Capítulo II – Simulação e aplicação da Lógica Fuzzy através do SCILAB e Arduino

Wagner De Lima Santos[1]
John Everson Rodriguo De Mello Jayme [2]
Fabio Oswaldo De Souza Koyano Filho [3]
Wesley Candido da Silva[4]
Carolina Alves Bianchini[5]
Rodolfo Alexandre Hildebrandt[6]

RESUMO

A Lógica Difusa, também conhecida como Lógica Fuzzy, é uma teoria que surgiu em 1965 com a proposta de preencher uma lacuna entre a comunicação humana e os sistemas computacionais. Utilizando conceitos da Teoria dos Conjuntos Difusos, a Lógica Fuzzy permite representar incertezas e imprecisões de forma mais próxima à forma como os seres humanos lidam com informações. Através das funções de pertinência, é possível mensurar em valores adjetivos que antes eram considerados subjetivos, como "muito alto" ou "pouco quente". O artigo apresenta um passo a passo para a simulação da Lógica Fuzzy no software Scilab, bem como uma explicação detalhada sobre um código em Arduino que utiliza a Lógica Fuzzy para controlar a luminosidade de um LED, utilizando um sensor LDR. A aplicação prática da Lógica Fuzzy em Arduino demonstra como a teoria pode ser utilizada em situações reais, permitindo um maior controle sobre sistemas complexos e incertos. Em suma, o artigo aborda tanto a teoria quanto a prática da Lógica Fuzzy, mostrando sua importância na solução de problemas que envolvem incertezas e imprecisões. Com a apresentação dos conceitos e técnicas envolvidos na Lógica Fuzzy, almeja-se capacitar os leitores para aplicar essa técnica tão relevante e importante na área da computação e automação em seus projetos, explorando todo o seu potencial e proporcionando soluções mais precisas e eficientes.

Palavras-chave: Lógica Fuzzy. Simulação. Aplicação Prática.

1 INTRODUÇÃO

A Lógica Fuzzy, também conhecida como Lógica Difusa, tem como objetivo modelar modos de raciocínio aproximados em vez de precisos, incorporando a inteligência humana à arquitetura computacional. Diferente da lógica binária, que considera apenas proposições

[1] Estudante Engenharia Elétrica UniSENAIPR -Campus Londrina, delimasantoswagner@gmail.com
[2] Estudante Engenharia Elétrica UniSENAIPR -Campus Londrina, jonhermj@hotmail.com
[3] Estudante Engenharia Elétrica UniSENAIPR -Campus Londrina, fabiokoyanofilho@outlook.com
[4] Docente UniSenaiPR - Campus Londrina, wesley.candido@sistemafiep.org.br
[5] Docente UniSenaiPR - Campus Londrina, carolina.bianchini@sistemafiep.org.br
[6] Docente UniSenaiPR - Campus Londrina, rodolfo.hildebrandt@sistemafiep.org.br

verdadeiras ou falsas, a Lógica Fuzzy permite valores intermediários entre essas duas possibilidades, ou seja, a veracidade das proposições pode variar entre 0 e 1, estabelecendo graus de pertinência. Na prática, a Lógica Fuzzy é capaz de lidar com problemas complexos que não podem ser representados pela lógica clássica, com seus casos precisos funcionando como situações limite.

Essa Lógica é uma técnica matemática utilizada para lidar com a incerteza presente em muitos problemas complexos do mundo real. Ela se baseia na teoria dos conjuntos fuzzy, onde cada elemento possa ter uma pertinência parcial em um conjunto. Com a evolução da tecnologia, sistemas complexos e inteligentes têm sido desenvolvidos para solucionar problemas em diversas áreas. Nesse contexto, a Lógica Fuzzy tem sido cada vez mais utilizada, pois permite uma análise mais precisa de sistemas não-lineares, não-determinísticos e imprecisos.

Este artigo tem como objetivo fornecer uma introdução detalhada sobre a Lógica Fuzzy apresentando uma aplicação prática em um sistema embarcado utilizando o Arduino, onde foi realizado o controle de um LED a partir da leitura de um sensor LDR, responsável por captar a luminosidade do ambiente. Será apresentado um passo a passo de como simular a Lógica Fuzzy no software Scilab, bem como a explicação do código desenvolvido para o Arduino. Com isso, espera-se que o leitor possa compreender os principais conceitos e técnicas envolvidos na Lógica Fuzzy e utilizá-los em seus próprios projetos e aplicações.

A utilização dessa lógica tem se destacado por sua flexibilidade e capacidade de modelagem em situações complexas, apresentando resultados mais precisos e próximos da realidade. A aplicação desenvolvida neste artigo permite uma melhor compreensão do funcionamento dessa lógica e de como ela pode ser implementada em sistemas embarcados, mostrando a importância de seu uso na resolução de problemas.

