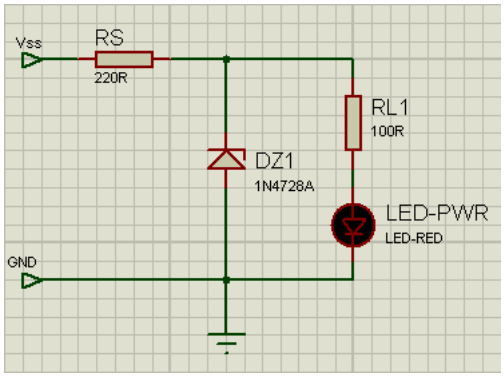
***Figura 3.1-14** - Subcircuito 5V.*



Fonte: Elaborado pelo autor.

O subcircuito 5V reduz o nível de tensão para 5V através do componente 7805 denominado `U2`, para utilização pelo CI MAX7219.

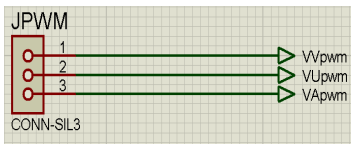
***Figura 3.1-15** - Subcircuito LED_L/D.*



Fonte: Elaborado pelo autor.

O subcircuito `LED_L/D` liga o led `LED-PWR` identificando que o circuito foi conectado a uma fonte de alimentação. Para reduzir a tensão para um nível adequado ao led foi utilizado o diodo zêner de 3,3V 1N4728A, denominado por `DZ1`.

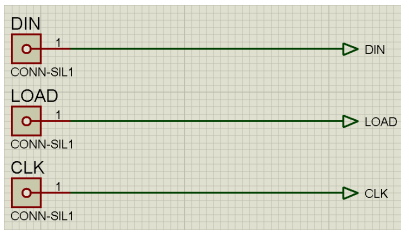
***Figura 3.1-16** - Subcircuito UC - SINAIS PWM.*



Fonte: Elaborado pelo autor.

O subcircuito `UC - SINAIS PWM` contém três conectores que devem ser interligados ao Arduino para obtenção do sinal dos atuadores em formato PWM (*pulse width modulation*).

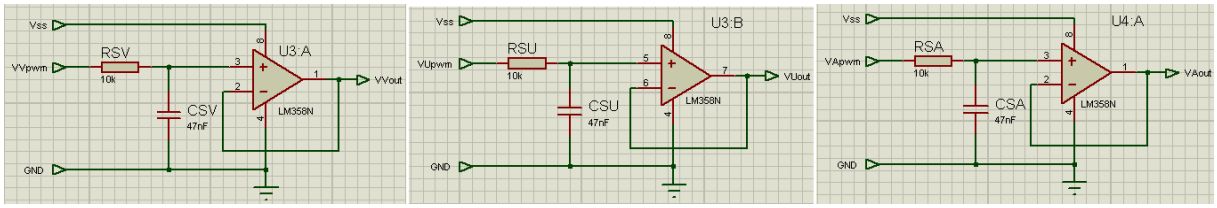
***Figura 3.1-17** - Sub-circuito UC-DISPLAY.*



Fonte: Elaborado pelo autor.

O subcircuito `UC – DISPLAY` contém três conectores que devem ser interligados ao Arduino para obtenção dos sinais referentes aos displays de 7 segmentos que apresentam a temperatura e umidade.

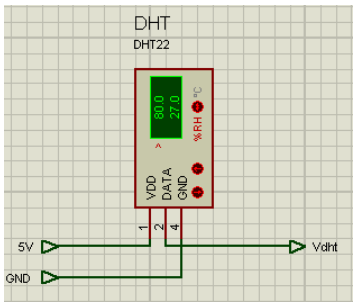
***Figura 3.1-18** - Subcircuitos SINAL_VENTILACAO, SINAL_UMIDIFICACAO e SINAL_AQUECIMENTO.*



Fonte: Elaborado pelo autor.

Cada subcircuito da figura acima converte um sinal PWM enviado pelo Arduino para um sinal de 0 a 5V proporcional à largura do pulso, referente a cada um dos atuadores. Para tanto, utilizam-se dois amplificadores operacionais (`U3` e `U4`) configurados como filtro.

***Figura 3.1-19** - Subcircuito SENSORES.*

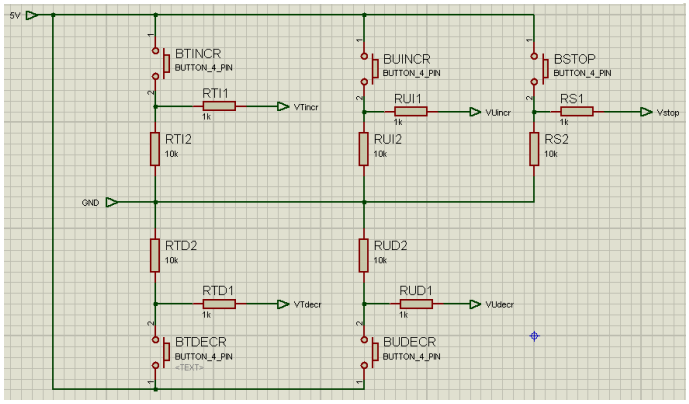


Fonte: Elaborado pelo autor.

Subcircuito contém um conector de quatro pinos que permitir ligar os fios provindos do sensor que fica dentro do ambiente a ser controlado.

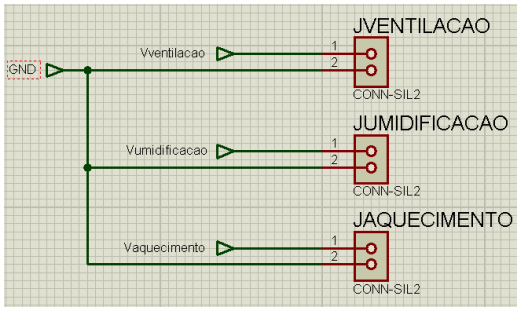
Subcircuito contendo os botões que permitem ao usuário alterar os valores de set point de temperatura e umidade, além de permitir que o sistema de controle seja parado. Os resistores em que os nomes finalizem com o número “1” foram colocados para que limitem a corrente, evitando assim curtos-circuitos em caso de montagem incorreta da placa.

***Figura 3.1-20** - Subcircuito BOTOES.*



Fonte: Elaborado pelo autor.

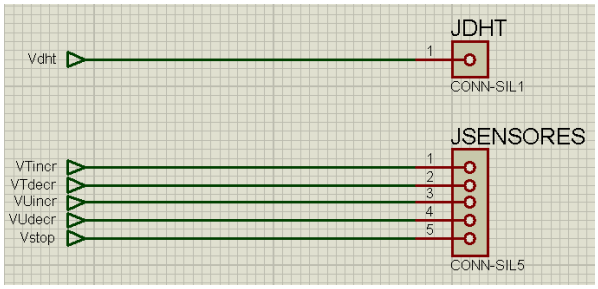
***Figura 3.1-21** - Subcircuito ATUADORES.*



Fonte: Elaborado pelo autor.

Contém conectores que devem ser interligados aos circuitos dos atuadores.

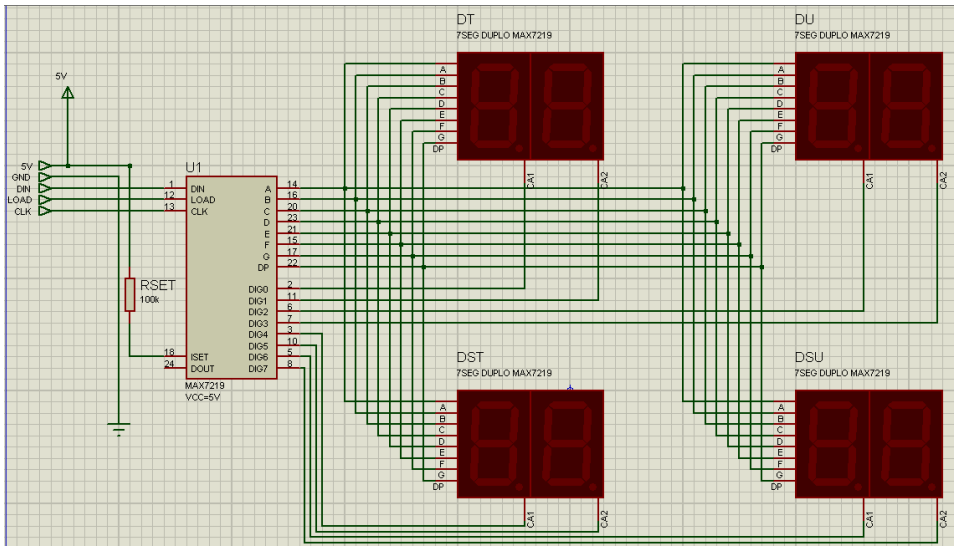
***Figura 3.1-22** - Subcircuito UC - SENSORES.*



Fonte: Elaborado pelo autor.

O conector `JDHT` deve ser interligado no Arduino para que o mesmo possa realizar a leitura da temperatura e umidade. Os demais conectores conectam-se ao Arduino para que possam informar se algum dos botões foi pressionado ou não.

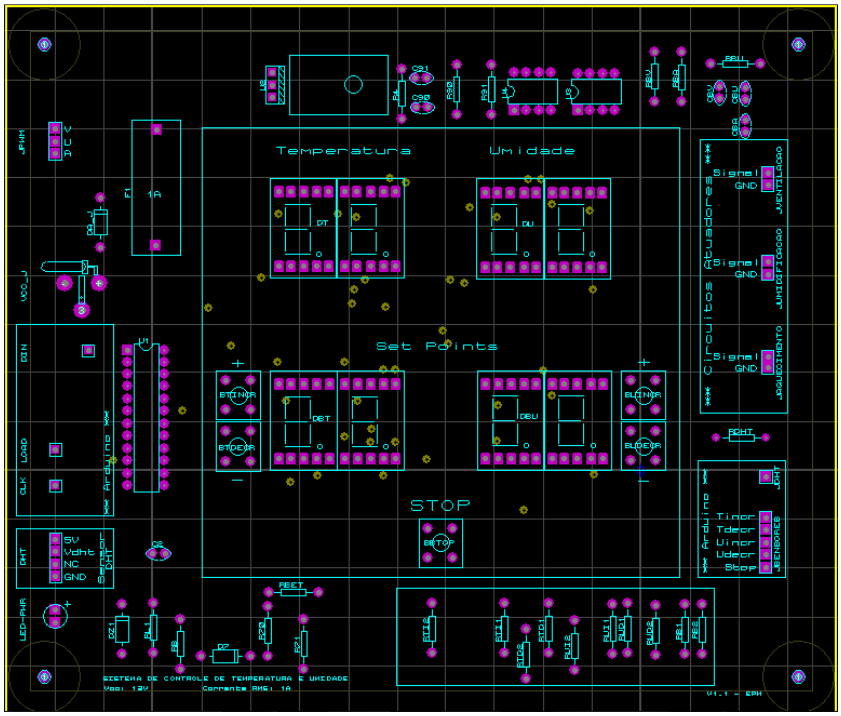
Figura 3.1-23 *- Subcircuito DISPLAYS.*



Fonte: Elaborado pelo autor.

Esse último subcircuito recebe pelo conector `DIN` dados seriais informando ao `MAX7219` qual o *display* ligar e qual valor apresentar. Todos os segmentos (de `A` a `G` e o ponto `DP`) dos *displays* catodo de 7 segmentos são interligados, sendo que o CI `MAX7219` controla pelo pino de *ground* (`GND`) qual *display* deverá ligar. Como o CI liga e desliga os *displays* em alta frequência, para um ser humano é como se os *displays* estivessem sempre ligados.

Figura 3.1-24 *- Placa de circuito impresso do circuito de interface com o operador.*



Fonte: Elaborado pelo autor.

A **Figura 3.1-24** representa a placa de circuito impresso desenvolvida para ser a interface com o operador, de modo a poder visualizar as medições, parâmetros e controles do sistema. Na região central da placa há oito *displays* de 7 segmentos para apresentação da temperatura e umidade atuais e dos valores definidos para *set point*, além dos botões. Ao lado esquerdo estão dispostos os conectores de entrada e à direita os conectores de saída.

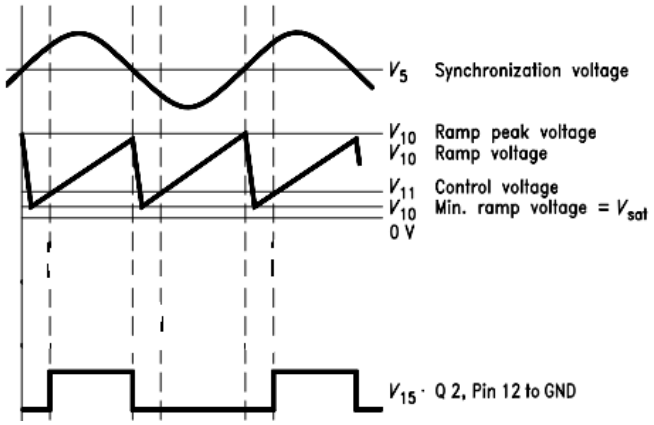
3.1.5 Circuito de aquecimento

Após o Sistema de Inferência Fuzzy calcular o novo valor da potência a ser aplicado ao atuador de aquecimento, esse o repassa ao módulo MATLAB, o qual por sua vez informa à Interface Software-Hardware via comunicação *serial* com o Arduino. Por último, então, é necessário repassar o valor da potência ao circuito de aquecimento, o qual controlará a potência dissipada pela resistência.

Como a resistência utilizada trabalha a uma tensão de 220 V em rede de frequência alternada de 60 Hz, a estratégia adotada para que se dissipe calor de forma proporcional ao valor calculado pelo *Fuzzy* é através do controle do ângulo de fase, ou seja, o circuito será "cortado" em determinada faixa de frequência, e será "fechado" nas demais.

Para o controle do ângulo de fase, utiliza-se o CI TCA785, o qual envia um sinal para um acoplador óptico que está interligado a um TRIAC. Portanto, o TRIAC trabalha a tensões de 220 V, enquanto os demais componentes ficam expostos a uma tensão de 5 ou 12 V.

***Figura 3.1-25** - Comportamento do CI TCA785 para controle de ângulo de fase.*



Fonte: ww.infineon.com

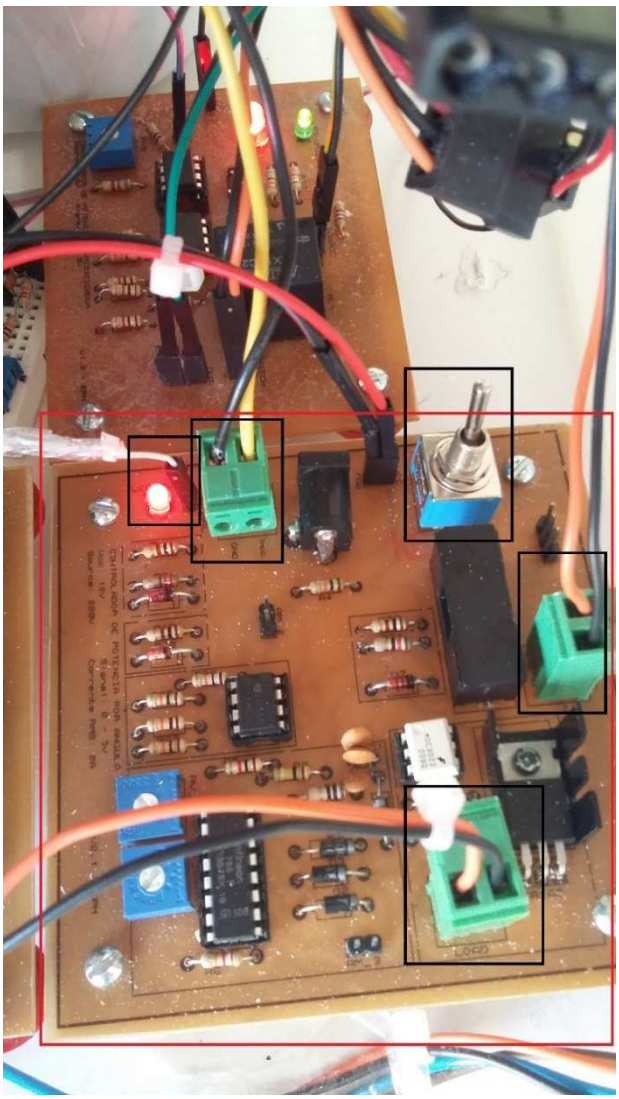
A **Figura 3.1-25** demonstra a forma de trabalho do TCA785, o qual recebe no seu pino 5 o sinal da rede senoidal através de um resistor ôhmico de alta impedância para que possa

assim identificar o ângulo de fase. No pino 10 gera-se um sinal em forma de rampa, a qual servirá de referência para o sinal de controle. O pino 11 recebe um sinal de controle, e no pino 15 se têm o sinal de saída conforme o sinal de controle. Quanto mais próximo de zero for o sinal do pino 11, maior será a largura do sinal gerado no pino 15, e quanto mais próximo for o sinal do valor de pico da rampa, menor será a largura.

Dessa forma, é possível controlar o quanto de potência é entregue na carga através do TRIAC, pois só circulará corrente enquanto o sinal do pino 15 estiver em nível alto.

$$V_{out} = V_{ref} - V_{in} \qquad (3.1\text{-}2)$$

***Figura 3.1-26** - Circuito de aquecimento.*



Fonte: Elaborado pelo autor.

Porém, o circuito de aquecimento recebe um sinal compreendido entre 0 e 5 V, onde 5 V equivale à potência máxima, e 0 V à potência mínima. Como o TCA785 trabalha de forma inversa, necessita-se inverter o sinal de entrada. Para tanto, utilizou-se um amplificador operacional configurado como subtrator, o qual funciona da seguinte forma: entra-se com um sinal de 5 V como referência (V_{ref}) e com o sinal a ser subtraído (V_{in}). O sinal de saída (V_{out}) será então a diferença entre os sinais de entrada, conforme **Equação (3.1-2)**.

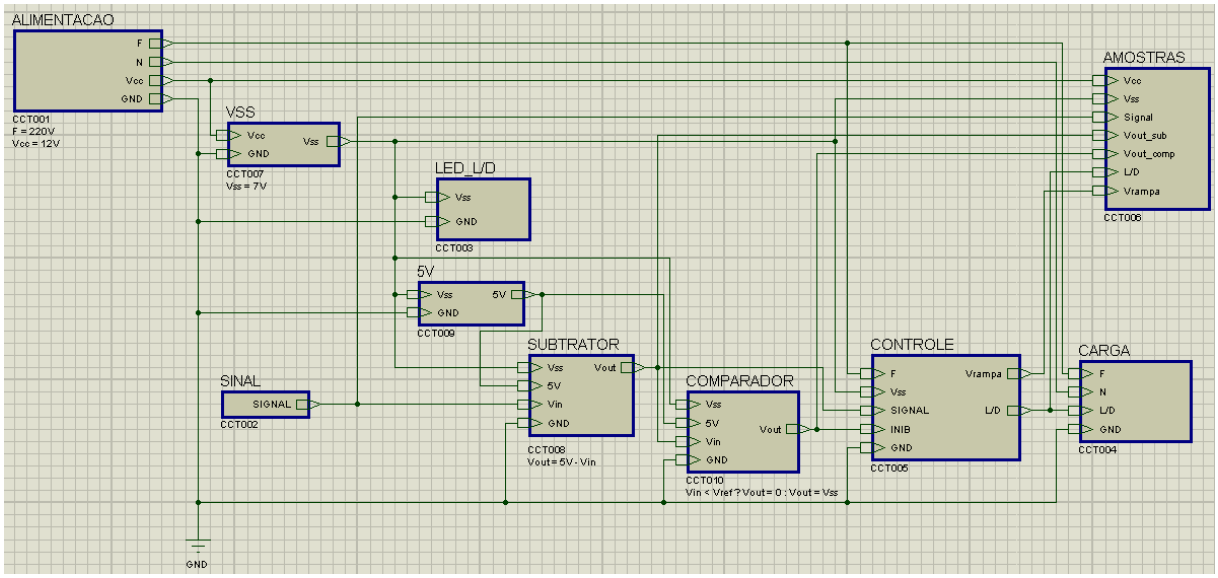
Na **Figura 3.1-26** é possível verificar a placa de circuito impresso já confeccionada, com alguns componentes destacados, que são, em ordem da esquerda para a direita:

- LED indicador de circuito on/off;

- Conector que recebe o sinal V_{in} provindo da interface Software-Hardware;
- Conector para alimentação 12 V;
- Chave liga/desliga;
- Conector para alimentação 220 V;
- Conector de saída para o atuador.

O diagrama de blocos da **Figura 3.1-27** representa o esquemático do circuito de aquecimento, no qual entra-se com uma alimentação de 220 V para a resistência e 12 V para o sistema eletrônico através do subcircuito ALIMENTACAO.

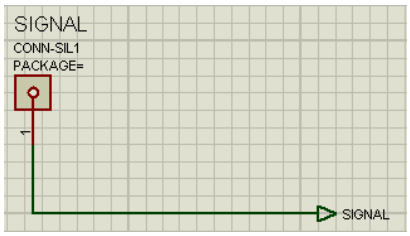
***Figura 3.1-27** - Diagrama de blocos do circuito de aquecimento.*



Fonte: Elaborado pelo autor.

Os subcircuitos VSS, LED_L/D e 5V são semelhantes aos existentes na **Seção 3.1.4**, e, portanto, não serão descritos.

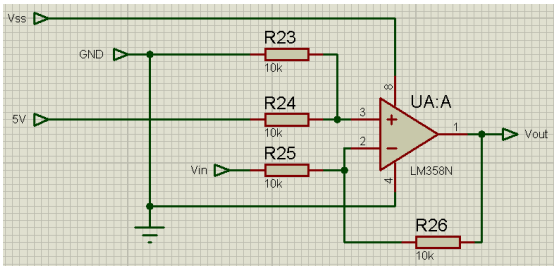
***Figura 3.1-28** - Subcircuito SIGNAL.*



Fonte: Elaborado pelo autor.

Contém apenas um conector para receber o sinal referente à potência a ser dissipada pela resistência.

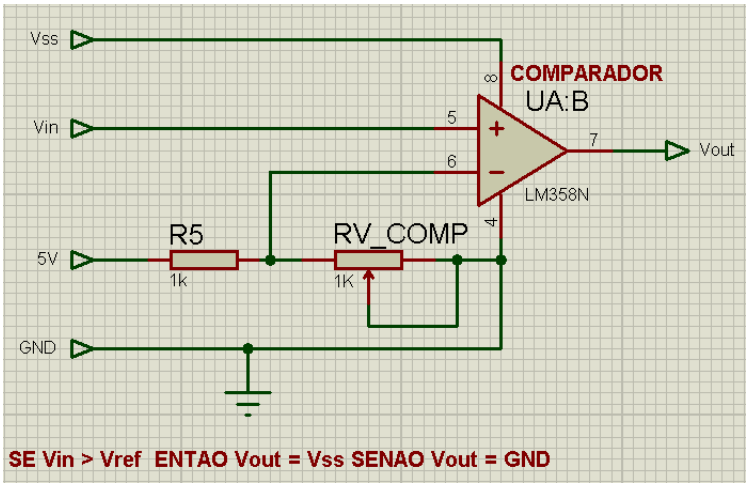
***Figura 3.1-29** - Subcircuito SUBTRATOR.*



Fonte: Elaborado pelo autor.

Subcircuito responsável por aplicar a **Equação (3.1-2)** para que inverta o sinal de entrada, através do uso de um amplificador operacional configurado como subtrator.

***Figura 3.1-30** - Subcircuito COMPARADOR.*



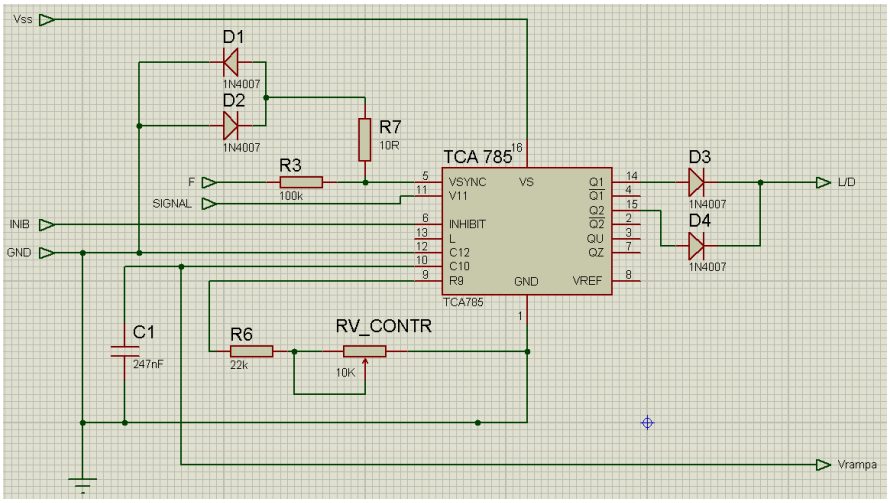
Fonte: Elaborado pelo autor.

Para evitar a aplicação de potências muito baixas, esse subcircuito verifica se V_{in} é menor do que o valor existente no pino 6 do amplificador operacional. Caso seja, o sinal na saída será de 0 V, senão terá o valor de V_{ss}, o qual será repassado ao TCA785, no seu pino de inibição (pino 6).

A **Figura 3.1-31** representa o principal subcircuito do circuito de aquecimento, o qual utiliza o `TCA785` para realizar o controle de quanta potência será dissipada pelo atuador, conforme representado na **Figura 3.1-25**.

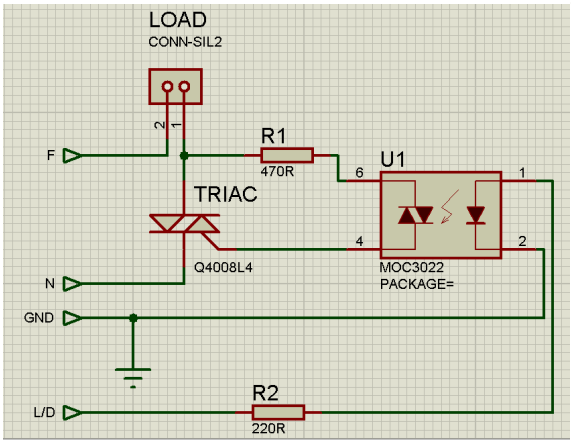
Para a geração da rampa utilizada pelo TCA785, utilizam-se o capacitor C1, o resistor R6 e o trimpot RV_CONTR, os quais definem o tempo que a rampa leva para atingir seu pico. O componente R3 é um resistor ôhmico de alta impedância que permite ao CI receber o sinal da rede de 220 V. No pino 11 entra-se com o sinal que será utilizado para definir qual o intervalor do ângulo de fase será utilizado para alimentar a resistência. Nos pinos 14 e 15 são obtidos os sinais de saída, sendo que o primeiro se refere à faixa de 180 a 360º, e o segundo à faixa de 0 a 180º.

***Figura 3.1-31** - Subcircuito CONTROLE.*



Fonte: Elaborado pelo autor.

***Figura 3.1-32** - Subcircuito CARGA.*

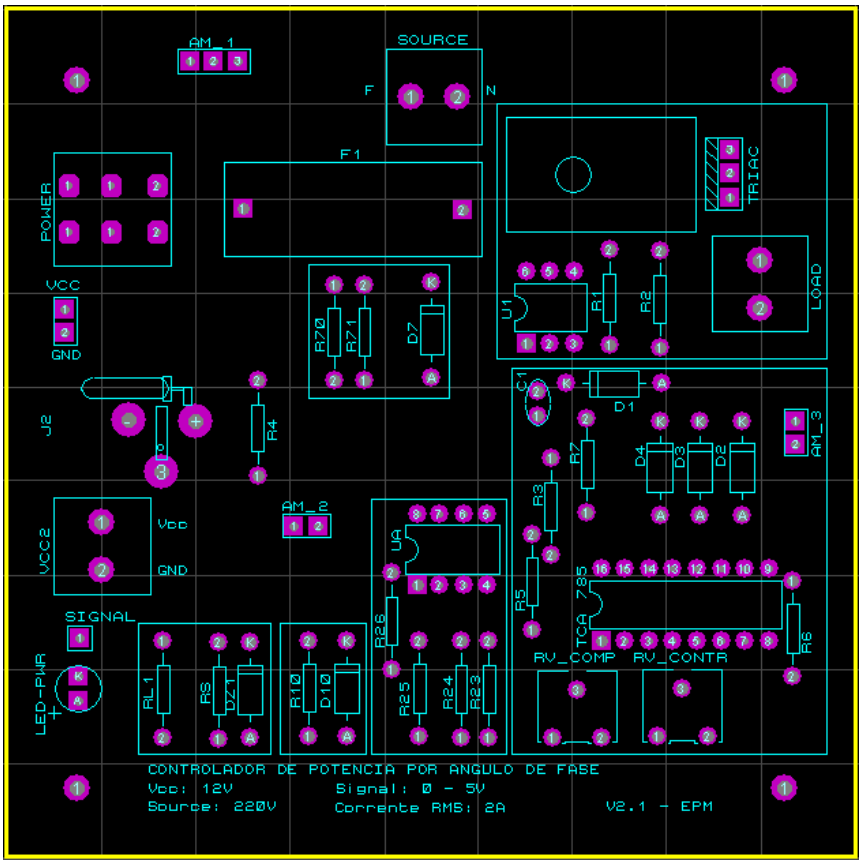


Fonte: Elaborado pelo autor.

Esse subcircuito recebe o sinal de saída do TCA785 e o repassa ao acoplador U1. Se o nível for alto, o acoplador irá acionar o TRIAC encapsulado, e com isso o componente TRIAC é disparado, permitindo assim que corrente circule pela resistência.

O subcircuito AMOSTRAS contém apenas pinos de saída para leitura de alguns sinais de interesse do circuito.

Figura 3.1-33 *- Placa de circuito impresso do circuito de aquecimento.*



Fonte: Elaborado pelo autor.

A **Figura 3.1-33** representa a placa de circuito impresso projetada para o sistema de aquecimento, semelhante à **Figura 3.1-26**, de modo a controlar a potência dissipada na resistência (a qual está contida dentro da caixa destacada em preto na **Figura 3.1-34**). Ao lado esquerdo (conectores `VCC`, `J2` e `VCC2`) e acima (conector `SOURCE`) estão dispostos os conectores de entrada e à direita conecta-se a carga a ser controlada (conector `LOAD`). O componente `F1` representa um fusível, para proteção contra sobrecargas de corrente.

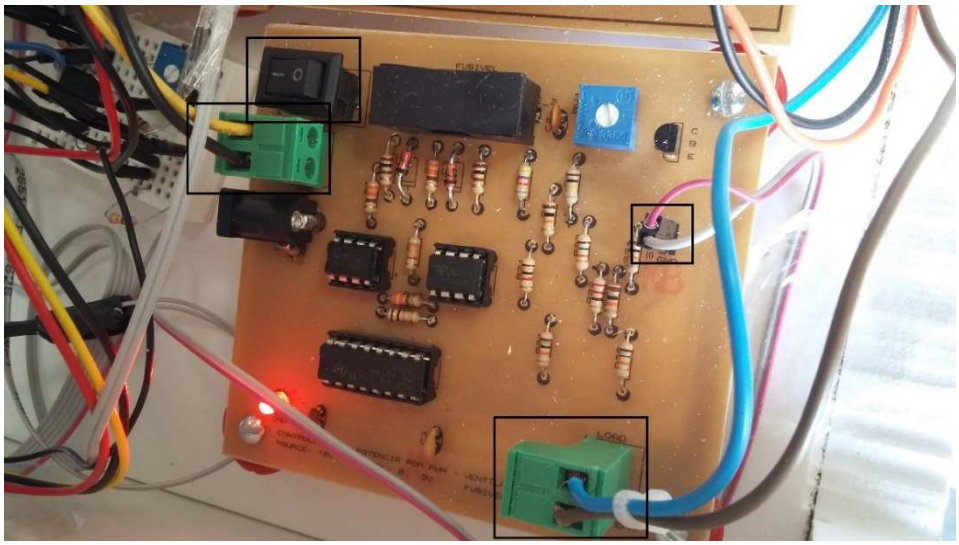
Figura 3.1-34 - *Destacado o invólucro dos atuadores de aquecimento e ventilação.*

Fonte: Elaborado pelo autor.

3.1.6 Circuito de ventilação

O circuito de ventilação aciona um cooler, o qual está tendo sua potência controlada através da geração de um sinal PWM. Dessa forma, o circuito recebe um sinal contido entre 0 e 5 V e o converte para um PWM de largura proporcional através da combinação entre o oscilador `555` e amplificador operacional.

Figura 3.1-35 - *Circuito de ventilacão.*

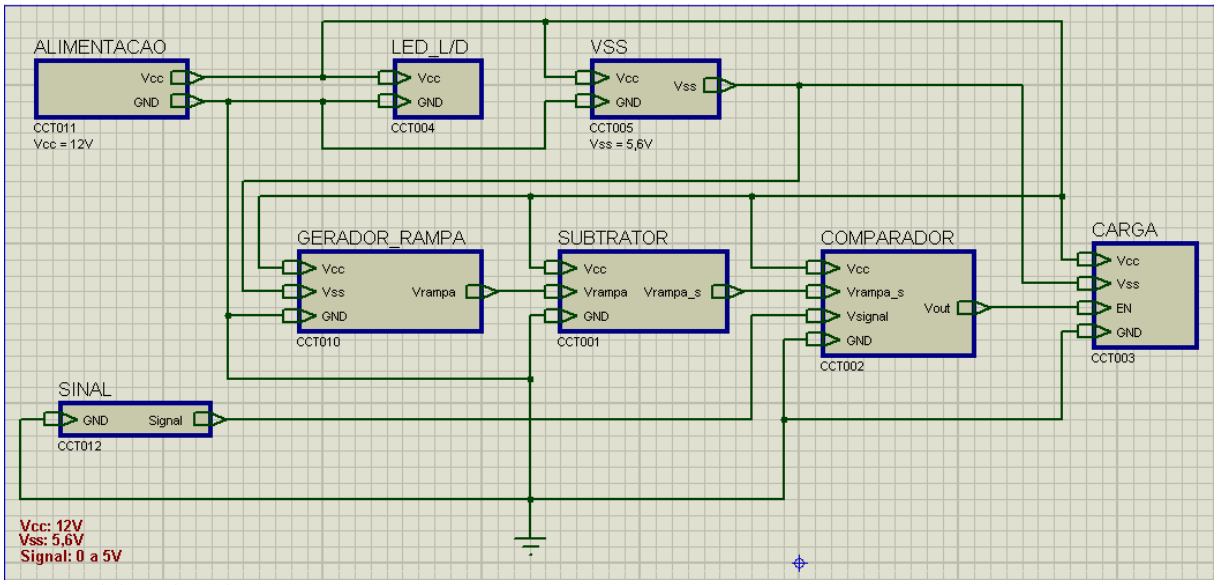


Fonte: Elaborado pelo autor.

A **Figura 3.1-35** apresenta a placa de circuito impresso confeccionada, contendo alguns componentes em destaque, os quais da esquerda para a direita são:

- Conector de alimentação 12 V;
- Chave liga/desliga;
- Conector que recebe o sinal de entrada;
- Conector de saída para o atuador.

Figura 3.1-36 *- Diagrama de blocos do circuito de ventilação.*



Fonte: Elaborado pelo autor.

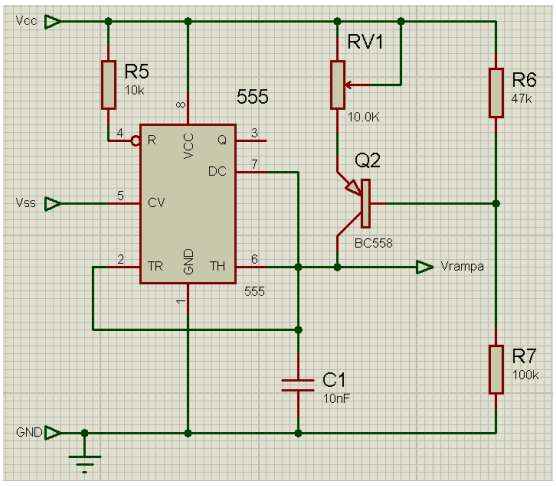
Os subcircuitos `ALIMENTACAO`, `LED_L/D` e `VSS` são semelhantes aos existentes nos circuitos anteriores, e, portanto, não serão detalhados.

O subcircuito `GERADOR_RAMPA` gera um sinal em formato de rampa através do uso do oscilador 555 configurado como astável.

O oscilador 555 gera sinais com amplitude entre 1/3 e 2/3 do valor de alimentação do pino 8. Porém é possível alterar o valor de pico através do pino 5.

Portanto, para limitar o valor de pico da rampa conectou-se ao pino 5 o sinal gerado no subcircuito `VSS`, de 5,6 V. Dessa forma, a rampa estará limitada ao valor nominal de 5,6 V.

***Figura 3.1-37** - Subcircuito GERADOR_RAMPA.*

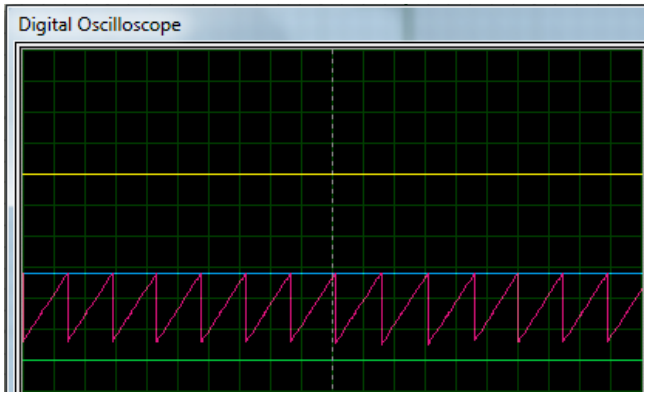


Fonte: Elaborado pelo autor.

Na **Figura 3.1-38** é possível verificar uma simulação gerada para o subcircuito, demonstrando a geração do sinal em rampa. A primeira linha horizontal (em amarelo), representa o sinal de `Vcc`, de valor nominal de 12 V.

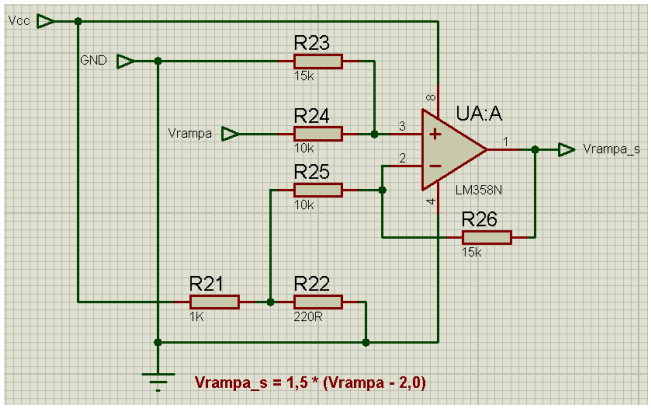
A segunda linha horizontal (em azul) representa o sinal gerado por `Vss`, de valor nominal de 5,6 V. E a última linha horizontal (em verde), representa o *ground* do circuito. Dessa forma é possível verificar que a rampa está compreendida entre a segunda e a última linha horizontal.

***Figura 3.1-38** - Sinal em rampa gerado pelo oscilador `555`.*



Fonte: Elaborado pelo autor.

***Figura 3.1-39** - Subcircuito SUBTRATOR.*



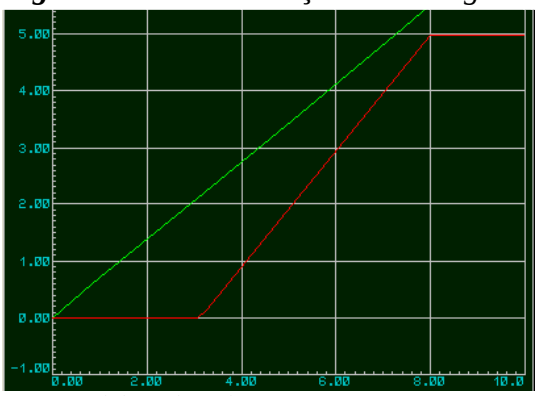
Fonte: Elaborado pelo autor.

Esse subcircuito reduz o sinal gerado na rampa, pois o menor valor não é 0 V, mas sim um valor compreendido entre 1 e 2 V. Para realizar a subtração, utiliza-se um amplificador configurado como subtrator com ganho de 1,5.

Na **Figura 3.1-40** há a apresentação de um gráfico gerado a partir de simulação realizada no subcircuito `SUBTRATOR`.

O sinal de amplitude maior (de cor verde) representa um sinal gerado pela rampa, enquanto o sinal de menor amplitude (de cor vermelha), representa a saída do subcircuito.

***Figura 3.1-40** - Simulação de sinal gerado pelo subcircuito SUBTRATOR.*

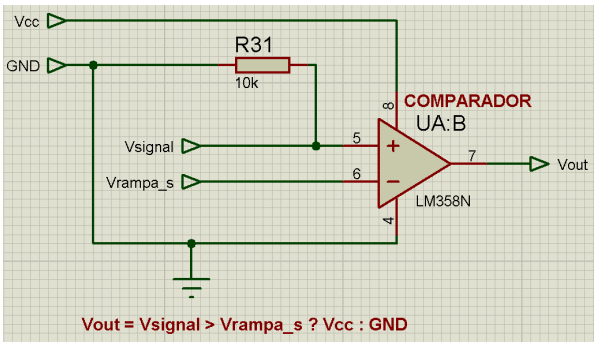


Fonte: Elaborado pelo autor.

É possível perceber pelo gráfico que a amplitude da segunda curva é maior que a primeira, sendo o seu valor nominal de 1,5 (quociente entre os valores dos resistores `R26` e `R24`).

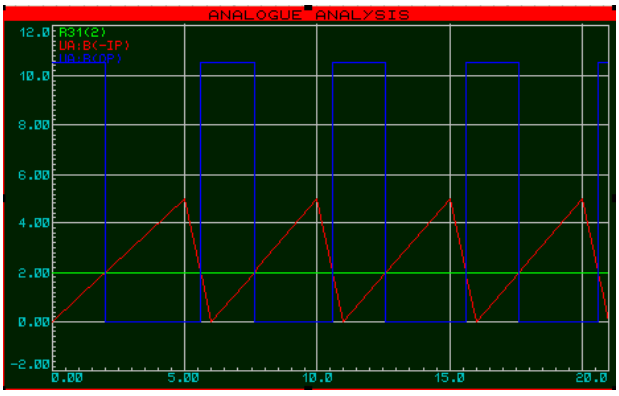
Esse subcircuito realiza a comparação entre os sinais V_{signal} (o qual representa o valor calculado pelo Sistema de Inferência *Fuzzy*) e o valor gerado pela rampa após a subtração. Dessa forma gera-se um sinal PWM, sendo que a largura será tanto maior quanto maior for o valor de V_{signal}. Quanto à amplitude, essa será o valor de V_{cc} (valor nominal de 12 V) ou 0 V.

***Figura 3.1-41** - Subcircuito COMPARADOR.*



Fonte: Elaborado pelo autor.

***Figura 3.1-42** - Simulação de sinal gerado pelo subcircuito COMPARADOR.*

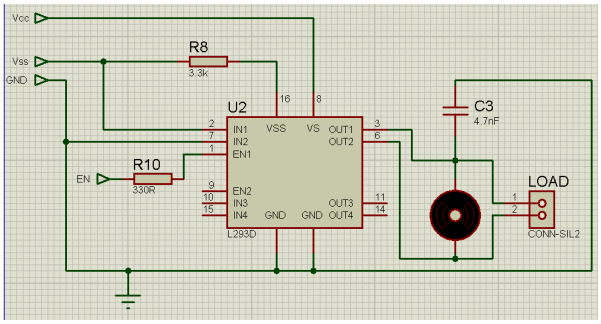


Fonte: Elaborado pelo autor.

A **Figura 3.1-42** é uma simulação representando o sinal gerado pelo subcircuito COMPARADOR para um sinal de 2 V (portanto o PWM deve possuir largura de 40%) enviado pelo *Fuzzy* (linha horizontal verde). Enquanto o valor do sinal é maior que o valor da rampa (sinal vermelho), a amplitude do PWM é o valor nominal de V_{cc}. Caso contrário, a amplitude será de 0 V.

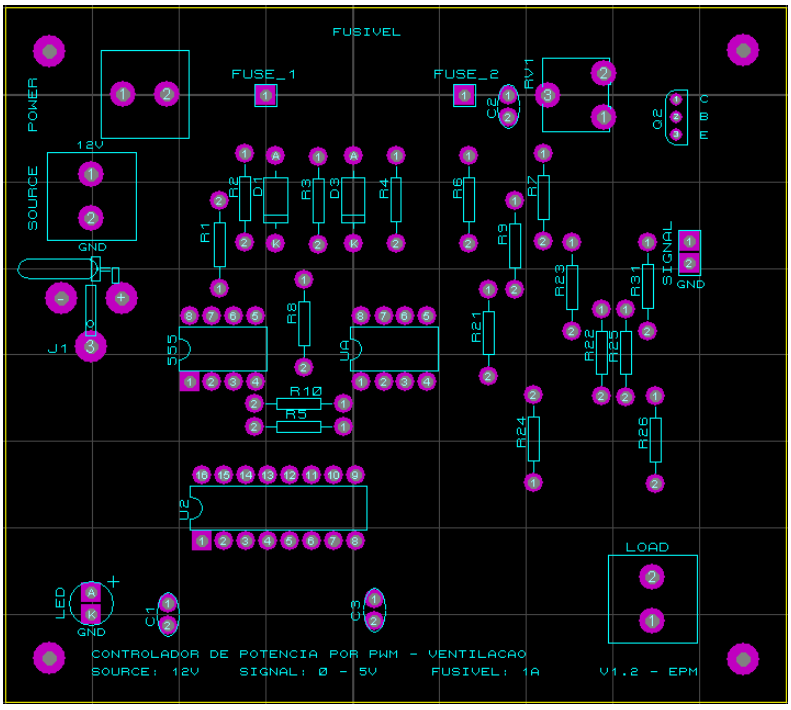
O último subcircuito do circuito de ventilação contém basicamente uma ponte-H que recebe no seu pino 1 (ENABLE1) o sinal do PWM gerado pelo subcircuito COMPARADOR. Conectado aos pinos 3 e 6 estará o cooler a ser controlado. O capacitor C3 possui a função de um filtro simples, no intuito de evitar que frequências altas que possam ser geradas pelo motor entrem no circuito através da ponte-H.

***Figura 3.1-43** - Subcircuito CARGA.*



Fonte: Elaborado pelo autor.

***Figura 3.1-44** - Placa de circuito impresso do circuito de ventilação.*



Fonte: Elaborado pelo autor.

A **Figura 3.1-44** representa a placa de circuito impresso projetada para o sistema de ventilação, semelhante à **Figura 3.1-35**, de modo a controlar a potência média entregue ao *cooler* (o qual está contido dentro da caixa destacada em preto na **Figura 3.1-34**). Ao lado esquerdo (conectores SOURCE e J1) e direita (conector SIGNAL) estão dispostos os conectores de entrada e à direita conecta-se a carga a ser controlada (conector LOAD). O componente F1 representa um fusível, para proteção contra sobrecargas de corrente. O componente UA representa o amplificador operacional e o U2 a ponte-H.

3.1.7 Circuito de umidificação

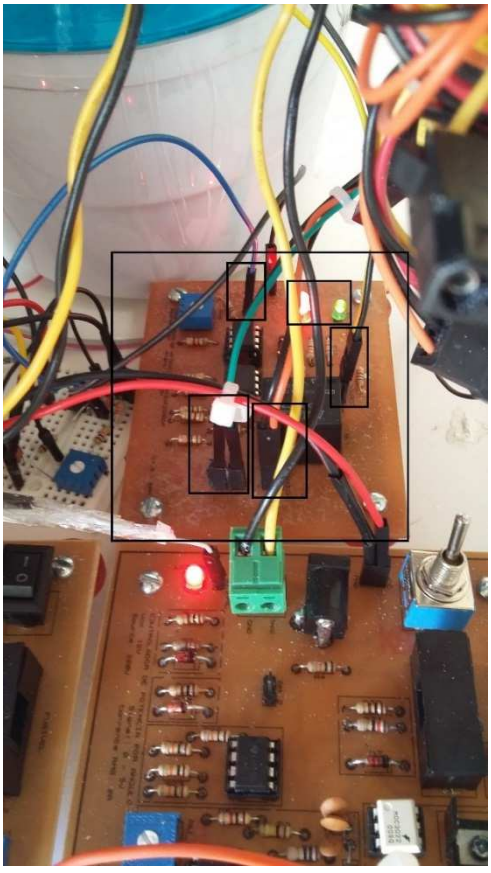
O circuito de umidificação aciona um umidificador de ar, o qual está tendo sua potência controlada através de um sinal que estar entre 0 e 12 V, onde zero equivale à máxima potência e 12 V à mínima potência. Dessa forma, o circuito recebe um sinal contido entre 0 e 5 V e o amplifica por um fator de 1,2. Após, o sinal resultante é invertido conforme **Equação (3.1-3)**:

$$V_{out} = 12 - 1{,}2 \ * V_{signal} \qquad (3.1\text{-}3)$$

Figura 3.1-45 *- Simulação do sinal de saída conforme recebe um sinal de entrada crescente.*

A **Figura 3.1-45** representa simulação gerada para verificar o comportamento do sinal de saída conforme aumenta-se o sinal de entrada. Perceba que quanto maior é o sinal de entrada, menor será o sinal de saída. Portanto, o umidificador possui uma geração maior de umidade conforme se diminui o nível de tensão.

Figura 3.1-46 *- Circuito de umidificação.*



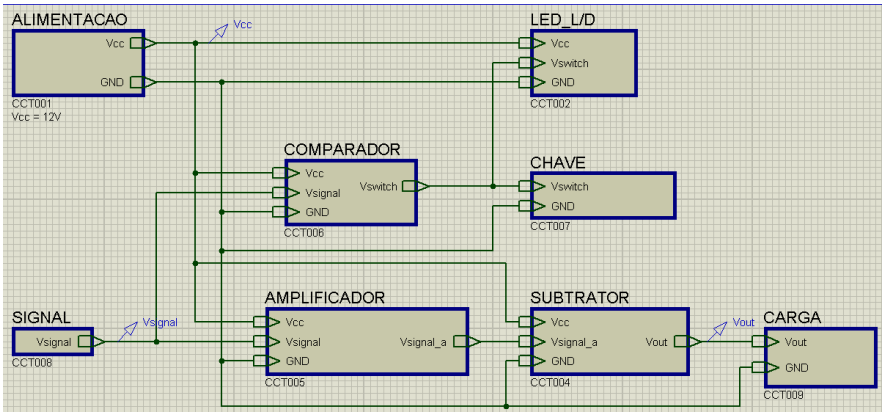
Na **Figura 3.1-46** apresenta-se a placa de circuito impresso, com alguns componentes em destaque, de cima para baixo:

- Conector que recebe o sinal de entrada;
- LED's que identificam se o circuito está ligado e se o umidificador está ligado ou não;
- Conector (amarelo) de alimentação 12 V;
- Conector (verde) de saída para o atuador informando a potência;
- Conector (laranja) de saída para chaveamento do atuador.

O diagrama de blocos abaixo representa o esquemático do circuito de umidificação, no qual entra-se com uma alimentação de 12 V para o sistema eletrônico através do subcircuito ALIMENTACAO.

Como os subcircuitos `ALIMENTACAO`, `LED_L/D`, `SIGNAL` e `SUBTRATOR` são semelhantes aos existentes nos circuitos anteriores, portanto, não serão detalhados.

***Figura 3.1-47** - Diagrama de blocos do circuito de umidificação.*

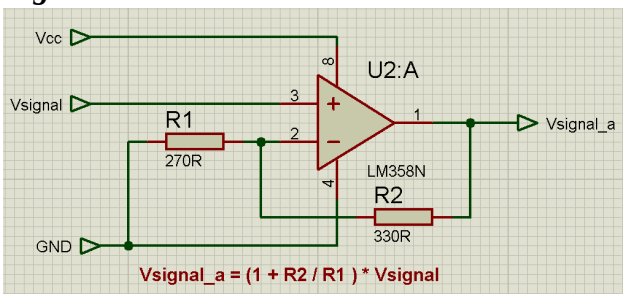


Fonte: Elaborado pelo autor.

O primeiro subcircuito analisado realiza a amplificação do sinal de entrada por um fator de 1,2 (`R2 / R1`) através do amplificador operacional nomeado no circuito como `U2`.

Esse sinal amplificado será então utilizado como entrada no subcircuito `SUBTRATOR`.

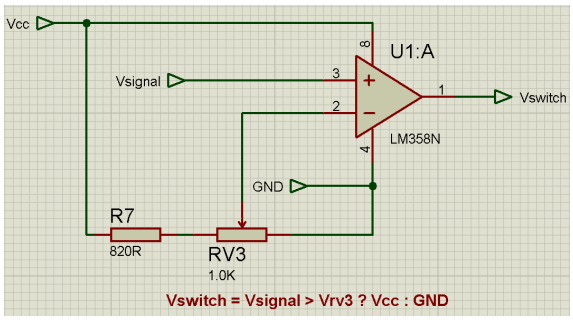
***Figura 3.1-48** - Subcircuito AMPLIFICADOR.*



Fonte: Elaborado pelo autor.

Após obtido o sinal de saída do subcircuito SUBTRATOR, esse é inserido no subcircuito CARGA, o qual contém apenas um conector a ser interligado no umidificador de ar.

***Figura 3.1-49** - Subcircuito COMPARADOR.*



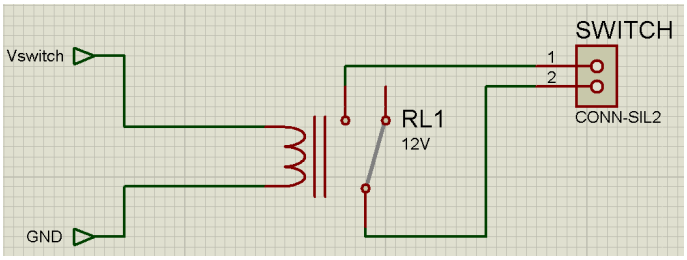
Fonte: Elaborado pelo autor.

Esse subcircuito verifica se o nível do sinal de entrada é maior que o sinal definido através do TRIMPOT `RV3`. Caso seja, o amplificador `U1` coloca na saída o valor V_{CC}. Caso contrário, terá na saída do valor do *ground*.

O sinal gerado possui a atribuição de acionar ou não a chave do atuador.

Definido o sinal a ser aplicado na chave através do amplificador operacional do subcircuito anterior, esse é repassado ao relê `RL1`, o qual será acionado caso receba um nível de tensão ALTO. Caso receba um sinal BAIXO, seu contato permanece na posição de normalmente fechado, e, portanto, abre a chave.

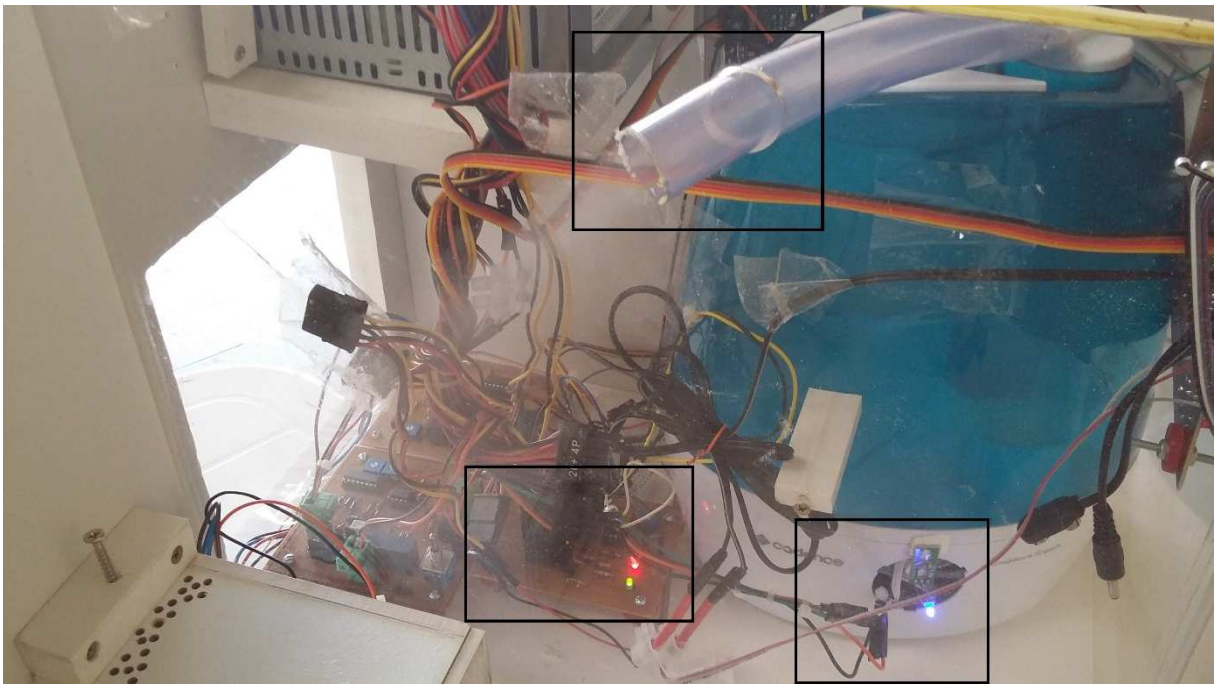
***Figura 3.1-50** - Subcircuito CHAVE.*



Fonte: Elaborado pelo autor.

Na **Figura 3.1-51** é possível verificar o umidificador de ar utilizado, sendo acima a saída de ar úmido, abaixo a entrada dos fios provenientes da placa de circuito impresso, a qual está destacada na parte inferior esquerda da figura.

Figura 3.1-51 *- Atuador - Umidificador de ar.*

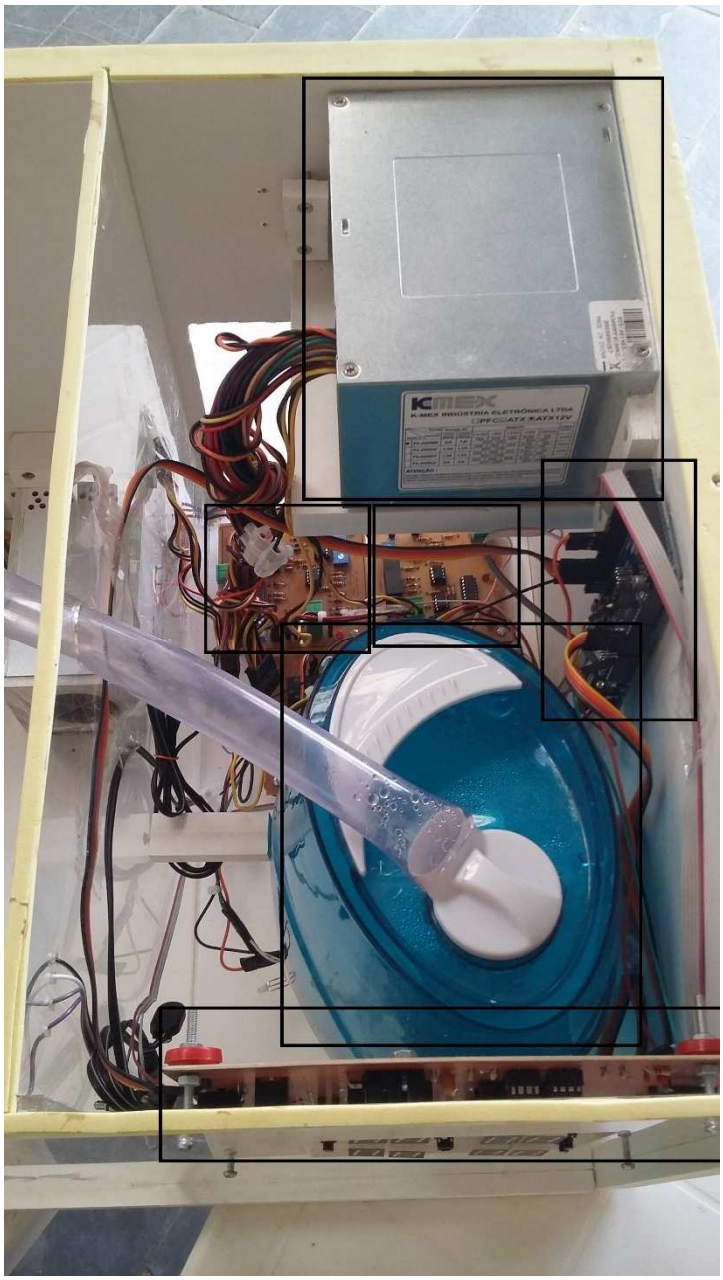


Fonte: Elaborado pelo autor.

3.2 Montagem do protótipo

Para a realização dos testes do sistema de controle de temperatura e umidade foi desenvolvida uma maquete, a qual possui dois compartimentos: um contem os circuitos eletrônicos necessários (à direita) e o outro compartimento (à esquerda) representa o ambiente a ser controlado.

***Figura 3.2-1** - Vista superior do compartimento da direita da maquete.*



Fonte: Elaborado pelo autor.

No compartimento dos circuitos eletrônicos (**Figura 3.2-1**) será fixada em sua parte frontal a placa de circuito descrita na **Seção 3.1.4**, a qual permite a interação do operador com o sistema. Essa placa está conectada ao computador onde se localiza o MATLAB através de uma placa de prototipagem Arduino, que por sua vez se comunica com o MATLAB através de cabo USB.

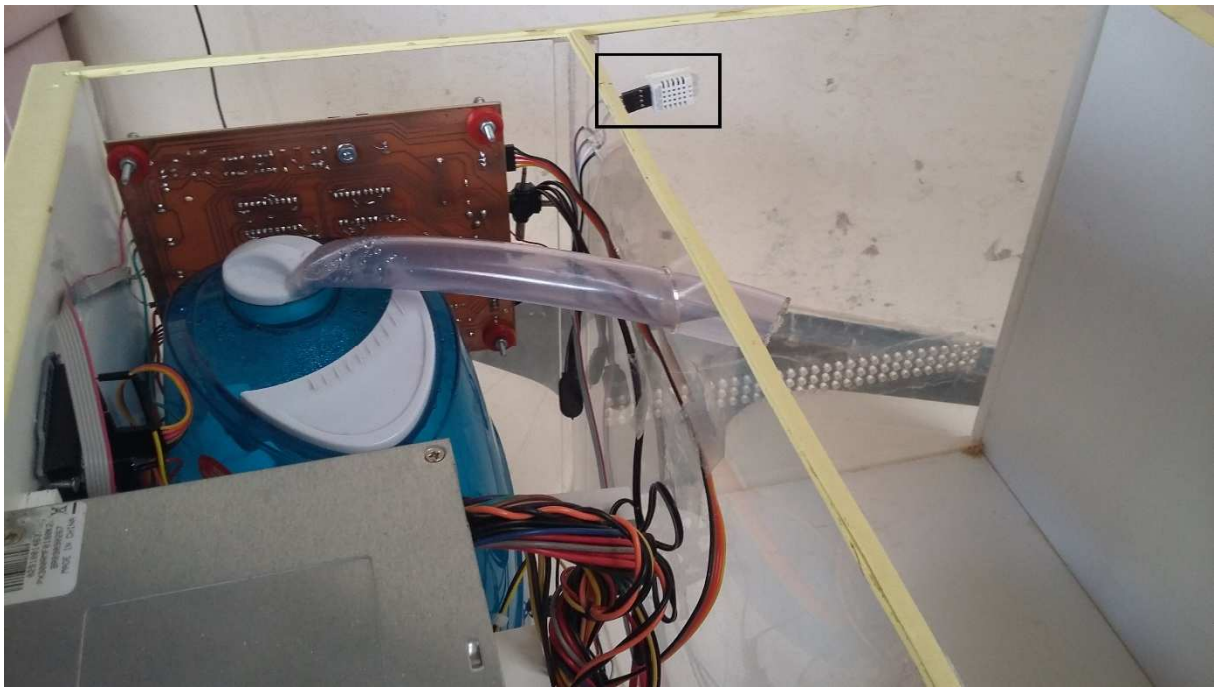
Dentro desse mesmo compartimento, há os circuitos descritos nas **Seções 3.1.5**, **3.1.6** e **3.1.7**, os quais controlam os atuadores de aquecimento, ventilação e umidificação, respectivamente.

O atuador de umidificação está localizado dentro do primeiro compartimento, e envia umidade através de uma mangueira que passa para o segundo compartimento.

Os demais atuadores e os sensores estão localizados no segundo compartimento, conforme **Figura 3.2-2**.

Todos os atuadores se comunicam com suas respectivas placas de circuito impresso através de rede cabeada.

***Figura 3.2-2** - Vista superior da maquete apresentando o sensor DHT22 em destaque.*



Fonte: Elaborado pelo autor.

Para iniciar a execução do sistema de controle, será necessário iniciar o programa do MATLAB conforme **Apêndice 1.1**.

Através do MATLAB é possível acompanhar o funcionamento do sistema com o auxílio de gráficos gerados conforme **Figura 3.1-2**. Caso o operador pressione o botão `STOP`, localizado na Interface MATLAB (conforme **Seção 3.1.1**) ou pela Interface *Software-Hardware* (conforme **Seção 3.1.3**, e botão localizado abaixo dos *displays* na **Figura 3.1-10**), o MATLAB interromperá a execução do *timer* responsável pela comunicação via *serial* e pela atualização de todos os dados e gráficos do programa.

4 TESTES E RESULTADOS

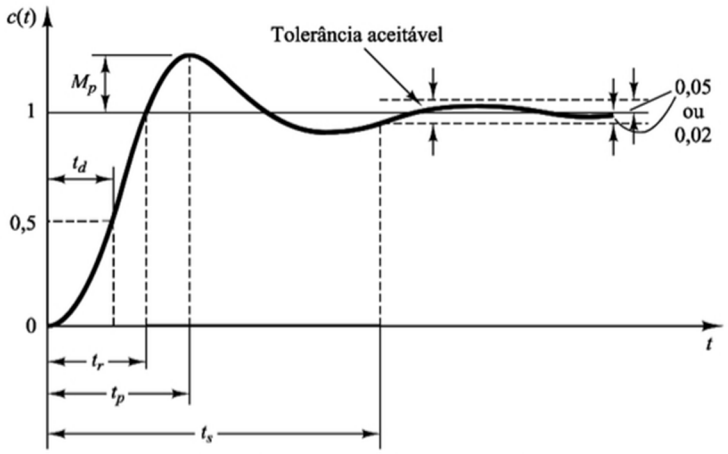
Este capítulo apresentará os testes e resultados obtidos no decorrer do desenvolvimento do projeto, tendo sido dividido em cenários para melhor organização.

Conforme OGATA (2003), na prática, antes de se atingir um regime permanente, a resposta do sistema de controle apresenta oscilações. E, portanto, na especificação das características do sistema é comum se especificar alguns parâmetros de resposta.

4.1 Considerações iniciais

Nos testes realizados foram monitorados cinco parâmetros, os quais estão ilustrados na **Figura 4.1-1**, que são:

Figura 4.1-1 *- Parâmetros analisados numa curva de resposta.*



Fonte: OGATA, 2013, p. 189

Δ: Diferença percentual entre o último valor lido pelo sensor e o valor especificado no *set point*;

t_r: Tempo de subida do sistema. Informa o momento em que se atingiu o valor de *set point* desejado;

M_p: Conhecido como máximo sobre-sinal, é o valor máximo (percentual) de pico da curva de resposta, medido a partir do *set point* especificado;

t_p: Tempo de pico, ou seja, o momento em que foi atingido o máximo sobre-sinal (M_p).

$t_{5\%}$: Tempo de acomodação do sistema, no qual se verifica o momento em que se alcança uma diferença de 5% ou menos em relação ao *set point*.

Além dos parâmetros acima citados, OGATA (2003) especifica ainda o tempo de atraso (**t_d**) e o tempo de acomodação de 2% (**$t_{2\%}$**), os quais não foram monitorados nos testes.

Além dos parâmetros a serem monitorados, acima citados, para alguns testes também foram considerados os valores mínimo e máximo de temperatura e umidade.

A seguir segue descrição sucinta de cada cenário de teste.

a) **Primeiro cenário:** verificar o comportamento do Sistema de Inferência *Fuzzy* (FIS) para elevação da umidade em 5%.
b) **Segundo cenário:** verificar comportamento do FIS para elevação da umidade em 10%.
c) **Terceiro cenário:** verificar o comportamento do FIS para elevação da umidade em 20%.
d) **Quarto cenário:** verificar comportamento do FIS para redução da umidade em 5%.
e) **Quinto cenário:** verificar comportamento do FIS para redução da umidade em 6%.
f) **Sexto cenário:** verificar o comportamento do FIS para redução da umidade em 10%.
g) **Sétimo cenário:** verificar comportamento do FIS para elevação da temperatura em 5 ºC.
h) **Oitavo cenário:** verificar o comportamento do FIS para elevação da temperatura em 9 ºC.
i) **Nono cenário:** verificar o comportamento do FIS para elevação da temperatura em 10 ºC.
j) **Décimo cenário:** verificar o comportamento do FIS para redução da temperatura em 2 ºC.
k) **Décimo primeiro cenário:** verificar o comportamento do FIS para elevação da temperatura e umidade simultaneamente em 4 ºC e 5%, respectivamente.
l) **Décimo segundo cenário:** verificar o comportamento do FIS para elevação da temperatura e umidade simultaneamente em 5 ºC e 10%, respectivamente.

m) **Décimo terceiro cenário:** verificar o comportamento do FIS para elevação da temperatura e umidade simultaneamente em 7 ºC e 15%, respectivamente.
n) **Décimo quarto cenário:** verificar o comportamento do FIS para elevação da temperatura em 3 ºC e redução da umidade em 5%.
o) **Décimo quinto cenário:** verificar o comportamento do FIS para elevação da temperatura em 5 ºC e redução da umidade em 7%.

Para cada teste são apresentados os gráficos gerados no MATLAB contendo os valores históricos desde o início do teste até sua finalização, nos quais será possível verificar os parâmetros, acima expostos, para a temperatura e umidade, além de serem apresentados os valores convertidos de temperatura e umidade para o entendimento do sistema de inferência *fuzzy*, e por último os gráficos dos valores relativos de potência (onde 0 (zero) significa que o atuador deva empregar a menor potência possível, 1 (um) refere-se à máxima potência que possa ser entregue, e valores intermediários referem-se a um valor de potência proporcional) para cada um dos atuadores, conforme definidos pelo *fuzzy*. Em todos os gráficos, o eixo da abscissa conterá o tempo do teste em segundos.

4.2 Primeiro cenário – elevar umidade em 5%

4.2.1 Descrição

Este cenário de teste tem por objetivo elevar a umidade para um valor de *set point* de definido acima do valor existente dentro do ambiente a ser controlado, e para a temperatura será definido valor igual ao existente dentro do ambiente.

4.2.2 Pré-requisitos

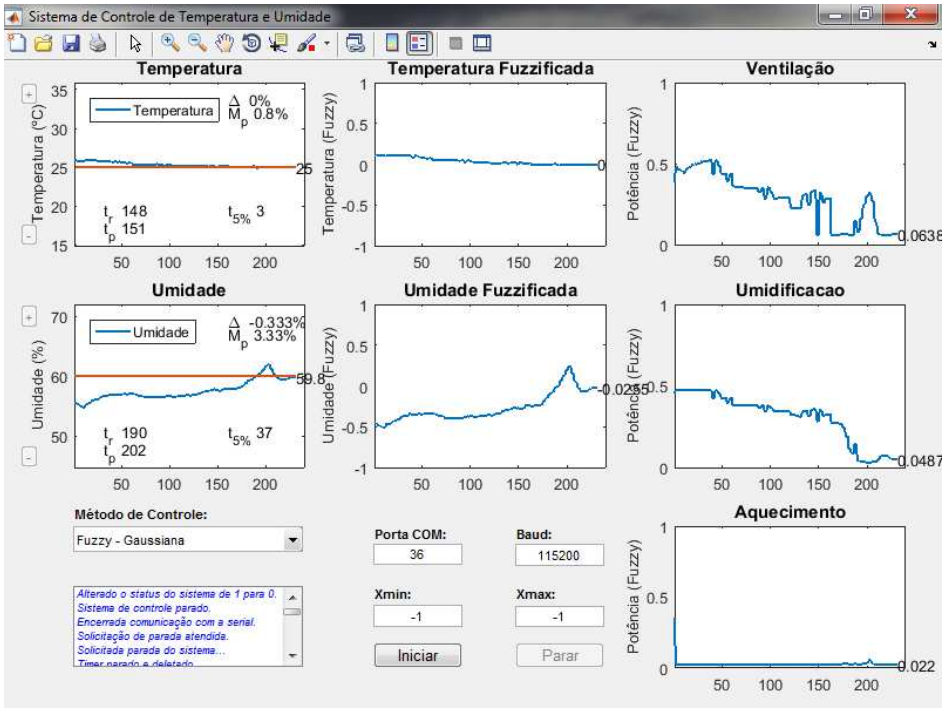
Com a umidade estável em 55%, configurou-se o sistema para atingir os 60%. Para a temperatura foi configurado *set point* igual ao valor já existente no início do teste, de 25 ºC.

4.2.3 Resultados

Conforme **Figura 4.2-1**, o sistema precisou de 190 segundos para atingir o valor de *set point* da umidade, obteve um sobressinal de 3,33% aos 202 segundos, e atingiu o valor de 5% de diferença aos 37 segundos.

Quanto à temperatura, obteve-se um sobressinal de 0,80%, mantendo-se praticamente estável por todo o tempo.

***Figura 4.2-1** - Gráficos gerados para o cenário de elevação da umidade em 5%.*



Fonte: Elaborado pelo autor.

4.3 Segundo cenário – elevar umidade em 10%

4.3.1 Descrição

Este cenário de teste tem por objetivo elevar a umidade para um valor de *set point* de definido acima do valor existente dentro do ambiente a ser controlado, e para a temperatura será definido valor igual ao existente dentro do ambiente.

4.3.2 Pré-requisitos

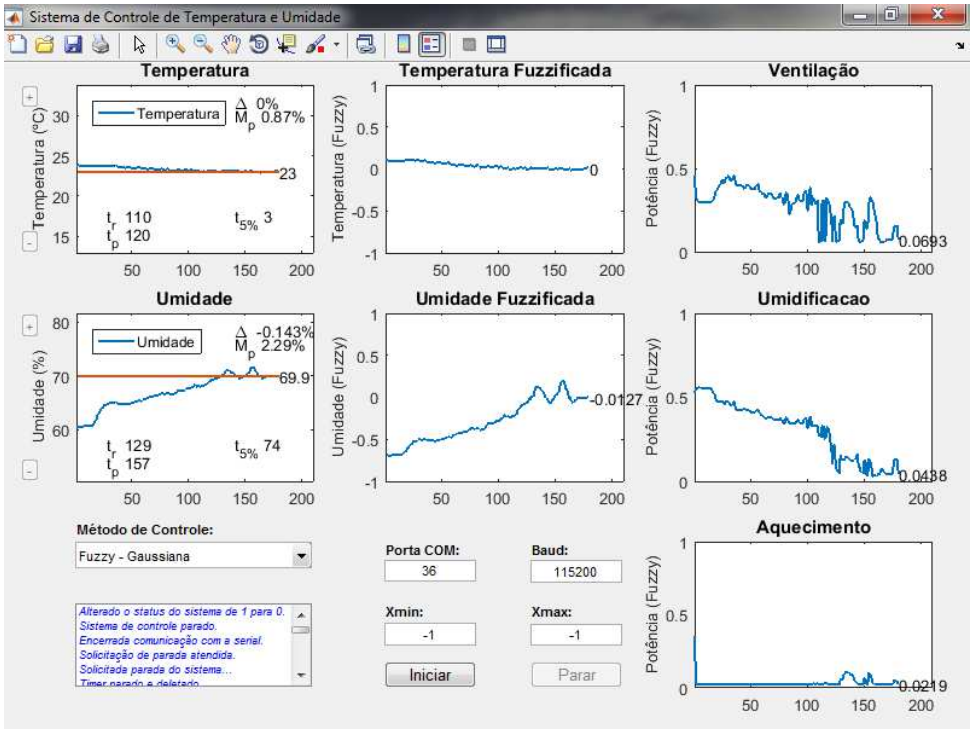
Com a umidade estável em 60%, configurou-se o sistema para atingir os 70%. Para a temperatura foi configurado *set point* igual ao valor já existente no início do teste, de 23 ºC.

4.3.3 Resultados

Conforme **Figura 4.3-1**, o sistema atingiu o valor de *set point* da umidade aos 129 segundos, com um sobressinal de 2,29% aos 157 segundos, e diferença de 5% aos 74.

Quanto à temperatura, obteve-se um sobressinal de 0,87%, mantendo-se praticamente estável por todo o tempo.

Figura 4.3-1 *- Gráficos gerados para o cenário de elevação da umidade em 10%.*



Fonte: **Elaborado pelo autor.**

4.4 Terceiro cenário – elevar umidade em 20%

4.4.1 Descrição

Este cenário de teste tem por objetivo elevar a umidade para um valor de *set point* de definido acima do valor existente dentro do ambiente a ser controlado, e para a temperatura será definido valor igual ao existente dentro do ambiente.

4.4.2 Pré-requisitos

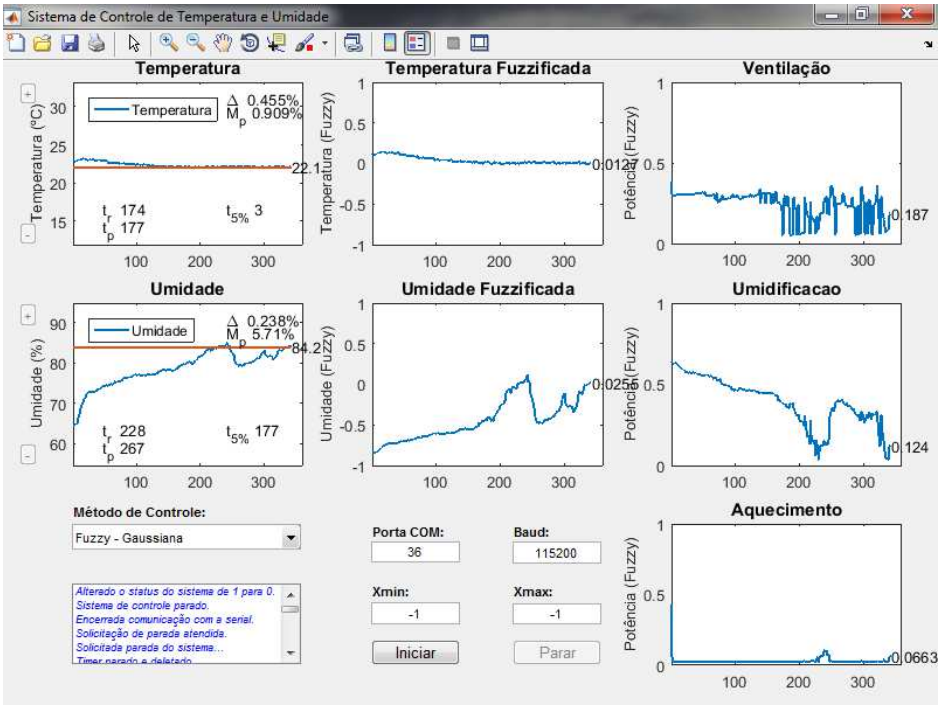
Com a umidade estável em 64%, configurou-se o sistema para atingir os 84%. Para a temperatura foi configurado *set point* igual ao valor já existente no início do teste, de 22 ºC.

4.4.3 Resultados

Conforme **Figura 4.4-1**, o sistema atingiu o valor de set point da umidade aos 228 segundos, com um sobressinal de 5,71% aos 267 segundos, e diferença de 5% aos 177.

Quanto à temperatura, obteve-se um sobressinal de 0,909%, mantendo-se praticamente estável por todo o tempo.

Figura 4.4-1 *- Gráficos gerados para o cenário de elevação da umidade em 20%.*



Fonte: Elaborado pelo autor.

4.5 Quarto cenário – reduzir umidade em 5%

4.5.1 Descrição

Este cenário de teste tem por objetivo reduzir a umidade para um valor de *set point* definido abaixo do valor existente dentro do ambiente a ser controlado, e para a temperatura será definido valor igual ao existente dentro do ambiente.

4.5.2 Pré-requisitos

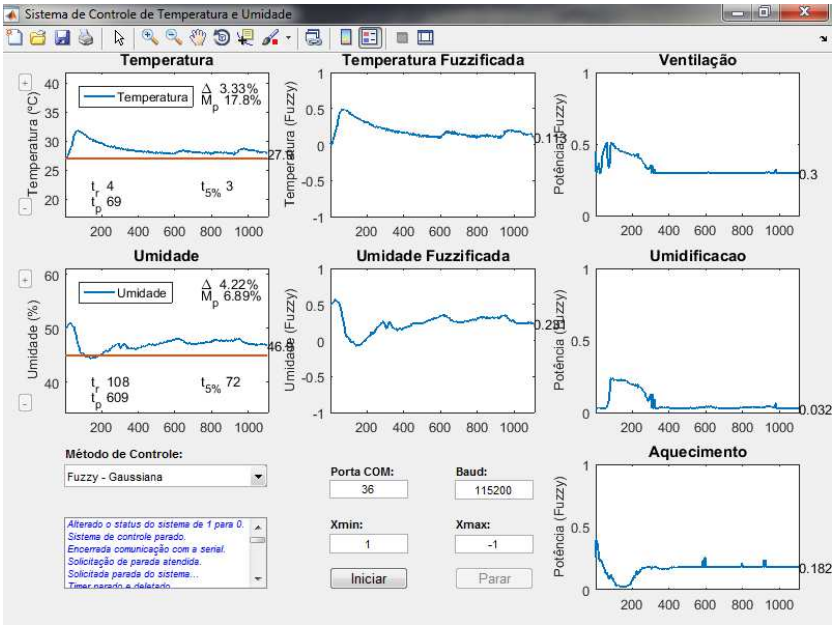
Com a umidade estável em 50%, configurou-se o sistema para atingir os 45%. Para a temperatura foi configurado *set point* igual ao valor já existente no início do teste, de 27 ºC.

4.5.3 Resultados

Conforme **Figura 4.5-1**, o sistema atingiu o valor de set point da umidade aos 108 segundos, com um sobressinal de 6,89% aos 609 segundos, e diferença de 5% aos 72. Porém, após atingir o *set point*, manteve seu percentual de umidade acima com uma diferença de aproximadamente 4,22%.

Quanto à temperatura, obteve-se um sobressinal de 17,8%, reduzindo seu valor gradativamente até estabilizar-se com uma diferença aproximada de 3,3% acima do *set point.*

***Figura 4.5-1** - Gráficos gerados para o cenário de redução da umidade em 5%.*



Fonte: Elaborado pelo autor.

4.6 Quinto cenário – reduzir umidade em 6%

4.6.1 Descrição

Este cenário de teste tem por objetivo reduzir a umidade para um valor de *set point* definido abaixo do valor existente dentro do ambiente a ser controlado, e para a temperatura será definido valor igual ao existente dentro do ambiente.

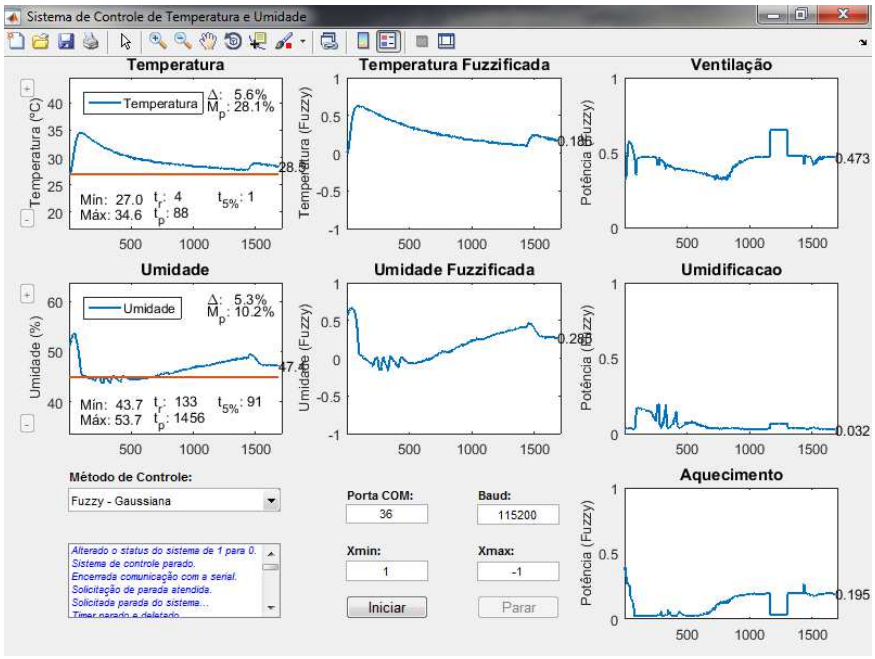
4.6.2 Pré-requisitos

Com a umidade estável em 51%, configurou-se o sistema para atingir os 45%. Para a temperatura foi configurado *set point* igual ao valor já existente no início do teste, de 27 ºC.

4.6.3 Resultados

Conforme **Figura 4.6-1**, o sistema atingiu o valor de set point da umidade aos 133 segundos, com um sobressinal de 10,2% aos 1456 segundos, e diferença de 5% aos 91. Porém, com a redução da temperatura, a umidade teve uma elevação, conforme se confirma no gráfico da umidade a partir de aproximadamente 750 segundos.

***Figura 4.6-1** - Gráficos gerados para o cenário de redução da umidade em 6%.*



Fonte: Elaborado pelo autor.

4.7 Sexto cenário – reduzir umidade em 10%

4.7.1 Descrição

Este cenário de teste tem por objetivo reduzir a umidade para um valor de *set point* definido abaixo do valor existente dentro do ambiente a ser controlado, e para a temperatura será definido valor igual ao existente dentro do ambiente.

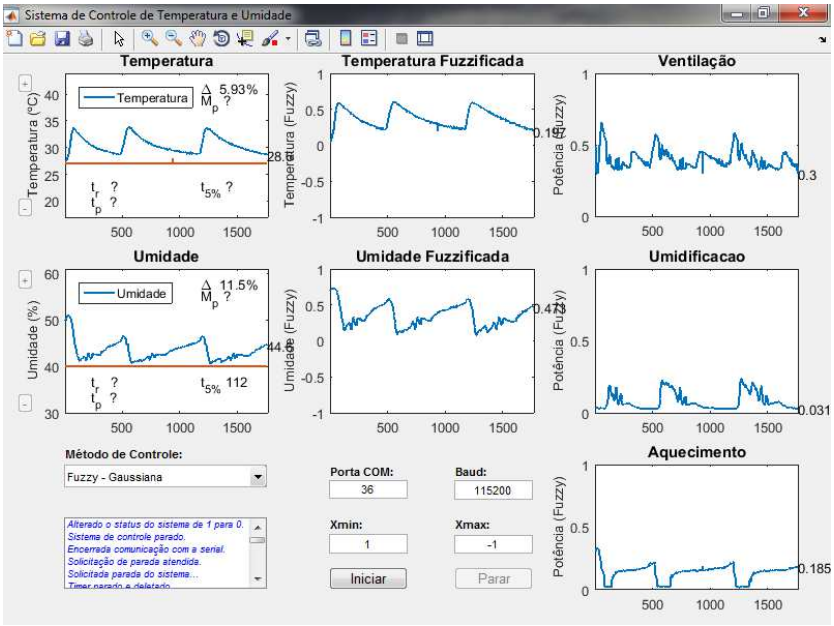
4.7.2 Pré-requisitos

Com a umidade estável em 50%, configurou-se o sistema para atingir os 40%. Para a temperatura foi configurado *set point* igual ao valor já existente no início do teste, de 27 ºC.

4.7.3 Resultados

Conforme **Figura 4.7-1**, o sistema passa a não conseguir estabilizar os valores das duas variáveis, pois conforme aproxima a temperatura de seu valor pretendido, a umidade aumenta o seu percentual. Da mesma forma, conforme a umidade se aproxima do *set point*, aumenta-se o valor da temperatura.

***Figura 4.7-1** - Gráficos gerados para o cenário de redução da umidade em 10%.*



Fonte: Elaborado pelo autor.

4.8 Sétimo cenário – elevar temperatura em 5 ºC

4.8.1 Descrição

Este cenário de teste tem por objetivo elevar a temperatura para um valor de *set point* definido acima do valor existente dentro do ambiente a ser controlado, e para a umidade será definido valor igual ao existente dentro do ambiente.

4.8.2 Pré-requisitos

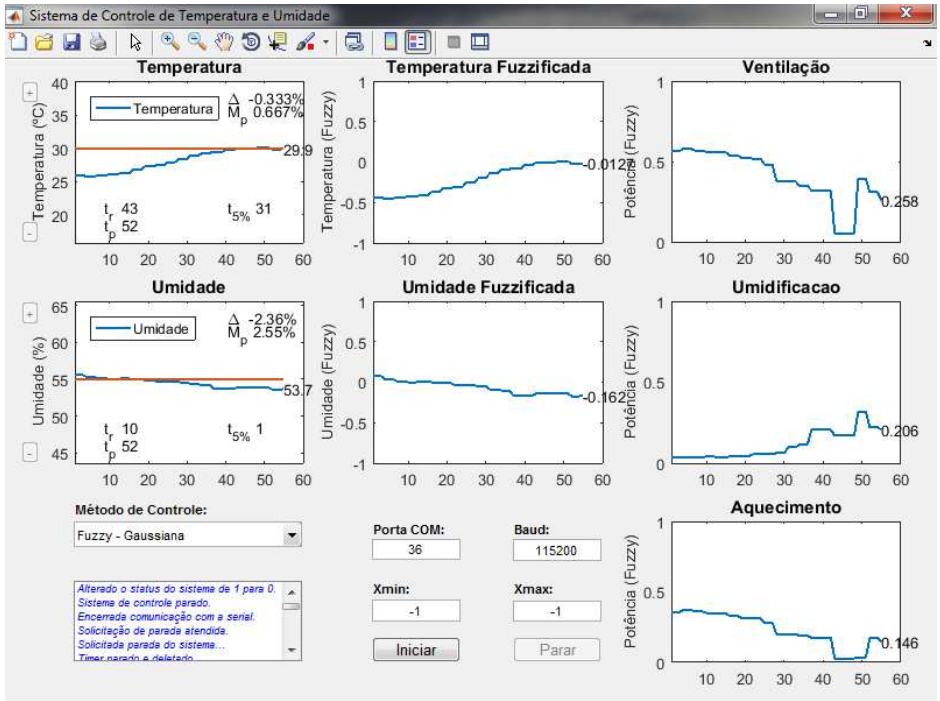
Com a temperatura estável em 25 ºC, configurou-se o sistema para atingir os 30 ºC. Para a umidade foi configurado *set point* igual ao valor já existente no início do teste, ou seja, 55%.

4.8.3 Resultados

Conforme **Figura 4.8-1 -** Gráficos gerados para o cenário de elevação de temperatura em 5 ºC.**Figura 4.8-1**, o sistema precisou de 43 segundos para atingir o valor de *set point* da temperatura, obteve um sobressinal de 0,667% aos 52 segundos, e atingiu o valor de 5% de diferença aos 31 segundos.

Quanto à umidade, obteve-se um sobressinal de 2,55% aos 52 segundos.

***Figura 4.8-1** - Gráficos gerados para o cenário de elevação de temperatura em 5 ºC.*



Fonte: Elaborado pelo autor.

4.9 Oitavo cenário – elevar temperatura em 9 ºC

4.9.1 Descrição

Este cenário de teste tem por objetivo elevar a temperatura para um valor de *set point* definido acima do valor existente dentro do ambiente a ser controlado, e para a umidade será definido valor igual ao existente dentro do ambiente.

4.9.2 Pré-requisitos

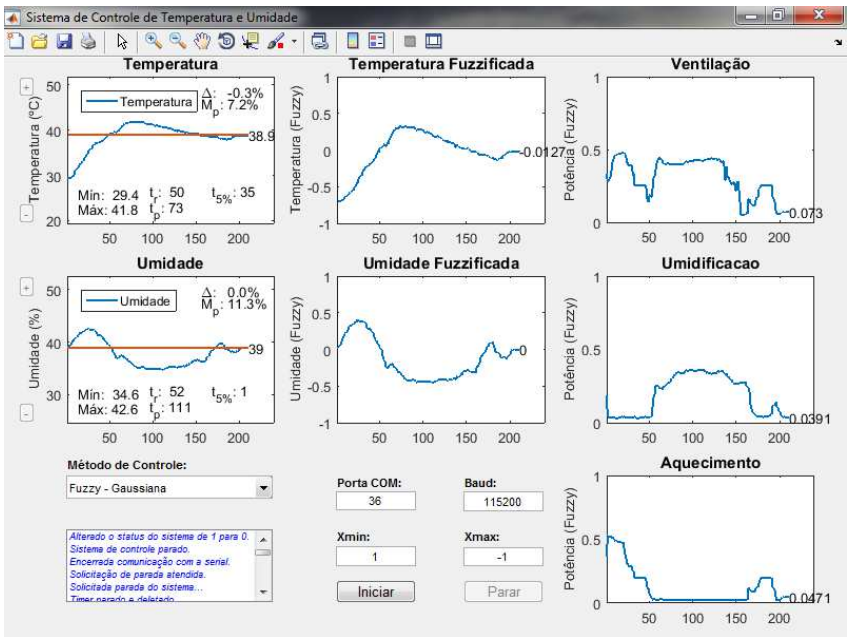
Com a temperatura estável em 29 ºC, configurou-se o sistema para atingir os 38 ºC. Para a umidade foi configurado *set point* igual ao valor já existente no início do teste, ou seja, 39%.

4.9.3 Resultados

Conforme **Figura 4.9-1**, o sistema precisou de 50 segundos para atingir o valor de *set point* da temperatura, com um sobressinal de 7,2% aos 73 segundos, e atingiu o valor de 5% de diferença aos 35 segundos.

A umidade obteve inicialmente um acréscimo até atingir 42,6 % de umidade relativa, após houve uma redução até 34,6 %, e por último estabilizou o valor em torno de 39%.

***Figura 4.9-1** - Gráficos gerados para o cenário de elevação de temperatura em 8 ºC.*



Fonte: Elaborado pelo autor.

4.10 Nono cenário – elevar temperatura em 10 ºC

4.10.1 Descrição

Este cenário de teste tem por objetivo elevar a temperatura para um valor de *set point* definido acima do valor existente dentro do ambiente a ser controlado, e para a umidade será definido valor igual ao existente dentro do ambiente.

4.10.2 Pré-requisitos

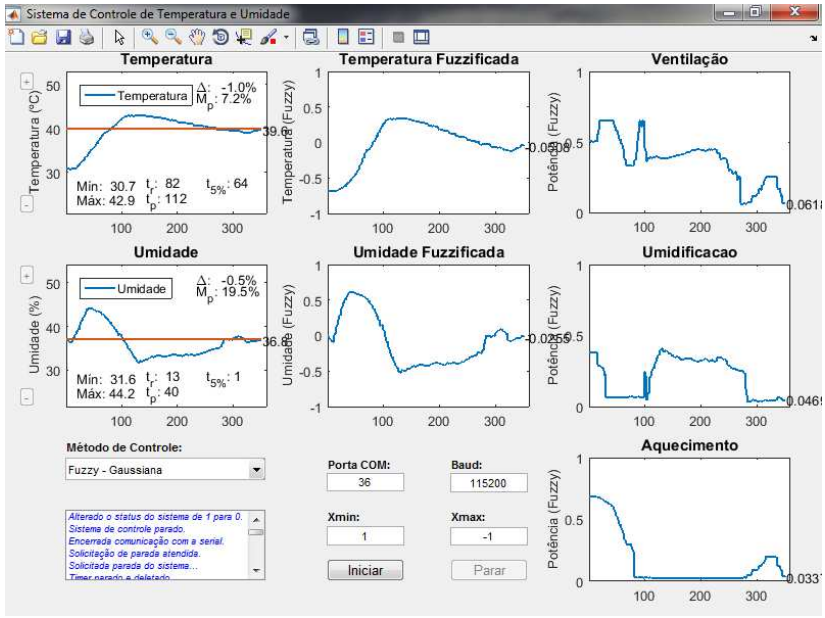
Com a temperatura estável em 30 ºC, configurou-se o sistema para atingir os 40 ºC. Para a umidade foi configurado *set point* igual ao valor já existente no início do teste de 37%.

4.10.3 Resultados

Conforme **Figura 4.10-1**, o sistema precisou de 82 segundos para atingir o valor de *set point* da temperatura, com um sobressinal de 7,2% aos 40 segundos, e atingiu o valor de 5% de diferença aos 64 segundos.

A umidade obteve inicialmente um acréscimo até atingir 44,2 % de umidade relativa, após houve uma redução para 31,6 %, e por último estabilizou o valor em torno de 37%.

Figura 4.10-1 *- Gráficos gerados para o cenário de elevação de temperatura em 10 ºC.*



Fonte: Elaborado pelo autor.

4.11 Décimo cenário – reduzir temperatura em 2 ºC

4.11.1 Descrição

Este cenário de teste tem por objetivo reduzir a temperatura para um valor de *set point* definido abaixo do valor existente dentro do ambiente a ser controlado, e para a umidade será definido valor igual ao existente dentro do ambiente.

4.11.2 Pré-requisitos

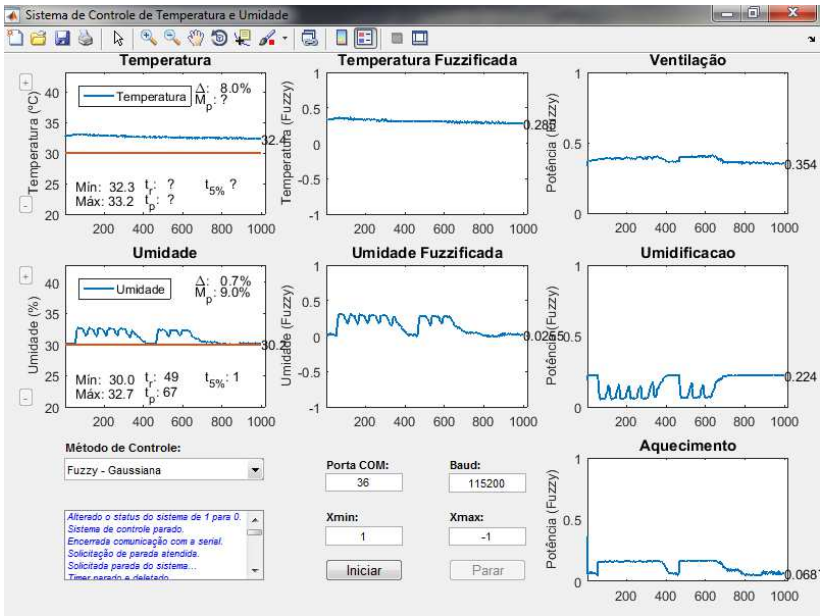
Com a temperatura estável em 32 ºC, configurou-se o sistema para atingir os 30 ºC. Para a umidade foi configurado *set point* igual ao valor já existente no início do teste, de 30%.

4.11.3 Resultados

Conforme **Figura 4.11-1**, mesmo após 1000 segundos de execução, o sistema foi incapaz de reduzir a temperatura, tendo variado de 32,3 ºC a 33,2 ºC, enquanto a umidade foi mantida relativamente próxima ao valor desejado, de 30%.

Essa incapacidade do sistema de controle em reduzir a temperatura deve-se à inexistência de atuador próprio para redução da temperatura.

***Figura 4.11-1** - Gráficos gerados para o cenário de redução da temperatura em 2 ºC.*



Fonte: Elaborado pelo autor.

4.12 Décimo primeiro cenário – elevar temperatura em 4 ºC e umidade em 5%

4.12.1 Descrição

Este cenário de teste tem por objetivo aumentar os valores de temperatura e umidade conforme os *set points* definidos.

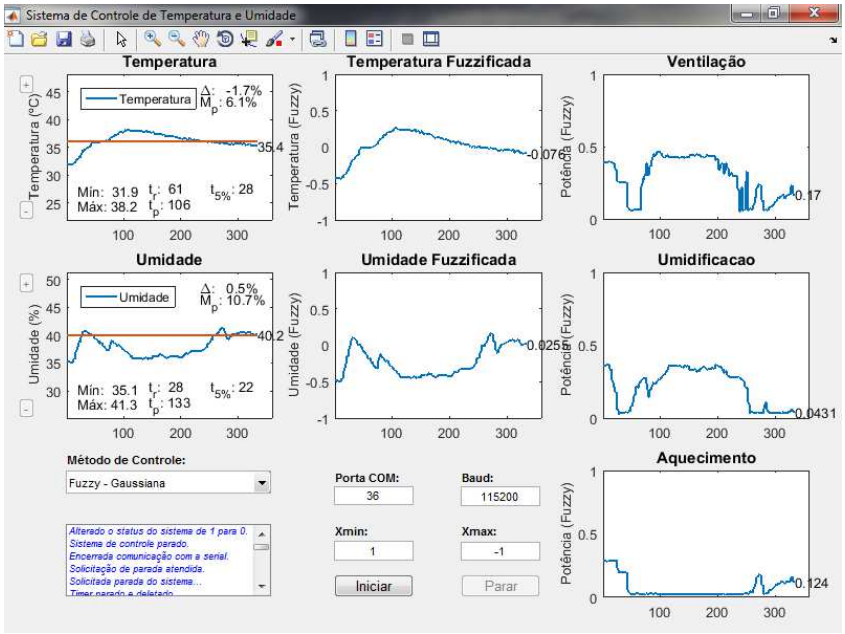
4.12.2 Pré-requisitos

Com a temperatura estável em 32 ºC, configurou-se o sistema para atingir os 36 ºC, e para uma umidade estável em 35%, configurou-se o sistema para atingir 40%.

4.12.3 Resultados

Conforme **Figura 4.12-1**, o sistema precisou de 61 segundos para atingir o valor de *set point* da temperatura e de 28 para a umidade, com um sobressinal de 6,1% aos 106 segundos para a temperatura e 10,7% aos 133 para a umidade, e uma diferença de 5% atingida aos 28 segundos para a temperatura e 22 para a umidade.

***Figura 4.12-1** - Gráficos para o cenário de aumento de temperatura em 4 ºC e umidade em 5%.*



Fonte: Elaborado pelo autor.

4.13 Décimo segundo cenário – elevar temperatura em 5 ºC e umidade em 10%

4.13.1 Descrição

Este cenário de teste tem por objetivo aumentar os valores de temperatura e umidade conforme os *set points* definidos.

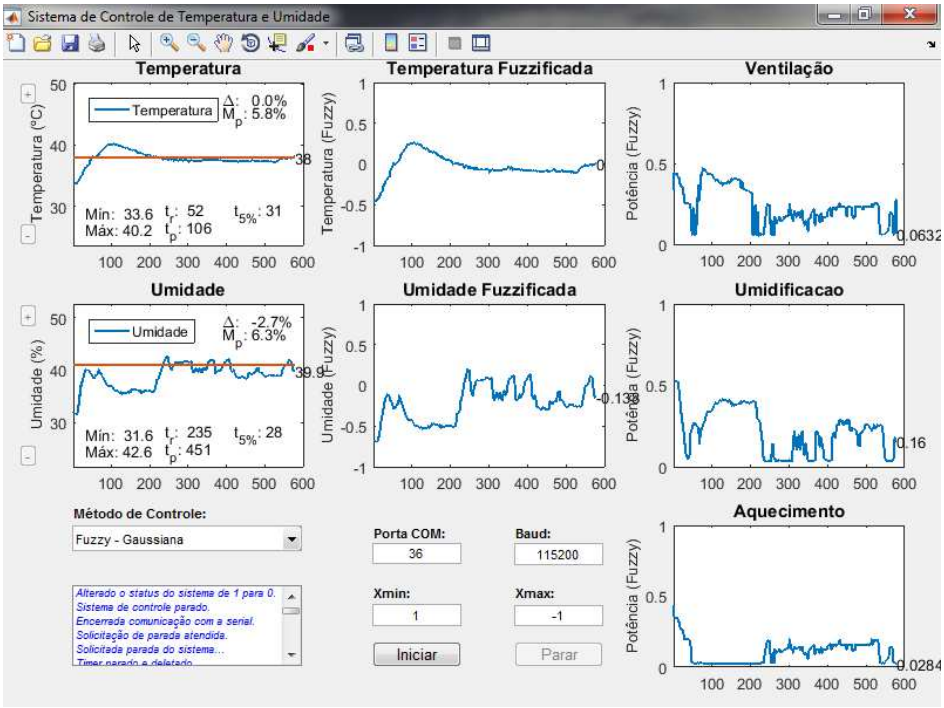
4.13.2 Pré-requisitos

Com a temperatura estável em 33 ºC, configurou-se o sistema para atingir os 38 ºC, e para uma umidade estável em 31%, configurou-se o sistema para atingir 41%.

4.13.3 Resultados

Conforme **Figura 4.13-1**, o sistema precisou de 52 segundos para atingir o valor de *set point* da temperatura e de 235 para a umidade, com um sobressinal de 5,8% aos 106 segundos para a temperatura e 6,3% aos 451 para a umidade, e uma diferença de 5% atingida aos 31 segundos para a temperatura e 28 para a umidade.

***Figura 4.13-1** - Gráficos para o cenário de aumento de temperatura em 5 ºC e umidade em 10%.*



Fonte: Elaborado pelo autor.

4.14 Décimo terceiro cenário – elevar temperatura em 7 ºC e umidade em 15%

4.14.1 Descrição

Este cenário de teste tem por objetivo aumentar os valores de temperatura e umidade conforme os *set points* definidos.

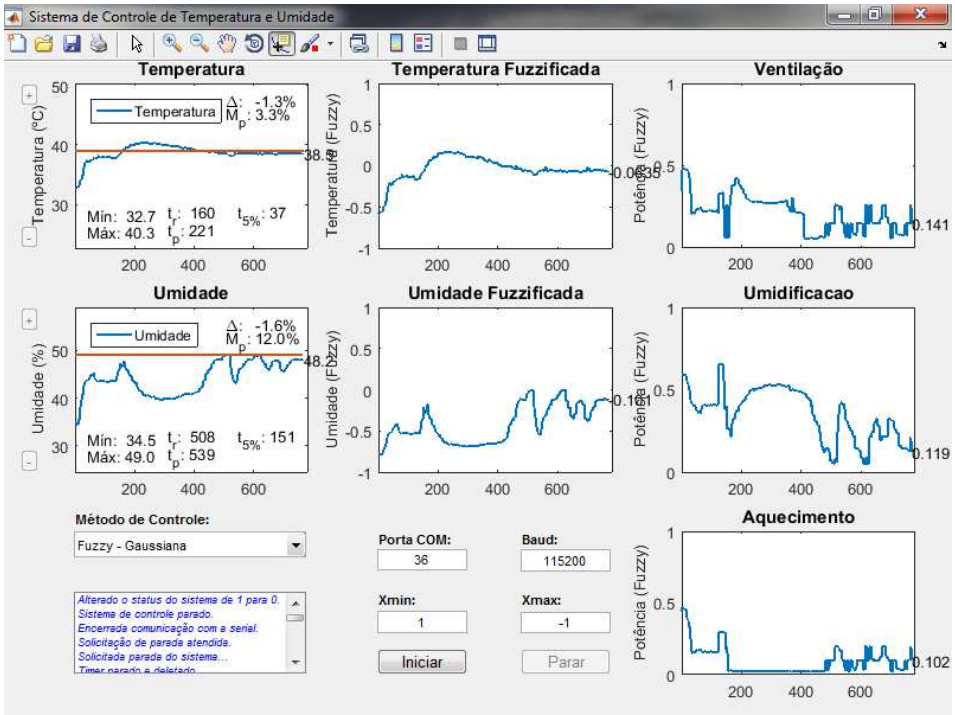
4.14.2 Pré-requisitos

Com a temperatura estável em 32 ºC, configurou-se o sistema para atingir os 39 ºC, e para uma umidade estável em 34%, configurou-se o sistema para atingir 49%.

4.14.3 Resultados

Conforme **Figura 4.14-1**, o sistema precisou de 160 segundos para atingir o valor de *set point* da temperatura e de 508 segundos para a umidade, com um sobressinal de 3,3% aos 221 segundos para a temperatura e 12,0% aos 539 para a umidade, e uma diferença de 5% atingida aos 37 segundos para a temperatura e 151 para a umidade.

***Figura 4.14-1** - Gráficos para o cenário de aumento de temperatura em 7 ºC e umidade em 15%.*



Fonte: Elaborado pelo autor.

4.15 Décimo quarto cenário – elevar temperatura em 3 ºC e reduzir umidade em 5%

4.15.1 Descrição

Este cenário de teste tem por objetivo aumentar a temperatura e reduzir a umidade conforme os *set points* definidos.

4.15.2 Pré-requisitos

Com a temperatura estável em 33 ºC, configurou-se o sistema para atingir os 36 ºC, e para uma umidade estável em 28%, configurou-se o sistema para atingir 23%.

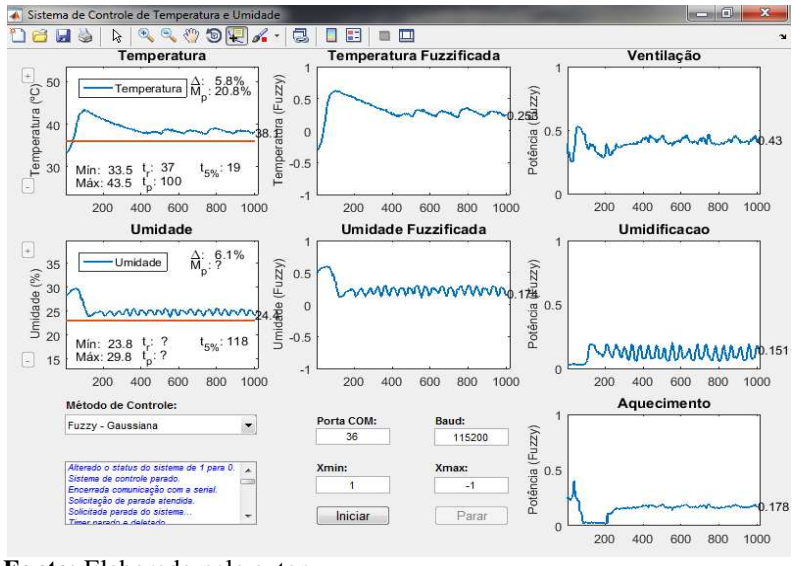
4.15.3 Resultados

Conforme **Figura 4.15-1**, o sistema precisou de 37 segundos para atingir o valor de *set point* da temperatura, não tendo atingido para a umidade. O sobressinal para a temperatura foi de 20,8% aos 100 segundos. A diferença de 5% foi atingida aos 19 segundos para a temperatura e 118 segundos para a umidade.

A temperatura atingiu o valor de estabilidade com aproximadamente 5% acima do valor pretendido, e a umidade ficou oscilando entre 4 e 6% acima do valor pretendido.

O insucesso em se atingir os valores de *set point* deve-se às influências entre as variáveis reguladas, pois há influência mútua entre elas.

***Figura 4.15-1** - Gráficos para aumento de temperatura em 3 ºC e redução da umidade em 5%.*



Fonte: Elaborado pelo autor.

4.16 Décimo quinto cenário – elevar temperatura em 5 ºC e reduzir umidade em 7%

4.16.1 Descrição

Este cenário de teste tem por objetivo aumentar a temperatura e reduzir a umidade conforme os *set points* definidos.

4.16.2 Pré-requisitos

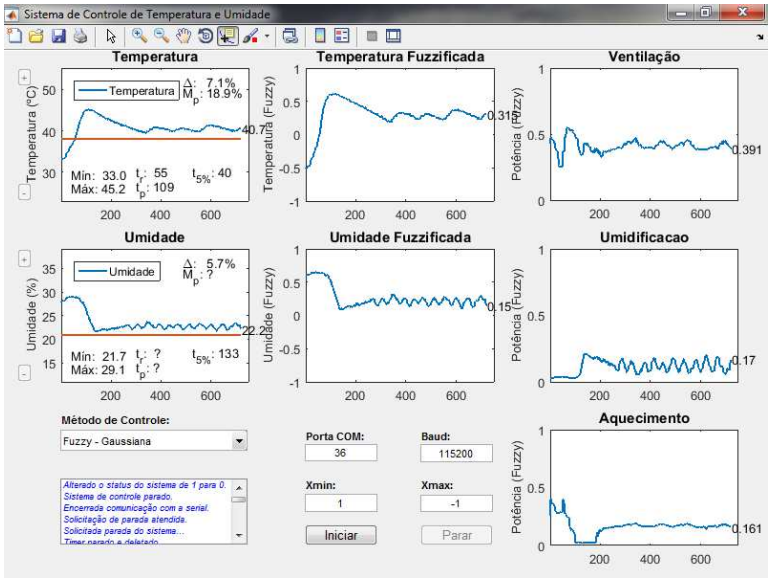
Com a temperatura estável em 33 ºC, configurou-se o sistema para atingir os 38 ºC, e para uma umidade estável em 28%, configurou-se o sistema para atingir 21%.

4.16.3 Resultados

Conforme **Figura 4.16-1**, o sistema precisou de 55 segundos para atingir o valor de *set point* da temperatura, não tendo atingido para a umidade. O sobressinal para a temperatura foi de 18,9% aos 109 segundos. A diferença de 5% foi atingida aos 40 segundos para a temperatura e 133 para a umidade.

A temperatura atingiu o valor de estabilidade com aproximadamente 7% acima do valor pretendido, e a umidade ficou oscilando entre 4 e 6% acima do valor pretendido.

Figura 4.16-1 *- Gráficos para aumento de temperatura em 5 ºC e redução da umidade em 7%.*



Fonte: Elaborado pelo autor.

4.17 Resumo dos cenários testados

A **Tabela 4.17-1** contém um resumo dos cenários de teste realizados.

Para facilitar o entendimento de cada cenário, na primeira coluna consta o número e dois símbolos que resumem o seu objetivo, onde "=" significa que a variável deveria ser mantida constante, "↑" que a variável deveria ter seu valor aumentado, e "↓" que a variável deveria ter seu valor reduzido. Dessa forma, no primeiro cenário contam os símbolos "=↑", ou seja, o objetivo desse cenário é manter constante o valor da temperatura e aumentar a umidade. Quanto à última coluna, foi informado "Sim" quando para o teste realizado foi possível obter o tempo de subida (t_r), ou seja, houve o atingimento do *set point*. Nas situações em que não se atingiu o *set point*, informou-se "Não".

Tabela 4.17-1 *- Resumo dos cenários de teste.*

	Temperatura (ºC)							**Umidade Relativa do Ar (%)**							**Ok ?**
	t_0	**t_f**	**Mín.**	**Máx.**	**M_p (%)**	**t_r (s)**	**$t_{5\%}$ (s)**	**U_0**	**U_f**	**Mín.**	**Máx.**	**M_p (%)**	**t_r (s)**	**$t_{5\%}$ (s)**	
1 =↑	25	25	-	-	0,8	148	3	55	60	-	-	3,3	190	37	Sim
2 =↑	23	23	-	-	0,9	110	3	60	70	-	-	2,3	129	74	Sim
3 =↑	22	22	-	-	0,9	174	3	64	84	-	-	5,71	228	177	Sim
4 =↓	27	27	-	-	17,8	4	3	50	45	-	-	6,9	108	72	Sim
5 =↓	27	27	-	-	28,1	4	1	51	45	-	-	10,2	133	91	Sim
6 =↓	27	27	-	-	?	?	112	50	40	-	-	?	?	?	Não
7 ↑=	25	30	-	-	0,7	43	31	55	55	-	-	2,6	10	1	Sim
8 ↑=	29	38	29,4	41,8	7,2	50	35	39	39	34,6	42,6	11,3	52	1	Sim
9 ↑=	30	40	30,7	42,9	7,2	82	64	37	37	31,6	44,2	19,5	13	1	Sim
10 ↓=	32	30	32,3	33,2	?	?	?	30	30	30,0	32,7	9,0	49	1	Não

11 ↑↑	32	36	31,9	38,2	6,1	61	28	35	40	35,1	41,3	10,7	28	22	Sim
12 ↑↑	33	38	33,6	40,2	5,8	52	31	31	41	31,6	42,6	6,3	235	28	Sim
13 ↑↑	32	39	32,7	40,3	3,3	160	37	34	49	34,5	49,0	12,0	508	151	Sim
14 ↑↓	33	36	33,5	43,5	20,8	37	19	28	23	23,8	29,8	?	?	118	Não
15 ↑↓	33	38	33,0	45,2	18,9	55	40	28	21	21,7	29,1	?	?	133	Não

Como se pode verificar pelos resultados obtidos, em 11 dos 15 cenários avaliados o valor de *set point* foi atingido pelo sistema.

Dos cenários em que houve falha no atingimento, o sexto demonstra claramente a interferência entre as duas variáveis controladas, pois pelo gráfico observa-se que quando há uma redução da temperatura, a umidade aumenta. E quando a temperatura aumenta, reduz-se a umidade.

No cenário 10 observa-se a incapacidade do sistema em reduzir a temperatura, pois se manteve constante ao longo do tempo, não tendo havido nenhuma redução significativa.

Nos cenários 14 e 15 há aumento da temperatura e diminuição da umidade, conforme esperado, porém mantem-se uma diferença em comparação com o valor desejado acima de 5%, tanto para temperatura, quanto para a umidade.

5 CONSIDERAÇÕES FINAIS

5.1 Conclusão

O cenário atual requer sistemas pensados a operarem com a máxima eficiência energética possível e que possam ser desenvolvidos no menor tempo possível. Dessa forma, a proposta deste trabalho tem relação direta com esses dois requisitos, pois ao mesmo tempo em que se pensa na otimização de recursos através de um sistema de controle, também objetiva-se reduzir o tempo de implementação, através do emprego da lógica *fuzzy*.

Para tanto, foi desenvolvido protótipo que permita testar o sistema de inferência *fuzzy* modelado através da *toolbox fuzzy*, o qual contém uma interface com o usuário e realiza comunicação com o MATLAB através de conexão via USB.

A metodologia empregada permitiu atingir os objetivos geral e específicos. O emprego da plataforma Arduino mostrou-se adequada devido à sua flexibilidade e facilidade de uso, e a utilização do MATLAB permitiu a modelagem do sistema de inferência *fuzzy* através de uma IDE amigável e de fácil utilização, além da facilidade na geração de gráficos diversos, mostrando-se desse modo ser uma ferramenta matemática poderosa.

A realização dos testes permitiu entender melhor o processo de modelagem utilizando lógica *fuzzy*, o qual exige do operador ajustes finos conforme melhor se entende o sistema a que se propõe controlar. Em diversos momentos houve a necessidade de se revisar o sistema de inferência *fuzzy* modelado, através da análise dos gráficos tridimensionais gerados pela *toolbox* e testando-se os valores calculados para as variáveis de saída conforme os valores especificados nas variáveis de entrada.

Os resultados dos testes evidenciaram os cenários onde o sistema melhor se comporta, assim como aqueles em que não se consegue atingir os valores de *set point* informados pelo usuário, definindo dessa forma o melhor domínio de atuação do sistema. Quanto a estes últimos, a falha no atingimento dos valores desejados deve-se a não existência de atuador específico para refrigeração do sistema.

E por último, o desenvolvimento do projeto mostrou-se altamente complexo, tendo em vista a necessidade de conversação entre os sistemas Arduino e MATLAB, além da criação de circuitos eletrônicos que permitissem o controle de potência entregue por cada um dos atuadores, os quais possuem funcionamento distintos, o que exigiu a tomada de estratégias individuais a cada um deles.

5.2 Propostas de Trabalhos Futuros

Durante o desenvolvimento do projeto proposto foram identificados diversos pontos de melhoria para futuros trabalhos.

O primeiro deles refere-se à elaboração de um conversor digital para analógico (DAC) que seja de baixo custo e eficiente. O DAC desenvolvido neste trabalho baseado no uso de amplificador operacional mostrou-se ser de baixo custo, porém possui limitações, pois não consegue estabilizar perfeitamente o sinal de saída de tal modo a retirar por completo os pulsos PWM do sinal de entrada, gerando dessa forma uma oscilação na saída.

A segunda proposta de melhoria refere-se aos ruídos gerados pelo relê existente no circuito do sistema de umidificação, o qual é responsável por desligar o atuador quando a potência a ser entregue seja muito baixa, evitando dessa forma uma operação não eficiente. O problema ocorre quando o relê ativa e desativa (altera entre os estados NC e NO) diversas vezes em um curto espaço de tempo, o qual atrapalha a apresentação das informações ao operador através dos *displays* da maquete, pois os mesmos passam a apresentar valores sem nenhum sentido. Para minimização do problema foi implementado nos circuitos um filtro passa-baixa passivo, porém o mesmo não inibe totalmente a passagem para dentro do circuito de ruídos. Portanto, sugere-se a elaboração de um filtro passa-baixa ativo.

A terceira proposta propõe a implementação de um sistema que permita escolher se o ar que passa pelo sistema de ventilação e aquecimento tenha como origem o ambiente externo ao ambiente de controle ou que venha do próprio ambiente que esteja sendo controlado, minimizando dessa forma as interferências do meio externo. Atualmente o ar sempre provêm do ambiente externo.

A quarta proposta é quanto ao acréscimo de mais duas variáveis de entrada para o sistema de inferência fuzzy, as quais passariam a informar se a temperatura e umidade estão tendo um acréscimo, decréscimo ou permanecem estáveis. Basicamente essas duas variáveis conteriam as derivadas primeira das curvas geradas pela temperatura e umidade, permitindo assim à lógica *fuzzy* realizar ajustes nos valores calculados para as variáveis de saída. O problema a ser resolvido é quanto ao acréscimo de regras a serem modeladas no fuzzy, pois poderiam passar das atuais 49 para milhares, devido ao seu aumento exponencial.

A quinta e última proposta refere-se ao uso da *toolbox* Simulink do MATLAB, a qual permite modelar sistema de controle através de uma interface gráfica, a qual pretendia ser utilizada, porém diversos problemas ocorreram e impossibilitaram seu uso. Dessa forma, há necessidade de resolver os problemas de integração com o Arduino e o MATLAB.

REFERÊNCIAS BIBLIOGRÁFICAS

AMENDOLA, Dra. Mariangela. et al. **Manual do uso da teoria dos conjuntos fuzzy no MATLAB 6.5.** 1. ed. - Campinas, SP: Unicamp/FEAGRI & IMECC, 2005.

ARDUINO. **Arduino.cc.** Acesso em 09 de outubro de 2016, disponível em: https://www.arduino.cc/en/Guide/Introduction.

BARROS, Laécio Carvalho de; BASSANEZI, Rodney Carlos. **Tópicos de lógica *fuzzy* e biomatemática.** 1. ed. - Campinas, SP: Unicamp/IMECC, 2006.

BILOBROVEC, Marcelo. **Sistema especialista em lógica *fuzzy* para o controle, gerenciamento e manutenção da qualidade em processo de aeração de grãos.** Dissertação (Mestrado) – Curso de Pós-Graduação em Engenharia de Produção. Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2005.

CAVALCANTE, Marcos Uchoa. et. al. **Controle de umidade e temperatura numa incubadora neonatal usando controle preditivo.** Bonito, MS: XVIII Congresso Brasileiro de Automática, 2010.

CAVALCANTI, José Homero Feitosa. et. al. **Lógica *fuzzy* aplicada às engenharias.** 1. ed. - João Pessoa, PB: s.n, 2012.

CIRCUITAR. **Esquema elétrico do Arduino. Guia definitivo.** Acesso em 17 de outubro de 2016, disponível em: https://www.circuitar.com.br/tutoriais/esquema-eletrico-do-arduino-o-guia-definitivo/.

FERNANDES, A. P. S. **Sistema Especialista Difuso de Apoio ao Aprendizado do traumatismo Dento-Alveolar utilizando Recursos Multimídia.** Dissertação de Mestrado. Universidade Federal de Santa Catarina, Florianópolis, 1997.

GILAT, Amos. **MATLAB com aplicações em engenharia.** 4. ed. - Porto Alegre: Bookman, 2012.

GONÇALVES, André Paim. **Aplicação de Lógica Fuzzy em Guerra Eletrônica.** Instituto Tecnológico da Aeronáutica, 2007.

KLIR, George Jiri; YUAN, Boo; CLAIR, Ute Saint. Fuzzy Set Theory: Foundations and Applications. United States: Prentice Hall, 1997.

MONK, Simon. **Projetos com arduino e android: use seu smartphone ou tablet para controlar o arduino;** tradução: Anatólio Laschuk. - Porto Alegre: Bookman, 2014.

MUKAIDONO, Mazao. **Fuzzy Logic For Beginners.** Singapore: World Scientific, 2001. RUSS, Eberhart; SIMPSON, Pat; DOBBINS, Roy. Computational Intelligence PC tool. London: AP Professional, 1996.

NICOLETTI, Maria do Carmo. et al. **Fundamentos da teoria de conjuntos *fuzzy*.** 1. ed. - São Carlos: Ed. UFSCar, 2013.

NILSSON, James W.; Riedel, Susan A. **Circuitos elétricos.** 6. ed. - Rio de janeiro: LTC, 2003.

OGATA, Katsuhiko. **Engenharia de controle moderno.** 4. ed. - São Paulo: Prentice Hall, 2003.

ORTEGA, Neli Regina Siqueira. **Aplicação da Teoria de Conjuntos *Fuzzy* a Problemas da Biomedicina.** Dissertação (Doutorado) – Instituto de Física. Universidade de São Paulo. São Paulo, 2001.

PALM, William J. **Introdução ao MATLAB para engenheiros.** 3. ed. – Porto Alegre: AMGH, 2013.

RASHIDI, Farzan. et al. ***A hybrid fuzzy logic and PID controller for control of nonlinear HVAC systems*.** IEEE, 2003.

RESNIK, Leonid. ***Fuzzy Controllers*.** Newnes, Reino Unido, 1997.

RIGNEL, Diego Gabriel de Souza. et al. **Uma introdução à lógica *Fuzzy*.** Revista RESIGeT, 2011.

ROMANO, Vitor Ferreira. **Robótica industrial – Aplicação na indústria de manufatura de processos.** 1. ed. - São Paulo: Editora Edgar Blucher LTDA, 2002.

SANTOS, Vitor Rodrigues. **Carretilha automatizada para pesca esportiva pesada e de espera.** Dissertação de graduação. Centro Universitário UniCEUB: Brasília, 2016.

SILVA, Renato Afonso Cota. **Inteligência artificial aplicada a ambientes de engenharia de software: uma visão geral.** Universidade Federal de Viçosa, 2005.

SIMÕES, Marcelo Godoy; Ian S. Shaw. **Controle e modelagem *fuzzy*.** - São Paulo: Blucher: FAPESP, 2007.

APÊNDICES

1 MATLAB

1.1 *Arquivo* gui.m

O programa a seguir é executado pela Central de Processamento no MATLAB, sendo o responsável por obter e enviar dados ao ARDUINO e à *toolbox Fuzzy*, tendo sido desenvolvido através da ferramenta GUIDE do MATLAB.

```
<code>
function varargout = gui (varargin)
% GUI_VRS01 MATLAB code for gui_vrs01.fig
%
% Begin initialization code - DO NOT EDIT
%
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_vrs01_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_vrs01_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui_vrs01 is made visible.
function gui_vrs01_OpeningFcn(hObject, eventdata, handles, varargin)
    % Choose default command line output for gui_vrs01
    handles.output = hObject;

    % Update handles structure
    guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
```

```
function varargout = gui_vrs01_OutputFcn(hObject, eventdata, handles)
    varargout{1} = handles.output;


function figure1_CreateFcn(hObject, eventdata, handles)
    % Inicializa as variaveis de trabalho.
    handles.i                           = 1;
    handles.status                      = 0;   % 0: Stop, 1: Start, 2: Pause
    handles.indiceMetodoControle        = 2;

    guidata( hObject, handles );


function axesTemperatura_CreateFcn(hObject, eventdata, handles)
    title(  'Temperatura' );
    ylabel( 'Temperatura (ºC)', 'FontSize', 10 );


function axesUmidade_CreateFcn(hObject, eventdata, handles)
    title(  'Umidade' );
    ylabel( 'Umidade (%)', 'FontSize', 10 );


function axesTemperaturaFuzzificada_CreateFcn(hObject, eventdata, handles)
    title(  'Temperatura Fuzzificada' );
    ylabel( 'Temperatura (Fuzzy)', 'FontSize', 10 );


function axesUmidadeFuzzificada_CreateFcn(hObject, eventdata, handles)
    title(  'Umidade Fuzzificada' );
    ylabel( 'Umidade Relativa (Fuzzy)', 'FontSize', 10 );


function axesVentilacao_CreateFcn(hObject, eventdata, handles)
    title(  'Ventilação' );
    ylabel( 'Potência (Fuzzy)', 'FontSize', 10 );


function axesUmidificacao_CreateFcn(hObject, eventdata, handles)
    title(  'Umidificação' );
    ylabel( 'Potência (Fuzzy)', 'FontSize', 10 );


function axesAquecimento_CreateFcn(hObject, eventdata, handles)
    title(  'Aquecimento' );
    ylabel( 'Potência (Fuzzy)', 'FontSize', 10 );


function popupmenuMetodoControle_Callback(hObject, eventdata, handles)
    disp( 'popupmenuMetodoControle_Callback()...' );
    itemSelecionado = get( hObject, 'Value' );
```

```
    setMetodoControleSelecionado( hObject, itemSelecionado );


function popupmenuMetodoControle_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end


function togglebuttonStartPause_Callback(hObject, eventdata, handles)
    disp( 'togglebuttonStartPause_Callback()...' );

    statusSistema = get( hObject, 'Value' );
    if( statusSistema == 0 )
        setStatusSistema( hObject, handles, 2 );
    else
        setStatusSistema( hObject, handles, 1 );
    end


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over togglebuttonStartPause.
function togglebuttonStartPause_ButtonDownFcn(hObject, eventdata, handles)
    disp( 'togglebuttonStartPause_ButtonDownFcn()...' );


function togglebuttonStartPause_CreateFcn(hObject, eventdata, handles)


function pushbuttonStop_Callback(hObject, eventdata, handles)
    disp( 'togglebuttonStop_Callback()...' );

    setStatusSistema( hObject, handles, 0 );


function textMsg_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end


function editPortaCOM_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
```

```
function editBaud_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end


% Hint: get(hObject,'Value') returns toggle state of checkboxIsZoom
function setMetodoControleSelecionado( hObject, cdMetodoControle )
    theHandlesStructure = guidata( hObject );
    indiceMetodoAtual = theHandlesStructure.indiceMetodoControle;
    theHandlesStructure.indiceMetodoControle = cdMetodoControle;

    msg = [ 'Alterando do metodo ', num2str(indiceMetodoAtual), ' para o ',
num2str( cdMetodoControle ), '.' ];
    setMsgSistema( hObject, msg );

    guidata( hObject, theHandlesStructure );


function setStatusSistema( hObject, handles, cdNovoStatus )
    disp( 'setStatusSistema()...' );

    theHandlesStructure = guidata( hObject );

    switch cdNovoStatus
        case 0  %Parar
            setMsgSistema( hObject, 'Parar sistema!' );
            set( handles.togglebuttonStartPause, 'String', 'Iniciar', 'Value', 0 );
            set( handles.pushbuttonStop        , 'Enable', 'off' );

            set( handles.pushbuttonIncrSpTemp  , 'Enable', 'off' );
            set( handles.pushbuttonDecrSpTemp  , 'Enable', 'off' );
            set( handles.pushbuttonIncrSpUmid  , 'Enable', 'off' );
            set( handles.pushbuttonDecrSpUmid  , 'Enable', 'off' );

            stopSistemaControle( hObject, theHandlesStructure );

        case 1  %Iniciar/Reiniciar
            setMsgSistema( hObject, 'Iniciar/reiniciar sistema!' );
            set( handles.togglebuttonStartPause, 'String', 'Pausar' );
            set( handles.pushbuttonStop        , 'Enable', 'on' );

            set( handles.pushbuttonIncrSpTemp  , 'Enable', 'on' );
            set( handles.pushbuttonDecrSpTemp  , 'Enable', 'on' );
            set( handles.pushbuttonIncrSpUmid  , 'Enable', 'on' );
            set( handles.pushbuttonDecrSpUmid  , 'Enable', 'on' );

            if( isfield( theHandlesStructure, 'timerCallSerial' ) == 1 )
                isTimerValidLocal = isvalid( theHandlesStructure.timerCallSerial );
            else
```

```
                isTimerValidLocal = 0;
            end

            if( isTimerValidLocal == 0 )
                [serialCriada, timerCriado, theHandlesStructure] =
startSistemaControle( hObject, theHandlesStructure );
                theHandlesStructure.serial = serialCriada;
                theHandlesStructure.timerCallSerial = timerCriado;
            end

        case 2  %Pausar
            setMsgSistema( hObject, 'Pausar sistema!' );
            set( handles.togglebuttonStartPause, 'String', 'Reiniciar' );
            set( handles.pushbuttonStop        , 'Enable', 'on' );
    end

    msg = [ 'Alterado o status do sistema de ', num2str(theHandlesStructure.status), ' para ',
num2str(cdNovoStatus), '.' ];
    setMsgSistema( hObject, msg );

    theHandlesStructure.status = cdNovoStatus;

    guidata( hObject, theHandlesStructure );


function setMsgSistema( hObject, msg )
    theHandlesStructure = guidata( hObject );

    currString = get( theHandlesStructure.textMsg, 'String' );

    if iscell( currString ) == 0 && ischar( currString ) == 1
        currString = cellstr( currString );
    end

    currString{ end + 1 } = msg;
    currString = circshift( currString, [1,0] );

    set( theHandlesStructure.textMsg, 'String', currString );


function [serial, t, handles] = startSistemaControle( hObject, handles )
    % Inicializa as variaveis que armazenam os dados historicos.
    handles.x                           = [];
    handles.temperaturas                = [];
    handles.umidades                    = [];
    handles.sPointTemperaturas          = [];
    handles.sPointUmidades              = [];
    handles.temperaturasFuzzificadas    = [];
    handles.umidadesFuzzificadas        = [];
    handles.handles.potenciasVentilacao = [];
    handles.potenciasVentilacao         = [];
```

```
handles.potenciasUmidificacao        = [];
handles.potenciasAquecimento         = [];

% Inicializa as variaveis de trabalho.
handles.i                            = 1;

%    Inicializa as variáveis que registram os parâmetros de controle.
handles.xUltAltSpTemp                = 0;
handles.xUltAltSpUmid                = 0;
handles.isAtingidoSpTemp             = 0;
handles.isAtingidoSpUmid             = 0;
handles.xTrTempAnt                   = 0;
handles.xTrTempUlt                   = 0;
handles.xTrUmidAnt                   = 0;
handles.xTrUmidUlt                   = 0;
handles.xMpTemp                      = 0;
handles.xMpUmid                      = 0;
handles.xT5Temp                      = 0;
handles.xT5Umid                      = 0;


%Carrega os sistemas de inferência fuzzy.
metodoControleFuzzyTrap  = 'fuzzy_contr_temp_umid_vrs03';
metodoControleFuzzyGauss = 'fuzzy_contr_temp_umid_umid_gauss_vrs01';

setMsgSistema( hObject, 'Carregando sistema de inferência fuzzy trapezoidal...' );
fismatTrap = readfis( metodoControleFuzzyTrap );
disp( 'Modelo Fuzzy Trapezoidal:' );
getfis( fismatTrap )
setMsgSistema( hObject, 'Sistema fuzzy trapezoidal carregado com sucesso.' );
handles.fismatTrap = fismatTrap;

setMsgSistema( hObject, 'Carregando sistema de inferência fuzzy gaussiana...' );
fismatGauss = readfis( metodoControleFuzzyGauss );
disp( 'Modelo Fuzzy Gaussiana:' );
getfis( fismatGauss )
setMsgSistema( hObject, 'Sistema fuzzy gaussiana carregado com sucesso.' );
handles.fismatGauss = fismatGauss;


%Obtem os valores da porta e baud da comunicação via serial.
portaCOM = get( handles.editPortaCOM, 'String' )
baud     = get( handles.editBaud    , 'String' )

%Estabelece comunicação com a Serial.
serial = getConnectionSerialFcn( str2num( portaCOM ), str2num( baud ) )
msg = [ 'Iniciada comunicação via serial pela porta ', portaCOM, ' a ', baud, ' baud.' ];
setMsgSistema( hObject, msg );


%Cria o timer que será responsavel por se comunicar com a Serial.
```

```
    t = timer( 'Name', 'timerCallSerial', 'ExecutionMode', 'fixedSpacing' );
    t.Period     = 0.2;
    t.StartDelay = 0.5;
    t.TimerFcn = {@TimerCallback, hObject, handles};

    start( t );

    setMsgSistema( hObject, 'Sistema de controle iniciado.' );

    return


function stopSistemaControle( hObject, handles )
    timerSerial = handles.timerCallSerial;
    stop( timerSerial );
    delete( timerSerial );
    setMsgSistema( hObject, 'Timer parado e deletado.' );

    serialLocal = handles.serial;

    setMsgSistema( hObject, 'Solicitada parada do sistema...' );
    if( stopSistemaFcn( serialLocal ) == true )
        setMsgSistema( hObject, 'Solicitação de parada atendida.' );
    else
        setMsgSistema( hObject, 'Solicitação de parada NÃO foi atendida!' );
    end

    closeConnectionSerialFcn( serialLocal );
    setMsgSistema( hObject, 'Encerrada comunicação com a serial.' );

    setMsgSistema( hObject, 'Sistema de controle parado.' );


function TimerCallback( obj, event, hObject, handles )
    theHandlesStructure = guidata( hObject );

    %Obtem o status atual do sistema.
    statusSistema = theHandlesStructure.status;

    i    = theHandlesStructure.i;
    if( i <= 1 ); iAnt = 1; else iAnt = i - 1; end

    [temp, umid, spTemp, spUmid, isStop]  = realizarLeituras( theHandlesStructure );
    [tempFuzz, umidFuzz]                  = fuzzificacao( temp, spTemp, umid, spUmid );
    [potVent   , potUmid   , potAquec]    = calcularPotencias( theHandlesStructure, tempFuzz,
umidFuzz );
    [potVentAnt, potUmidAnt, potAquecAnt] = getPotenciasAnteriores( theHandlesStructure, i );

    % Armazena os dados em array.
    theHandlesStructure.x(i)                        = theHandlesStructure.i;
    theHandlesStructure.temperaturas(i)             = temp;
```

```
    theHandlesStructure.sPointTemperaturas(i)       = spTemp;
    theHandlesStructure.umidades(i)                 = umid;
    theHandlesStructure.sPointUmidades(i)           = spUmid;
    theHandlesStructure.temperaturasFuzzificadas(i) = tempFuzz;
    theHandlesStructure.umidadesFuzzificadas(i)     = umidFuzz;
    theHandlesStructure.potenciasVentilacao(i)      = potVent;
    theHandlesStructure.potenciasUmidificacao(i)    = potUmid;
    theHandlesStructure.potenciasAquecimento(i)     = potAquec;

    % Verifica se houve alteração de set point;
    if( theHandlesStructure.sPointTemperaturas(i) ~=
theHandlesStructure.sPointTemperaturas(iAnt) )
        handles.xUltAltSpTemp = i;
    end
    if( theHandlesStructure.sPointUmidades(i) ~= theHandlesStructure.sPointUmidades(iAnt) )
        handles.xUltAltSpUmid = i;
    end

    % Verifica se houve atingimento dos set points.
    [isAtingidoSpTemp, isAtingidoSpUmid, xTrTempAnt, xTrTempUlt, xTrUmidAnt, xTrUmidUlt] =
getAtingidoSp( theHandlesStructure );
    theHandlesStructure.isAtingidoSpTemp = isAtingidoSpTemp;
    theHandlesStructure.isAtingidoSpUmid = isAtingidoSpUmid;
    theHandlesStructure.xTrTempAnt = xTrTempAnt;
    theHandlesStructure.xTrTempUlt = xTrTempUlt;
    theHandlesStructure.xTrUmidAnt = xTrUmidAnt;
    theHandlesStructure.xTrUmidUlt = xTrUmidUlt;

    % Analisa se houve mudança do máximo sobre-sinal (Mp).
    [xMpTemp, xMpUmid] = getMp( theHandlesStructure );
    theHandlesStructure.xMpTemp = xMpTemp;
    theHandlesStructure.xMpUmid = xMpUmid;

    % Verifica atingimento de acomodação.
    [xT5Temp, xT5Umid] = getAcomodacao( theHandlesStructure );
    theHandlesStructure.xT5Temp = xT5Temp;
    theHandlesStructure.xT5Umid = xT5Umid;

    % Plota os graficos.
    if( statusSistema ~= 2 )
        plotTemperatura(  theHandlesStructure );
        plotUmidade(      theHandlesStructure );
        plotTempFuzz(     theHandlesStructure );
        plotUmidFuzz(     theHandlesStructure );
        plotVentilacao(   theHandlesStructure );
        plotUmidificacao( theHandlesStructure );
        plotAquecimento(  theHandlesStructure );

        configAxisAxes( theHandlesStructure );
        configTextAxes( theHandlesStructure );
```

```
        disp( [num2str( i ), ': ', event.Type ' executed ', datestr(event.Data.time,'dd-mmm-
yyyy HH:MM:SS.FFF')] );
    end

    if( isStop == 1 )
        % Envia valores de potencia ZERADOS p/ que pare todos os atuadores.
        sendPotenciasFcn( theHandlesStructure.serial, -1, 0, -1, 0, -1, 0 );

        setMsgSistema( hObject, 'Pressionado o botão STOP pela maquete.' );
        %setStatusSistema( hObject, handles, 0 );

        handleWindow   = findall( 0, 'Tag', 'figure1' );
        handleButton   = findobj( handleWindow, 'Tag', 'pushbuttonStop' );
        callbackButton = get( handleButton, 'Callback');
        callbackButton( handleWindow, [] );
    end

    % Envia os valores de potencia
    sendPotenciasFcn( theHandlesStructure.serial, potVentAnt, potVent, potUmidAnt, potUmid,
potAquecAnt, potAquec );



    theHandlesStructure.i = i + 1;

    guidata( hObject, theHandlesStructure );


function [xMin, xMax] = getXExtremos( handles, i, isValid )
    %Obtem os valores informados para o xMin e xMax.
    xMinInfo = get( handles.editXMin, 'String' );
    xMaxInfo = get( handles.editXMax, 'String' );

    [xMinInfoNum, xMinStatus] = str2num( xMinInfo );
    if( xMinStatus == 0 )
        xMinInfoNum = 0;
    end

    [xMaxInfoNum, xMaxStatus] = str2num( xMaxInfo );
    if( xMaxStatus == 0 )
        xMaxInfoNum = 0;
    end

    % Define valores minimo e maximo do eixo horizontal dos graficos.
    if( xMinInfoNum > 0 )
        xMin = xMinInfoNum;
    else
        if( i <= 400 )
            xMin = 1;
        else
            xMin = i - 400;
        end
```

```
    end

    if( xMaxInfoNum > 0 )
        xMax = xMaxInfoNum;
    else
        xMax = ( fix( i / 30 ) + 1 ) * 30;
    end

    % Se informado "true" no parâmetro isValid, retorna um valor para xMax
    %que esteja limitado ao tamanho dos arrays que armazenam os dados.
    if( isValid == true )
        i     = size( handles.temperaturas, 2 );

        if( xMax > i ); xMax = i; end
    end


% Realiza as leituras de temperatura e umidade, inclusive set points.
function [temp, umid, spTemp, spUmid, isStop] = realizarLeituras( handles )
    serialLocal = handles.serial;

    spTemp = readSetPointTemperaturaFcn( serialLocal );
    spUmid = readSetPointUmidadeFcn( serialLocal );
    temp   = readTemperaturaFcn( serialLocal );
    umid   = readUmidadeFcn( serialLocal );
    isStop = readStatusStopFcn( serialLocal );


% Fuzzificacao dos valores lidos.
function [ tempFuzz, umidFuzz ] = fuzzificacao( temperatura, spTemperatura, umidade,
spUmidade )
    tempFuzz = atan( ( ( temperatura - spTemperatura ) / 5 ) ) / ( pi / 2 );
    umidFuzz = atan( ( ( umidade     - spUmidade     ) / 5 ) ) / ( pi / 2 );


%Calcula as potências conforme o método de controle selecionado.
function [potVent, potUmid, potAquec] = calcularPotencias( handles, tempFuzz, umidFuzz )
    metodoControle = handles.indiceMetodoControle;

    if( metodoControle ~= 3 )
        if( metodoControle == 1 )
            fismat = handles.fismatTrap;
        else
            fismat = handles.fismatGauss;
        end

        % Aciona a logica fuzzy.
        out = evalfis( [tempFuzz umidFuzz], fismat );

        % Obtem os valores retornados pela logica fuzzy.
        potVent  = out(1, 1);
```

```
        potUmid  = out(1, 2);
        potAquec = out(1, 3);
    else
        potVent  = 0.0;
        potUmid  = 0.0;
        potAquec = 0.0;

        if( tempFuzz < 0.0 )
            potVent  = 0.9;
            potAquec = 0.5;
        else
            potAquec = 0.0;
        end

        if( umidFuzz < 0.0 )
            potUmid = 0.9;
        else
            potVent  = 0.9;
            potUmid = 0.0;
        end
    end


%Obtem os valores anteriores de potencia.
function [potVentAnt, potUmidAnt, potAquecAnt] = getPotenciasAnteriores( handles, i )
    if( i > 1 ); potVentAnt  = handles.potenciasVentilacao(i - 1);   else potVentAnt  = 0; end
    if( i > 1 ); potUmidAnt  = handles.potenciasUmidificacao(i - 1); else potUmidAnt  = 0; end
    if( i > 1 ); potAquecAnt = handles.potenciasAquecimento(i - 1);  else potAquecAnt = 0; end


function [isAtingidoSpTemp, isAtingidoSpUmid, xTrTempAnt, xTrTempUlt, xTrUmidAnt, xTrUmidUlt]
= getAtingidoSp( handles )
    i     = size( handles.temperaturas, 2 );

    if( i > 2 )
        iAnt = i - 1;

        spTempAtual      = handles.sPointTemperaturas(i);
        spTempAnt        = handles.sPointTemperaturas(iAnt);
        xTrTempAnt       = handles.xTrTempAnt;
        xTrTempUlt       = handles.xTrTempUlt;

        if( spTempAtual ~= spTempAnt )
            isAtingidoSpTemp = 0;

            xTrTempAnt = i;
            xTrTempUlt = 0;
        else
            if( handles.isAtingidoSpTemp == 0 )
                tempAtual   = handles.temperaturas(i);
                tempAnt     = handles.temperaturas(iAnt);
```

```
            varAtual    = tempAtual - spTempAtual;
            varAnt      = tempAnt   - spTempAtual;

            if( varAtual == 0 | varAnt == 0 | (varAtual / varAnt) < 0 )
                isAtingidoSpTemp = 1;

                xTrTempUlt = i;
            else
                isAtingidoSpTemp = 0;

                xTrTempUlt = 0;
            end
        else
            isAtingidoSpTemp = 1;
        end
    end

    spUmidAtual      = handles.sPointUmidades(i);
    spUmidAnt        = handles.sPointUmidades(iAnt);
    xTrUmidAnt       = handles.xTrUmidAnt;
    xTrUmidUlt       = handles.xTrUmidUlt;

    if( spUmidAtual ~= spUmidAnt )
        isAtingidoSpUmid = 0;

        xTrUmidAnt = i;
        xTrUmidUlt = 0;
    else
        if( handles.isAtingidoSpUmid == 0 )
            umidAtual   = handles.umidades(i);
            umidAnt     = handles.umidades(iAnt);

            varAtual    = umidAtual - spUmidAtual;
            varAnt      = umidAnt   - spUmidAtual;

            if( varAtual == 0 | varAnt == 0 | (varAtual / varAnt) < 0 )
                isAtingidoSpUmid = 1;

                xTrUmidUlt = i;
            else
                isAtingidoSpUmid = 0;

                xTrUmidUlt = 0;
            end
        else
            isAtingidoSpUmid = 1;
        end
    end
else
    isAtingidoSpTemp = 0;
```

```
        isAtingidoSpUmid = 0;

        xTrTempAnt = 0;
        xTrTempUlt = 0;
        xTrUmidAnt = 0;
        xTrUmidUlt = 0;
    end


function [xMpTemp, xMpUmid] = getMp( handles )
    i     = size( handles.temperaturas, 2 );
    if( i <= 1 ); iAnt = i; else iAnt = i - 1; end

    if( handles.isAtingidoSpTemp == 1 )
        xMpTemp = handles.xMpTemp;

        spTempAtual = handles.sPointTemperaturas(i);
        spTempAnt   = handles.sPointTemperaturas( iAnt );

        % Verifica se houve mudança de set point da temperatura.
        if( spTempAtual ~= spTempAnt | i == iAnt )
            xMpTemp = i;
        else
            if( xMpTemp <= 0 )
                xMpTemp = i;
            else
                tempAtual = handles.temperaturas( i );
                tempMp    = handles.temperaturas( xMpTemp );
                spTempMp  = handles.sPointTemperaturas( xMpTemp );

                varTempAtual = abs( tempAtual - spTempAtual );
                varTempMp    = abs( tempMp - spTempMp );

                if( varTempAtual > varTempMp )
                    xMpTemp = i;
                end
            end
        end
    else
        xMpTemp = 0;
    end

    if( handles.isAtingidoSpUmid == 1 )
        xMpUmid = handles.xMpUmid;

        spUmidAtual = handles.sPointUmidades(i);
        spUmidAnt   = handles.sPointUmidades( iAnt );

        % Verifica se houve mudança de set point.
        if( spUmidAtual ~= spUmidAnt | i == iAnt )
            xMpUmid = i;
```

```
        else
            if( xMpUmid <= 0 )
                xMpUmid = i;
            else
                umidAtual = handles.umidades( i );
                umidMp    = handles.umidades( xMpUmid );
                spUmidMp  = handles.sPointUmidades( xMpUmid );

                varUmidAtual = abs( umidAtual - spUmidAtual );
                varUmidMp    = abs( umidMp - spUmidMp );

                if( varUmidAtual > varUmidMp )
                    xMpUmid = i;
                end
            end
        end
    else
        xMpUmid = 0;
    end


function [xT5Temp, xT5Umid] = getAcomodacao( handles )
    i     = size( handles.temperaturas, 2 );
    if( i <= 1 ); iAnt = i; else iAnt = i - 1; end

    spTempAtual = handles.sPointTemperaturas(i);
    spTempAnt   = handles.sPointTemperaturas( iAnt );

    if( spTempAtual == spTempAnt )
        if( handles.xT5Temp == 0 )
            tempAtual = handles.temperaturas(i);
            varTemp = abs(tempAtual - spTempAtual) / spTempAtual;

            if( varTemp <= 0.05 )
                xT5Temp = i;
            else
                xT5Temp = 0;
            end
        else
            xT5Temp = handles.xT5Temp;
        end
    else
        xT5Temp = 0;
    end



    spUmidAtual = handles.sPointUmidades(i);
    spUmidAnt   = handles.sPointUmidades( iAnt );

    if( spUmidAtual == spUmidAnt )
        if( handles.xT5Umid == 0 )
```

```
            umidAtual = handles.umidades(i);
            varUmid = abs(umidAtual - spUmidAtual) / spUmidAtual;

            if( varUmid <= 0.05 )
                xT5Umid = i;
            else
                xT5Umid = 0;
            end
        else
            xT5Umid = handles.xT5Umid;
        end
    else
        xT5Umid = 0;
    end


% Plota o grafico de temperatura.
function plotTemperatura( handles )
    i     = size( handles.temperaturas, 2 );

    plot( handles.axesTemperatura, handles.x, handles.temperaturas, handles.x,
handles.sPointTemperaturas, 'LineWidth', 1.5 );

    % Apresenta a diferença entre a temperatura e seu respectivo set point.
    value   = handles.temperaturas(i);
    valueSP = handles.sPointTemperaturas(i);
    varPerc = (value - valueSP) / (valueSP) * 100.0;
    msgPerc = [ '{\Delta}:   ', num2str( varPerc, '%.1f' ), '%' ];

    text( handles.axesTemperatura, 0.65, 0.9, msgPerc, 'Units', 'normalized' );

    % Apresenta os valores mínimo, máximo e médio.
    [xMin, xMax] = getXExtremos( handles, i, true );
    valores = handles.temperaturas( 1:1, xMin:xMax );
    vMin = min(    valores );
    vMax = max(    valores );
    vMed = median( valores );

    msgMin = [ 'Mín:  ' , num2str( vMin, '%.1f' ) ];
    msgMax = [ 'Máx: ' , num2str( vMax, '%.1f' ) ];

    text( handles.axesTemperatura, 0.05, 0.20, msgMin  , 'Units', 'normalized' );
    text( handles.axesTemperatura, 0.05, 0.10, msgMax  , 'Units', 'normalized' );

    % Apresenta os valores do Mp, tr e tp.
    if( handles.xMpTemp > 0 )
        mp      = handles.temperaturas( handles.xMpTemp );
        spMp    = handles.sPointTemperaturas( handles.xMpTemp );
        varMp   = (mp - spMp) / (spMp) * 100.0;
        tr      = handles.xTrTempUlt - handles.xTrTempAnt;
        tp      = handles.xMpTemp    - handles.xTrTempAnt;
```

```
        msgMp   = [ 'M_p: ' , num2str( abs(varMp), '%.1f' ), '%' ];
        msgTr   = [ 't_r:  ', num2str( tr, '%.0f' ) ];
        msgTp   = [ 't_p: ' , num2str( tp, '%.0f' ) ];

        text( handles.axesTemperatura, 0.65, 0.80, msgMp  , 'Units', 'normalized' );
        text( handles.axesTemperatura, 0.40, 0.20, msgTr  , 'Units', 'normalized' );
        text( handles.axesTemperatura, 0.40, 0.09, msgTp  , 'Units', 'normalized' );
    else
        text( handles.axesTemperatura, 0.65, 0.80, 'M_p: ?', 'Units', 'normalized' );
        text( handles.axesTemperatura, 0.40, 0.20, 't_r:   ?', 'Units', 'normalized' );
        text( handles.axesTemperatura, 0.40, 0.09, 't_p:  ?', 'Units', 'normalized' );
    end

    % Apresenta o tempo para atingimento dos 5% de diferença.
    if( handles.xT5Temp ~= 0 )
        xUltAltSpTemp = handles.xUltAltSpTemp;

        msg = [ 't_{5%}: ', num2str( (handles.xT5Temp - xUltAltSpTemp) ) ];
        text( handles.axesTemperatura, 0.70, 0.2, msg    , 'Units', 'normalized' );
    else
        text( handles.axesTemperatura, 0.70, 0.2, 't_{5%} ?', 'Units', 'normalized' );
    end

    title(  handles.axesTemperatura, 'Temperatura' );
    ylabel( handles.axesTemperatura, 'Temperatura (ºC)', 'FontSize', 10 );
    legend( handles.axesTemperatura, 'Temperatura', 'Location', 'northwest' );


% Plota o grafico de umidade.
function plotUmidade( handles )
    i     = size( handles.umidades, 2 );

    plot( handles.axesUmidade, handles.x, handles.umidades, handles.x, handles.sPointUmidades,
'LineWidth', 1.5 );

    % Apresenta a diferença entre a umidade e seu respectivo set point.
    value = handles.umidades(i);
    valueSP = handles.sPointUmidades(i);
    varPerc = (value - valueSP) / (valueSP) * 100.0;
    msgPerc = [ '{\Delta}:   ', num2str( varPerc, '%.1f' ), '%' ];

    text( handles.axesUmidade, 0.65, 0.9, msgPerc, 'Units', 'normalized' );

    % Apresenta os valores mínimo, máximo e médio.
    [xMin, xMax] = getXExtremos( handles, i, true );
    valores = handles.umidades( 1:1, xMin:xMax );
    vMin = min(    valores );
    vMax = max(    valores );
    vMed = median( valores );
```

```
    msgMin = [ 'Mín:  ' , num2str( vMin, '%.1f' ) ];
    msgMax = [ 'Máx: ' , num2str( vMax, '%.1f' ) ];

    text( handles.axesUmidade, 0.05, 0.20, msgMin  , 'Units', 'normalized' );
    text( handles.axesUmidade, 0.05, 0.10, msgMax  , 'Units', 'normalized' );

    % Apresenta os valores do Mp, tr e tp.
    if( handles.xMpUmid > 0 )
        mp      = handles.umidades( handles.xMpUmid );
        spMp    = handles.sPointUmidades( handles.xMpUmid );
        varMp   = (mp - spMp) / (spMp) * 100.0;
        tr      = handles.xTrUmidUlt - handles.xTrUmidAnt;
        tp      = handles.xMpUmid    - handles.xTrUmidAnt;

        msgMp   = [ 'M_p: ' , num2str( abs(varMp), '%.1f' ), '%' ];
        msgTr   = [ 't_r:  ', num2str( tr, '%.0f' ) ];
        msgTp   = [ 't_p: ' , num2str( tp, '%.0f' ) ];

        text( handles.axesUmidade, 0.65, 0.80, msgMp  , 'Units', 'normalized' );
        text( handles.axesUmidade, 0.40, 0.20, msgTr  , 'Units', 'normalized' );
        text( handles.axesUmidade, 0.40, 0.09, msgTp  , 'Units', 'normalized' );
    else
        text( handles.axesUmidade, 0.65, 0.80, 'M_p: ?', 'Units', 'normalized' );
        text( handles.axesUmidade, 0.40, 0.20, 't_r:  ?', 'Units', 'normalized' );
        text( handles.axesUmidade, 0.40, 0.09, 't_p: ?', 'Units', 'normalized' );
    end

    % Apresenta o tempo para atingimento dos 5% de diferença.
    if( handles.xT5Umid ~= 0 )
        xUltAltSpUmid = handles.xUltAltSpUmid;

        msg = [ 't_{5%}: ', num2str( (handles.xT5Umid - xUltAltSpUmid) ) ];
        text( handles.axesUmidade, 0.70, 0.2, msg    , 'Units', 'normalized' );
    else
        text( handles.axesUmidade, 0.70, 0.2, 't_{5%}: ?', 'Units', 'normalized' );
    end

    title(  handles.axesUmidade, 'Umidade' );
    ylabel( handles.axesUmidade, 'Umidade (%)', 'FontSize', 10 );
    legend( handles.axesUmidade, 'Umidade', 'Location', 'northwest' );


% Plota o grafico da temperatura fuzzificada.
function plotTempFuzz( handles )
    plot( handles.axesTemperaturaFuzzificada, handles.x, handles.temperaturasFuzzificadas,
'LineWidth', 1.5 );

    title(  handles.axesTemperaturaFuzzificada, 'Temperatura Fuzzificada' );
    ylabel( handles.axesTemperaturaFuzzificada, 'Temperatura (Fuzzy)', 'FontSize', 10 );
```

```
% Plota o grafico da umidade fuzzificada.
function plotUmidFuzz( handles )
    plot( handles.axesUmidadeFuzzificada, handles.x, handles.umidadesFuzzificadas,
'LineWidth', 1.5 );

    title(  handles.axesUmidadeFuzzificada, 'Umidade Fuzzificada' );
    ylabel( handles.axesUmidadeFuzzificada, 'Umidade (Fuzzy)', 'FontSize', 10 );


% Plota o grafico da ventilação.
function plotVentilacao( handles )
    plot( handles.axesVentilacao, handles.x, handles.potenciasVentilacao, 'LineWidth', 1.5 );

    title(  handles.axesVentilacao, 'Ventilação' );
    ylabel( handles.axesVentilacao, 'Potência (Fuzzy)', 'FontSize', 10 );


% Plota o grafico da umidificacao.
function plotUmidificacao( handles )
    plot( handles.axesUmidificacao, handles.x, handles.potenciasUmidificacao, 'LineWidth',
1.5 );

    title(  handles.axesUmidificacao, 'Umidificacao' );
    ylabel( handles.axesUmidificacao, 'Potência (Fuzzy)', 'FontSize', 10 );


% Plota o grafico do aquecimento.
function plotAquecimento( handles )
    plot( handles.axesAquecimento, handles.x, handles.potenciasAquecimento, 'LineWidth',
1.5 );

    title(  handles.axesAquecimento, 'Aquecimento' );
    ylabel( handles.axesAquecimento, 'Potência (Fuzzy)', 'FontSize', 10 );


% Configura os valores dos eixos de cada gráfico
function configAxisAxes( handles )
    i = size( handles.potenciasUmidificacao, 2 );

    [xMin, xMax] = getXExtremos( handles, i, false );

    if( xMax <= i ); xMaxValid = xMax; else xMaxValid = i; end

    yMinTemp = min( min( handles.temperaturas(xMin:xMaxValid) ),
min( handles.sPointTemperaturas(xMin:xMaxValid) ) ) - 10;
    yMaxTemp = max( max( handles.temperaturas(xMin:xMaxValid) ),
max( handles.sPointTemperaturas(xMin:xMaxValid) ) ) + 10;

    yMinUmid = min( min( handles.umidades(xMin:xMaxValid) )    ,
min( handles.sPointUmidades(xMin:xMaxValid) ) ) - 10;
```

```
    yMaxUmid = max( max( handles.umidades(xMin:xMaxValid) )    ,
max( handles.sPointUmidades(xMin:xMaxValid) ) ) + 10;

    axis( handles.axesTemperatura, [xMin xMax yMinTemp yMaxTemp] );
    axis( handles.axesUmidade    , [xMin xMax yMinUmid yMaxUmid] );


    yMinFuzz = -1;
    yMaxFuzz = +1;

    axis( handles.axesTemperaturaFuzzificada, [xMin xMax yMinFuzz yMaxFuzz] );
    axis( handles.axesUmidadeFuzzificada    , [xMin xMax yMinFuzz yMaxFuzz] );


    yMinAtuadores = +0;
    yMaxAtuadores = +1;

    axis( handles.axesVentilacao  , [xMin xMax yMinAtuadores yMaxAtuadores] );
    axis( handles.axesUmidificacao, [xMin xMax yMinAtuadores yMaxAtuadores] );
    axis( handles.axesAquecimento , [xMin xMax yMinAtuadores yMaxAtuadores] );


% Configura os textos apresentados nos graficos.
function configTextAxes( handles )
    i = size( handles.potenciasUmidificacao, 2 );

    [xMin, xMax] = getXExtremos( handles, i, false );

    if( i > xMax ); xText = xMax; else xText = i; end

    temp     = handles.temperaturas(i);
    umid     = handles.umidades(i);
    tempFuzz = handles.temperaturasFuzzificadas(i);
    umidFuzz = handles.umidadesFuzzificadas(i);
    potVent  = handles.potenciasVentilacao(i);
    potUmid  = handles.potenciasUmidificacao(i);
    potAquec = handles.potenciasAquecimento(i);

    text( handles.axesTemperatura           , xText, temp    , num2str( temp    , 3 ) );
    text( handles.axesUmidade               , xText, umid    , num2str( umid    , 3 ) );
    text( handles.axesTemperaturaFuzzificada, xText, tempFuzz, num2str( tempFuzz, 3 ) );
    text( handles.axesUmidadeFuzzificada    , xText, umidFuzz, num2str( umidFuzz, 3 ) );
    text( handles.axesVentilacao            , xText, potVent , num2str( potVent , 3 ) );
    text( handles.axesUmidificacao          , xText, potUmid , num2str( potUmid , 3 ) );
    text( handles.axesAquecimento           , xText, potAquec, num2str( potAquec, 3 ) );


function editXMin_Callback(hObject, eventdata, handles)
    configAxisAxes( handles );
    configTextAxes( handles );
```

```
function editXMin_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end


function editXMax_Callback(hObject, eventdata, handles)
configAxisAxes( handles );
configTextAxes( handles );


function editXMax_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end


function pushbuttonIncrSpTemp_Callback(hObject, eventdata, handles)
    i = size( handles.sPointTemperaturas, 2 );
    sendSetPointTempFcn( handles.serial, handles.sPointTemperaturas(i) + 1 );


function pushbuttonDecrSpTemp_Callback(hObject, eventdata, handles)
    i = size( handles.sPointTemperaturas, 2 );
    sendSetPointTempFcn( handles.serial, handles.sPointTemperaturas(i) - 1 );


function pushbuttonIncrSpUmid_Callback(hObject, eventdata, handles)
    i = size( handles.sPointUmidades, 2 );
    sendSetPointUmidFcn( handles.serial, handles.sPointUmidades(i) + 1 );


function pushbuttonDecrSpUmid_Callback(hObject, eventdata, handles)
    i = size( handles.sPointUmidades, 2 );
    sendSetPointUmidFcn( handles.serial, handles.sPointUmidades(i) - 1 );
</code>
```

1.2 Arquivo getConnectionSerialFcn.m

Retorna uma conexão com a *serial*. Caso ainda não exista uma conexão estabelecida, solicita a criação de uma nova.

```
<code>
function serial = getConnectionSerialFcn( nrCOM, baudRate )
    coder.extrinsic( 'exist' );
```

```
    isExistSerial = 0;
    isExistSerial = exist( 'serial', 'var' );

    if( isExistSerial == 1 )
        statusSerial = '    ';
        statusSerial = get( serial, 'status' );

        if( ~strcmp( statusSerial, 'open' ) )
            serial = openConnectionSerialFcn( nrCOM, baudRate );
            aguardarRespostaConnectionSerialFcn( serial );
            set( serial, 'Timeout', 0.200 );
        end
    else
        serial = openConnectionSerialFcn( nrCOM, baudRate );
        aguardarRespostaConnectionSerialFcn( serial );
    end
end
</code>
```

1.3 Arquivo closeConnectionSerialFcn.m

Encerra a conexão com a *serial*, para liberação do recurso.

```
<code>
function closeConnectionSerialFcn( s )
    coder.extrinsic( 'delete' );

    fclose(s);
    delete(s);
end
</code>
```

1.4 Arquivo readDoubleSerialFcn.m

Envia dados via serial à interface Software-Hardware.

```
<code>
function value = readDoubleSerialFcn( serial, textoEnvio )
    %% Read buffer data
    dados = blanks(8);

    writedata = uint8( textoEnvio );
    fwrite( serial, writedata, 'uint8' );
```

```
    dados = fscanf( serial );

    value = str2double( dados );
end
</code>
```

1.5 Arquivo readSetPointTemperaturaFcn.m

Solicita o valor de *set point* da temperatura à interface Software-Hardware.

```
<code>
function value = readSetPointTemperaturaFcn( s )
    value = readDoubleSerialFcn( s, 'T' );
end
</code>
```

1.6 Arquivo readSetPointUmidadeFcn.m

Solicita o valor de *set point* da umidade à interface Software-Hardware.

```
<code>
function value = readSetPointUmidadeFcn( s )
    value = readDoubleSerialFcn( s, 'U' );
end
</code>
```

1.7 Arquivo readStatusStopFcn.m

Verifica junto à interface Software-Hardware se houve o pressionamento do botão STOP.

```
<code>
function isStop = readStatusStopFcn( s )
    dados = readDoubleSerialFcn( s, 's' );

    if( dados == 1 )
        isStop = true;
    else
        isStop = false;
    end
end
</code>
```

1.8 Arquivo readTemperaturaFcn.m

Solicita o valor da temperatura à interface Software-Hardware.

```
<code>
function value = readTemperaturaFcn( s )
    value = readDoubleSerialFcn( s, 't' );
end
</code>
```

1.9 Arquivo readUmidadeFcn.m

Solicita o valor da umidade à interface Software-Hardware.

```
<code>
function value = readUmidadeFcn( s )
    value = readDoubleSerialFcn( s, 'h' );
end
</code>
>
```

1.10 Arquivo sendSerialFcn.m

Envia à interface Software-Hardware os dados especificados.

```
<code>
function sendSerialFcn( serial, key, valor )
    %coder.extrinsic( 'fgetl' );
    %coder.extrinsic( 'str2double' );

    writedata = uint8( key );
    fwrite( serial, writedata, 'uint8' );

    fscanf( serial );

    writedata = num2str( valor );
    fwrite( serial, writedata );

    fscanf( serial );
end
</code>
```

1.11 Arquivo sendPotenciasFcn.m

Envia à interface Software-Hardware os valores de potência a serem aplicados nos atuadores, desde que tenha ocorrido mudança em relação ao valor da iteração anterior.

```
<code>
function sendPotenciasFcn( serial, potVentAnt, potenciaVentilacao, potUmidAnt,
potenciaUmidificacao, potAquecAnt, potenciaAquecimento )
    %coder.extrinsic( 'fgetl' );
    %coder.extrinsic( 'str2double' );

    %% Read buffer data
    dados = blanks(8);

    if( potVentAnt ~= potenciaVentilacao )
        sendSerialFcn( serial, 'x', potenciaVentilacao );
    end

    if( potUmidAnt ~= potenciaUmidificacao )
        sendSerialFcn( serial, 'y', potenciaUmidificacao );
    end

    if( potAquecAnt ~= potenciaAquecimento )
        sendSerialFcn( serial, 'z', potenciaAquecimento );
    end
end
</code>
```

1.12 Arquivo sendSetPointTempFcn.m

Envia à interface Software-Hardware o novo valor de set point para a temperatura.

```
<code>
function sendSetPointTempFcn( serial, novoSp )
    sendSerialFcn( serial, 'p', novoSp );
end
</code>
```

1.13 Arquivo sendSetPointUmidadeFcn.m

Envia à interface Software-Hardware o novo valor de set point para a umidade.

```
<code>
function sendSetPointUmidFcn( serial, novoSp )
    sendSerialFcn( serial, 'P', novoSp );
end
</code>
```

1.14 Arquivo stopSistemaFcn.m

Envia à interface Software-Hardware a instrução para que o sistema seja parado. Equivale ao pressionamento do botão STOP na maquete

```
<code>
function isStop = stopSistemaFcn( s )
    %coder.extrinsic( 'fgetl' );
    %coder.extrinsic( 'str2double' );

    cd = readDoubleSerialFcn( s, 'S' );

    if( cd == 1 )
        isStop = true;
    else
        isStop = false;
    end
end
</code>
```

2 FUZZY

2.1 Arquivo “fuzzy_contr_temp_umid_vrs03.fis”

Arquivo de definição gerado através da modelagem *fuzzy*, o qual contém a descrição do método de inferência, das variáveis de entrada e saída, e do conjunto de regras elaboradas. Esse é o arquivo carregado pelo programa MATLAB e a partir do qual são realizados os cálculos para obtenção dos valores de potência a serem aplicados nos atuadores.

```
<code>
[System]
Name='fuzzy_contr_temp_umid_vrs03'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=3
NumRules=49
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='temperatura'
Range=[-1 1]
NumMFs=7
MF1='NS':'trapmf',[-0.5 -0.35 -0.15 0]
MF2='ZE':'trapmf',[-0.1 -0.05 0.05 0.1]
MF3='PS':'trapmf',[0 0.15 0.35 0.5]
MF4='NM':'trapmf',[-0.8 -0.6 -0.4 -0.2]
MF5='PM':'trapmf',[0.2 0.4 0.6 0.8]
MF6='PB':'trapmf',[0.5 0.8 1 2]
MF7='NB':'trapmf',[-2 -1 -0.8 -0.5]

[Input2]
Name='umidade'
Range=[-1 1]
NumMFs=7
MF1='NB':'trapmf',[-2 -1 -0.9 -0.6]
MF2='ZE':'trapmf',[-0.15 -0.05 0.05 0.15]
MF3='PB':'trapmf',[0.5 0.8 1 2]
MF4='NM':'trapmf',[-0.85 -0.75 -0.65 -0.45]
MF5='NS':'trapmf',[-0.5 -0.4 -0.3 -0.1]
```

```
MF6='PS':'trapmf',[0.1 0.15 0.35 0.5]
MF7='PM':'trapmf',[0.2 0.4 0.6 0.8]

[Output1]
Name='ventilador'
Range=[0 1]
NumMFs=4
MF1='ZE':'trimf',[0 0 0.15]
MF2='PS':'trapmf',[0.1 0.3 0.4 0.55]
MF3='PB':'trapmf',[0.6 0.9 1 1]
MF4='PM':'trapmf',[0.4 0.55 0.75 0.9]

[Output2]
Name='umidificador'
Range=[0 1]
NumMFs=4
MF1='ZE':'trimf',[0 0 0.201446280991735]
MF2='PS':'trapmf',[0.15 0.35 0.45 0.6]
MF3='PM':'trapmf',[0.4 0.6 0.75 0.9]
MF4='PB':'trapmf',[0.7 0.9 1 1]

[Output3]
Name='resistencia'
Range=[0 1]
NumMFs=4
MF1='ZE':'trimf',[0 0 0.1]
MF2='PS':'trapmf',[0 0.2 0.4 0.6]
MF3='PM':'trapmf',[0.3 0.5 0.7 0.9]
MF4='PB':'trapmf',[0.6 0.8 1 1]

[Rules]
7 1, 4 4 4 (1) : 1
7 4, 4 3 4 (1) : 1
7 5, 2 2 4 (1) : 1
7 2, 2 2 4 (1) : 1
7 6, 4 2 4 (1) : 1
7 7, 4 1 4 (1) : 1
7 3, 4 1 4 (1) : 1
4 1, 4 4 3 (1) : 1
4 4, 4 3 3 (1) : 1
4 5, 4 2 3 (1) : 1
4 2, 4 2 3 (1) : 1
4 6, 4 1 3 (1) : 1
4 7, 4 1 3 (1) : 1
4 3, 4 1 3 (1) : 1
1 1, 2 4 2 (1) : 1
1 4, 2 3 2 (1) : 1
1 5, 1 2 2 (1) : 1
1 2, 2 1 2 (1) : 1
1 6, 2 1 2 (1) : 1
1 7, 2 1 2 (1) : 1
```

```
1 3, 2 1 2 (1) : 1
2 3, 2 1 1 (1) : 1
2 7, 2 1 1 (1) : 1
2 6, 2 1 1 (1) : 1
3 1, 2 4 1 (1) : 1
3 4, 2 3 1 (1) : 1
3 5, 2 2 1 (1) : 1
3 2, 2 1 1 (1) : 1
3 6, 4 1 1 (1) : 1
3 7, 4 1 1 (1) : 1
3 3, 4 1 1 (1) : 1
5 1, 4 4 1 (1) : 1
5 4, 4 3 1 (1) : 1
5 5, 2 2 1 (1) : 1
5 2, 4 2 1 (1) : 1
5 6, 4 1 1 (1) : 1
5 7, 4 1 1 (1) : 1
5 3, 3 1 1 (1) : 1
6 1, 4 4 1 (1) : 1
6 4, 4 3 1 (1) : 1
6 5, 4 2 1 (1) : 1
6 2, 4 2 1 (1) : 1
6 6, 4 1 1 (1) : 1
6 7, 4 1 1 (1) : 1
6 3, 3 1 1 (1) : 1
2 1, 1 4 1 (1) : 1
2 5, 1 2 1 (1) : 1
2 4, 1 3 1 (1) : 1
2 2, 1 1 1 (1) : 1
</code>
```

2.2 Arquivo “fuzzy_contr_temp_umid_umid_gauss_vrs01.fis”

Arquivo de definição gerado para quando se necessita um comportanto mais suave e preciso, pois utiliza a função **gaussiana** para praticamente todas as funções de pertinência.

```
<code>
[System]
Name='fuzzy_contr_temp_umid_umid_gauss_vrs01'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=3
NumRules=49
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'
```

```
[Input1]
Name='temperatura'
Range=[-1 1]
NumMFs=7
MF1='NS':'trapmf',[-0.5 -0.3 -0.25 -0.05]
MF2='ZE':'trapmf',[-0.075 -0.05 0.05 0.075]
MF3='PS':'trapmf',[0 0.15 0.35 0.5]
MF4='NM':'trapmf',[-0.75 -0.6 -0.45 -0.25]
MF5='PM':'trapmf',[0.2 0.4 0.6 0.8]
MF6='PB':'trapmf',[0.5 0.8 1 2]
MF7='NB':'trapmf',[-2 -1 -0.8 -0.5]

[Input2]
Name='umidade'
Range=[-1 1]
NumMFs=7
MF1='NB':'trapmf',[-2 -1 -0.9 -0.6]
MF2='ZE':'gaussmf',[0.101028198018131 0]
MF3='PB':'trapmf',[0.5 0.8 1 2]
MF4='NM':'gaussmf',[0.155 -0.679125806451613]
MF5='NS':'gaussmf',[0.116 -0.380193548387097]
MF6='PS':'gaussmf',[0.08 0.225]
MF7='PM':'gaussmf',[0.112 0.537507936507936]

[Output1]
Name='ventilador'
Range=[0 1]
NumMFs=4
MF1='ZE':'trapmf',[-1 0 0.05 0.15]
MF2='PS':'gaussmf',[0.1 0.25]
MF3='PB':'sigmf',[52.9 0.919355026455027]
MF4='PM':'gaussmf',[0.125 0.475]

[Output2]
Name='umidificador'
Range=[0 1]
NumMFs=4
MF1='ZE':'trimf',[0 0 0.1]
MF2='PS':'gaussmf',[0.07638 0.23]
MF3='PM':'gaussmf',[0.142 0.455]
MF4='PB':'sigmf',[18.4 0.760596774193549]

[Output3]
Name='resistencia'
Range=[0 1]
NumMFs=4
MF1='ZE':'trimf',[0 0 0.075]
MF2='PS':'gaussmf',[0.06 0.195]
MF3='PM':'gaussmf',[0.0545 0.4]
MF4='PB':'gaussmf',[0.06 0.6]
```

```
[Rules]
7 1, 4 4 4 (1) : 1
7 4, 4 3 4 (1) : 1
7 5, 2 2 4 (1) : 1
7 2, 2 1 4 (1) : 1
7 6, 4 1 4 (1) : 1
7 7, 4 1 3 (1) : 1
7 3, 4 1 3 (1) : 1
4 1, 4 4 3 (1) : 1
4 4, 4 3 3 (1) : 1
4 5, 4 2 3 (1) : 1
4 2, 2 1 3 (1) : 1
4 6, 4 1 3 (1) : 1
4 7, 4 1 3 (1) : 1
4 3, 4 1 3 (1) : 1
1 1, 1 4 1 (1) : 1
1 4, 1 3 1 (1) : 1
1 5, 2 2 2 (1) : 1
1 2, 2 1 2 (1) : 1
1 6, 2 1 2 (1) : 1
1 7, 2 1 2 (1) : 1
1 3, 2 1 3 (1) : 1
2 3, 4 1 3 (1) : 1
2 7, 2 1 3 (1) : 1
2 6, 2 1 2 (1) : 1
3 1, 2 4 1 (1) : 1
3 4, 2 3 1 (1) : 1
3 5, 2 2 1 (1) : 1
3 2, 2 2 1 (1) : 1
3 6, 4 1 2 (1) : 1
3 7, 4 1 2 (1) : 1
3 3, 4 1 3 (1) : 1
5 1, 4 4 1 (1) : 1
5 4, 4 3 1 (1) : 1
5 5, 2 2 1 (1) : 1
5 2, 4 2 1 (1) : 1
5 6, 2 1 1 (1) : 1
5 7, 4 1 1 (1) : 1
5 3, 3 1 2 (1) : 1
6 1, 4 4 1 (1) : 1
6 4, 4 3 1 (1) : 1
6 5, 4 2 1 (1) : 1
6 2, 4 2 1 (1) : 1
6 6, 4 1 1 (1) : 1
6 7, 4 1 1 (1) : 1
6 3, 3 1 1 (1) : 1
2 1, 1 4 1 (1) : 1
2 5, 1 2 1 (1) : 1
2 4, 1 3 1 (1) : 1
2 2, 1 1 1 (1) : 1
</code>
```

3 ARDUINO

3.1 Código-fonte “interface_sh_vrs02.ino”

O código-fonte a seguir é executado no Arduino e é responsável por comunicar-se com a Central de Processamento, obter os valores de temperatura e umidade através do sensor DHT22 (ou DHT11), repassar aos circuitos dos atuadores os valores de potência definidos pelo Sistema de Inferência *Fuzzy*, verificar se algum botão da interface foi pressionado pelo usuário e apresentar nos *displays* os valores de temperatura, umidade e *set points*.

```
<code>
#include <Arduino.h>
#include <string.h>

#if defined (__arm__) && defined (__SAM3X8E__)
  #include <FlashNumber.h>
#else
  #include <EEPROM.h>
  #include <EepromNumber.h>
#endif

#include "DHT.h"
#include "LedControl.h"

#include "Constantes.c"

//Objetos
#if defined (__arm__) && defined (__SAM3X8E__)
  FlashNumber nonVolatileNumber = FlashNumber();
#else
  EepromNumber nonVolatileNumber = EepromNumber();
#endif

LedControl lc = LedControl( PIN_DIN, PIN_CLK, PIN_LOAD, QTD_DISPLAYS );
DHT dht( DHT_PIN, DHT_TYPE );

//Variáveis primitivas
float sPointTemperatura   = 0.0;
float sPointUmidade       = 0.0;

float temperaturaAnterior = 0.0;
float umidadeAnterior     = 0.0;
```

```
float sPTAnterior         = 0.0;
float sPUAnterior         = 0.0;

byte requisicao;

boolean isPressBotaoTDecr = false;
boolean isPressBotaoTIncr = false;
boolean isPressBotaoUDecr = false;
boolean isPressBotaoUIncr = false;
boolean isPressBotaoStop  = false;

float potenciaVentilacao                = 0.0;   //Valor de 0 a 1.
float potenciaUmidificacao              = 0.0;   //Valor de 0 a 1.
float potenciaAquecimento               = 0.0;   //Valor de 0 a 1.

/**
 * Comandos possiveis na comunicação ARDUINO - MATLAB:
 * 1, '1', 't'.....: envia valor da temperatura em grau celcius.
 * 2, '2', 'h', 'u': envia percentual da umidade.
 * 'T'.............: envia valor do set point da temperatura.
 * 'H', 'U'........: envia valor do set point da umidade.
 * 'p'.............: recebe comando para alterar o valor de set point da temperatura.
 * 'P'.............: recebe comando para alterar o valor de set point da umidade.
 * 's'.............: envia status do stop.
 * 'S'.............: recebe comando para PARAR (STOP)o sistema de controle.
 * 'x'.............: recebe valor de potencia (de 0 a 1) a ser aplicado na ventilacao.
 * 'y'.............: recebe valor de potencia (de 0 a 1) a ser aplicado na umidificacao.
 * 'z'.............: recebe valor de potencia (de 0 a 1) a ser aplicado no aquecimento.
 * 10, 13..........: ignorado.
 */

void setup(){
  int count = 0;

  initDisplays();
  updateDisplaysTeste( count++ );
  initPinos();
  updateDisplaysTeste( count++ );
  initSerial();
  updateDisplaysTeste( count++ );
  initSensorTermohigrometria();
  updateDisplaysTeste( count++ );
  initSetPoints();
  updateDisplaysTeste( count++ );
}

void loop() {
  digitalWrite( PIN_PROCESSANDO, LOW );

  //Realiza a leitura dos sensores.
  float temperatura = dht.readTemperature();
```

```
  float umidade = dht.readHumidity();

  //Verifica se algum dos botoes foi pressionado.
  checkBotaoPressionado();

  //Verifica se alguma informação chegou pela Serial.
  checkSerial( temperatura, umidade );

  //Atualiza os dados apresentados nos displays.
  updateDisplays( temperaturaAnterior, umidadeAnterior, temperatura, umidade );
  updateDisplaysSetPoint( sPTAnterior, sPUAnterior, sPointTemperatura, sPointUmidade );

  //Atualiza as potencias aplicadas nos atuadores.
  updatePotencias();

  //Atualiza os valores armazenados na EEPROM.
  updateEeprom( sPTAnterior, sPUAnterior, sPointTemperatura, sPointUmidade );

  //Altera os valores das medidas anteriores para conterem os valores obtidos neste ciclo.
  temperaturaAnterior = temperatura;
  umidadeAnterior = umidade;
  sPTAnterior = sPointTemperatura;
  sPUAnterior = sPointUmidade;

  delay( DELAY_LOOP );
}


//***** INITs *****//

void initDisplays(){
  lc.shutdown( 0,false );

  for( int index = 0; index < QTD_DISPLAYS; index++ ){
    lc.setIntensity( index, 10 );
    lc.clearDisplay( index );
  }

  updateDisplays( -1, -1, temperaturaAnterior, umidadeAnterior );
}

void initPinos(){
  pinMode( PIN_PROCESSANDO, OUTPUT );

  pinMode( PIN_BOTAO_T_DECR, INPUT );
  pinMode( PIN_BOTAO_T_INCR, INPUT );
  pinMode( PIN_BOTAO_U_DECR, INPUT );
  pinMode( PIN_BOTAO_U_INCR, INPUT );
  pinMode( PIN_BOTAO_STOP  , INPUT );

  pinMode( PIN_POT_VENTILACAO  , OUTPUT );
```

```
  pinMode( PIN_POT_UMIDIFICACAO, OUTPUT );
  pinMode( PIN_POT_AQUECIMENTO , OUTPUT );
}

void initSerial(){
  Serial.begin( SERIAL_BAUD );
  Serial.setTimeout( SERIAL_TIMEOUT_INCOMING );
  Serial.println( "OK" );
}

void initSensorTermohigrometria(){
  dht.begin();
}

/**
 * Atualiza as variáveis de set point da temperatura e umidade conforme os valores salvos na
EEPROM.
 */
void initSetPoints(){
  int statusSistema     = nonVolatileNumber.readInt( EEPROM_ADDRESS_STATUS_SIS  );
  int temperaturaEeprom = nonVolatileNumber.readInt( EEPROM_ADDRESS_TEMPERATURA );
  int umidadeEeprom     = nonVolatileNumber.readInt( EEPROM_ADDRESS_UMIDADE     );

  if( statusSistema == EEPROM_STATUS_PRONTO ){
    sPointTemperatura = (temperaturaEeprom >= TEMPERATURA_MIN_VALIDA && temperaturaEeprom <=
TEMPERATURA_MAX_VALIDA) ? temperaturaEeprom : TEMPERATURA_DEFAULT;
    sPointUmidade     = (umidadeEeprom     >= UMIDADE_MIN_VALIDA     && umidadeEeprom     <=
UMIDADE_MAX_VALIDA    ) ? umidadeEeprom     : UMIDADE_DEFAULT;
  }
  else{
    sPointTemperatura = TEMPERATURA_DEFAULT;
    sPointUmidade     = UMIDADE_DEFAULT;

    nonVolatileNumber.writeInt( EEPROM_ADDRESS_STATUS_SIS, EEPROM_STATUS_PRONTO );
  }
}


//***** CHECKs *****//

void checkBotaoPressionado(){
  //Verifica se botao STOP foi pressionado
  if( digitalRead( PIN_BOTAO_STOP ) ){
    stopSistema();
  }

  //Verifica se algum botao foi pressionado
  if( digitalRead( PIN_BOTAO_T_DECR ) ){
    if( isPressBotaoTDecr == false ){
      updateSetPointTemp( sPointTemperatura - 1 );
```

```
      isPressBotaoTDecr = true;
    }
  }
  else{
    isPressBotaoTDecr = false;
  }

  if( digitalRead( PIN_BOTAO_T_INCR ) ){
    if( isPressBotaoTIncr == false ){
      updateSetPointTemp( sPointTemperatura + 1 );

      isPressBotaoTIncr = true;
    }
  }
  else{
    isPressBotaoTIncr = false;
  }

  if( digitalRead( PIN_BOTAO_U_DECR ) ){
    if( isPressBotaoUDecr == false ){
      updateSetPointUmid( sPointUmidade - 1 );

      isPressBotaoUDecr = true;
    }
  }
  else{
    isPressBotaoUDecr = false;
  }

  if( digitalRead( PIN_BOTAO_U_INCR ) ){
    if( isPressBotaoUIncr == false ){
      updateSetPointUmid( sPointUmidade + 1 );

      isPressBotaoUIncr = true;
    }
  }
  else{
    isPressBotaoUIncr = false;
  }
}

void checkSerial( float temperatura, float umidade ){
  //Verifica se ha alguma informacao chegando pela Serial.
  if( Serial.available() > 0 ){
    digitalWrite( PIN_PROCESSANDO, HIGH );
    delay( DELAY_READ );
    requisicao = Serial.read();

    float value = 0;

    switch( requisicao ){
```

```
    case 1:
    case '1':
    case 't':
      value = temperatura;
      Serial.println( value );

      break;

    case 2:
    case '2':
    case 'h':
    case 'u':
      value = umidade;
      Serial.println( value );
      break;

    case 'T':
      Serial.println( sPointTemperatura );
      break;

    case 'U':
    case 'H':
      Serial.println( sPointUmidade );
      break;

    case 'p':
      getValorSetPointTemperatura();
      break;

    case 'P':
      getValorSetPointUmidade();
      break;

    case 's':
      Serial.println( isPressBotaoStop == true ? 1 : 0 );
      break;

    case 'S':
      Serial.println( stopSistema() == true ? 1 : 0 );
      break;

    case 'x':
      potenciaVentilacao = getValorPotencia();

      break;

    case 'y':
      potenciaUmidificacao = getValorPotencia();

      break;
```

```
      case 'z':
        potenciaAquecimento = getValorPotencia();

        break;

      case 10:
      case 13:
        break;

      default:
        Serial.println( "nao reconhecido comando " + requisicao );
    }

    digitalWrite( PIN_PROCESSANDO, LOW );
  }
}

float getValorPotencia(){
  Serial.println( '>' );

  String valueStr;
  float value = 0.0;

  boolean isDadoRecebido = false;
  long tsInicio = millis();

  do{
    if( Serial.available() > 0 ){
      valueStr = Serial.readString();
      value = valueStr.toFloat();
      isDadoRecebido = true;
      Serial.println( '.' );
    }
  }
  while( ( millis() - tsInicio ) < SERIAL_TIMEOUT_READ_MS && isDadoRecebido == false );

  return value;
}

void getValorSetPointTemperatura(){
  Serial.println( '>' );

  String valueStr;

  boolean isDadoRecebido = false;
  long tsInicio = millis();

  do{
    if( Serial.available() > 0 ){
      valueStr = Serial.readString();
      int value = valueStr.toInt();
```

```
      isDadoRecebido = true;
      Serial.println( '.' );

      updateSetPointTemp( value );
    }
  }
  while( ( millis() - tsInicio ) < SERIAL_TIMEOUT_READ_MS && isDadoRecebido == false );
}

void getValorSetPointUmidade(){
  Serial.println( '>' );

  String valueStr;

  boolean isDadoRecebido = false;
  long tsInicio = millis();

  do{
    if( Serial.available() > 0 ){
      valueStr = Serial.readString();
      int value = valueStr.toInt();
      isDadoRecebido = true;
      Serial.println( '.' );

      updateSetPointUmid( value );
    }
  }
  while( ( millis() - tsInicio ) < SERIAL_TIMEOUT_READ_MS && isDadoRecebido == false );
}

//***** UPDATEs *****//

void updateDisplays( float temperaturaAnterior, float umidadeAnterior, float temperatura,
float umidade ){
  updateDisplays( temperaturaAnterior, temperatura, DIG_DISP_T_DEZENA, DIG_DISP_T_UNIDADE,
TEMPERATURA_MIN_VALIDA, TEMPERATURA_MAX_VALIDA, VALOR_IDENTIFICADOR_ERRO );
  updateDisplays( umidadeAnterior    , umidade    , DIG_DISP_U_DEZENA, DIG_DISP_U_UNIDADE,
UMIDADE_MIN_VALIDA    , UMIDADE_MAX_VALIDA    , VALOR_IDENTIFICADOR_ERRO );
}

void updateDisplaysSetPoint( float temperaturaAnterior, float umidadeAnterior, float
temperatura, float umidade ){
  updateDisplays( temperaturaAnterior, temperatura, DIG_DISP_SP_T_DEZENA,
DIG_DISP_SP_T_UNIDADE, TEMPERATURA_MIN_VALIDA, TEMPERATURA_MAX_VALIDA,
VALOR_IDENTIFICADOR_ERRO );
  updateDisplays( umidadeAnterior    , umidade    , DIG_DISP_SP_U_DEZENA,
DIG_DISP_SP_U_UNIDADE, UMIDADE_MIN_VALIDA    , UMIDADE_MAX_VALIDA    ,
VALOR_IDENTIFICADOR_ERRO );
}
```

```
void updateDisplays( float valorAnterior, float valorAtual, int indexDisplayDezena, int
indexDisplayUnidade, int valorMinimoValido, int valorMaximoValido, int
valorIdentificadorErro ){
  if( valorAtual != valorAnterior ){
    if( valorAtual >= valorMinimoValido && valorAtual <= valorMaximoValido ){
      updateDisplay( indexDisplayDezena , getDezena( valorAtual ) , false );
      updateDisplay( indexDisplayUnidade, getUnidade( valorAtual ), true );
    }
    else{
      updateDisplay( indexDisplayDezena , valorIdentificadorErro, true );
      updateDisplay( indexDisplayUnidade, valorIdentificadorErro, true );
    }
  }
}

void updateDisplaysTeste( int value ){
  for( int index = 0; index < QTD_DISPLAYS; index++ ){
    updateDisplay( index, value, true );
  }

  delay( DELAY_INIT_SIS );
}

void updateDisplay( int indexDisplay, int valor, boolean isPontoDecimal ){
  lc.setDigit( 0, indexDisplay, valor, isPontoDecimal );
}

void updatePotencias(){
  analogWrite( PIN_POT_VENTILACAO  , (isPressBotaoStop == false) ? (int)( potenciaVentilacao *
255.0 )   : 0 );
  analogWrite( PIN_POT_UMIDIFICACAO, (isPressBotaoStop == false) ? (int)( potenciaUmidificacao
* 255.0 ) : 0 );
  analogWrite( PIN_POT_AQUECIMENTO , (isPressBotaoStop == false) ? (int)( potenciaAquecimento
* 255.0 )  : 0 );
}

/**
 * Solicita a parada do sistema.
 */
bool stopSistema(){
  if( isPressBotaoStop == false ){
    isPressBotaoStop = true;
    return true;
  }
  else{
    return false;
  }
}

bool updateSetPointTemp( int novoSp ){
  if( abs( (novoSp - sPointTemperatura ) ) == 1 ){
```

```
    sPointTemperatura = novoSp;
    return true;
  }
  else{
    return false;
  }
}

bool updateSetPointUmid( int novoSp ){
  if( abs( (novoSp - sPointUmidade ) ) == 1 ){
    sPointUmidade = novoSp;
    return true;
  }
  else{
    return false;
  }
}

/**
 * Atualiza os valores armazenados na EEPROM.
 */
void updateEeprom( float temperaturaAnterior, float umidadeAnterior, float temperatura, float
umidade ){
  if( temperatura != temperaturaAnterior ){
    nonVolatileNumber.writeInt( EEPROM_ADDRESS_TEMPERATURA, temperatura );
  }

  if( umidade != umidadeAnterior ){
    nonVolatileNumber.writeInt( EEPROM_ADDRESS_UMIDADE, umidade );
  }
}

// ***** UTILITARIOS *****//
int getUnidade( int num ){
  return ( num % 100 ) % 10;
}

int getDezena( int num ){
  return ( num % 100 ) / 10;
}
</code>
```

3.2 Código-fonte “Constantes.c”

Código-fonte utilizado pelo programa `interface_sh_vrs01.ino` para armazenar as constantes utilizadas.

```
<code>
/**
```

```
  * Pinagem
  * 2..............: DHT
  * 3.......~......: Display - DIN
  * 4..............: Display - LOAD
  * 5.......~......: Display - CLK
  * 6.......~......:
  * 7..............:
  * 8..............:
  * 9.......~......: Atuador - Ventilacao
  * 10......~......: Atuador - Umidificacao (UNO)
  * 11......~......: Atuador - Aquecimento
  * 12.............:
  * 13.............: LED indicador de processamento
  *
  * A0.............: Botão - Temperatura incremento
  * A1.............: Botão - Temperatura decremento
  * A2.............: Botão - Umidade incremento
  * A3.............: Botão - Umidade decremento
  * A4.............: Botão - Stop
  * A5.............:
  *
  * DAC0...........:
  * DAC1...........: Atuador - Umidificacao (DUE)
  */


//Sensor de termohigrometria
#define DHT_PIN      2
#define DHT_TYPE     DHT22

//Displays
#define PIN_DIN      3
#define PIN_LOAD     4
#define PIN_CLK      5

#define QTD_DISPLAYS 8

#define DIG_DISP_T_DEZENA     0
#define DIG_DISP_T_UNIDADE    1
#define DIG_DISP_U_DEZENA     2
#define DIG_DISP_U_UNIDADE    3
#define DIG_DISP_SP_T_DEZENA  4
#define DIG_DISP_SP_T_UNIDADE 5
#define DIG_DISP_SP_U_DEZENA  6
#define DIG_DISP_SP_U_UNIDADE 7

//Botoes
#define PIN_BOTAO_T_INCR   A0
#define PIN_BOTAO_T_DECR   A1
#define PIN_BOTAO_U_INCR   A2
#define PIN_BOTAO_U_DECR   A3
```

```
#define PIN_BOTAO_STOP     A4

//Potencias
#define PIN_POT_VENTILACAO   9
#if defined (__arm__) && defined (__SAM3X8E__)
  #define PIN_POT_UMIDIFICACAO DAC1
#else
  #define PIN_POT_UMIDIFICACAO 10
#endif
#define PIN_POT_AQUECIMENTO  11

//Tempos
#define DELAY_INIT_SIS           250    //Tempo (ms) mínimo de duração de cada valor
apresentado no display na inicialização do sistema.
#define DELAY_LOOP               1      //Tempo (ms) de parada após o final de um loop.
#define DELAY_READ               1      //Tempo (ms) de parada após verificar que houve o
recebimento de dado via Serial.
#define SERIAL_BAUD              115200
#define SERIAL_TIMEOUT_READ_MS   2000   //Tempo (ms) máximo de espera pela chegada de dados
pela Serial.
#define SERIAL_TIMEOUT_INCOMING  5     //Em ms. Ver
https://www.arduino.cc/en/Serial/SetTimeout

//Led
#define PIN_PROCESSANDO 13

//EEPROM
#define EEPROM_ADDRESS_STATUS_SIS  0    //1: EEPROM gravada com sucesso.
#define EEPROM_ADDRESS_TEMPERATURA 4
#define EEPROM_ADDRESS_UMIDADE     8
#define EEPROM_STATUS_PRONTO       1    //Valor que identifica que a EEPROM já foi preparada
em execução anterior.

//Valores default
#define TEMPERATURA_MIN_VALIDA     0
#define TEMPERATURA_MAX_VALIDA     99
#define UMIDADE_MIN_VALIDA         0
#define UMIDADE_MAX_VALIDA         99
#define TEMPERATURA_DEFAULT        28
#define UMIDADE_DEFAULT            38
#define VALOR_IDENTIFICADOR_ERRO   8
</code>
```