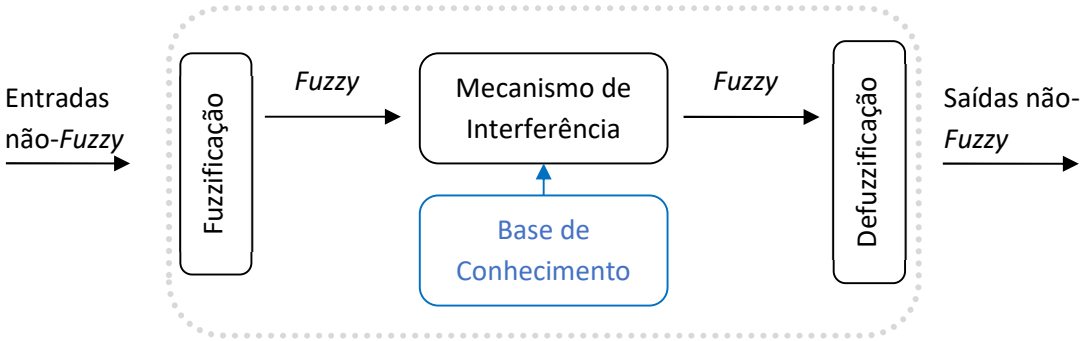
2 MATERIAIS E MÉTODOS

A lógica fuzzy permite representar e manipular informações imprecisas ou incertas através de valores numéricos fuzzy, que são transformados por meio da fuzzificação e defuzzificação. A fuzzificação é o processo de mapear uma variável de entrada em um conjunto de valores fuzzy. Isso significa que a variável é transformada em um conjunto de valores que representam o grau de pertinência da variável a diferentes categorias. Por

exemplo, se a variável de entrada for a temperatura de um ambiente, ela pode ser fuzzificada em conjuntos como "frio", "morno" ou "quente".

A defuzzificação é o processo oposto à fuzzificação. Ela consiste em transformar o conjunto de valores fuzzy em um valor numérico único, que representa o grau de pertinência da variável a uma categoria específica. Isso é feito para que o resultado da lógica fuzzy possa ser utilizado em sistemas de controle ou tomadas de decisão.

Figura 1 – Representação Lógica Fuzzy



Fonte: Dos Autores.

2.1 SIMULAÇÃO DE UMA LÓGICA FUZZY NO SOFTWARE SCILAB V.6.1.1

Para representar a aplicação da lógica fuzzy, será realizada uma implementação no software Scilab V.6.1.1, utilizando a extensão sciFLT. Essa implementação irá envolver um controle simples de tensão em um LED, que será controlado de acordo com a iluminação do ambiente, classificando a iluminação em quatro categorias: BAIXO, MÉDIO BAIXO, MÉDIO ALTO e ALTO. A resposta do sistema será dada em três categorias de tensão no LED: BAIXA, MÉDIA e ALTA. Posteriormente, essa lógica será aplicada de forma prática, utilizando um Arduino programado com base nos componentes físicos utilizados na protoboard.

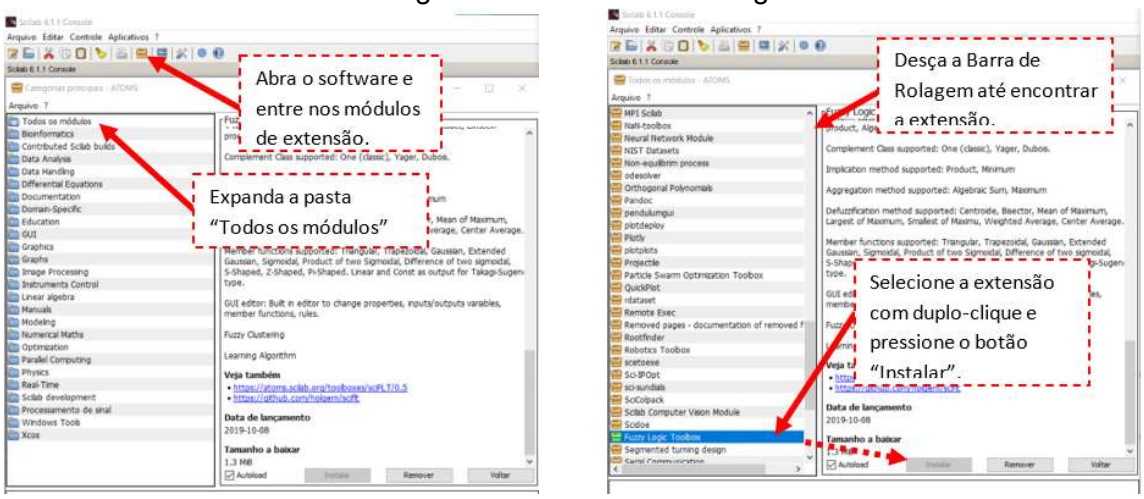
Figura 2 - Software Scilab V.6.1.1

Fonte: Dos Autores.

2.1.1 INSTALAÇÃO E CONFIGURAÇÃO DA FERRAMENTA

Após a instalação do software, é necessário realizar a configuração da ferramenta computacional para suportar a lógica Fuzzy. Para isso, será necessário adicionar a extensão “Fuzzy Logic Toolbox” ao programa, de acordo com a Figura 3.

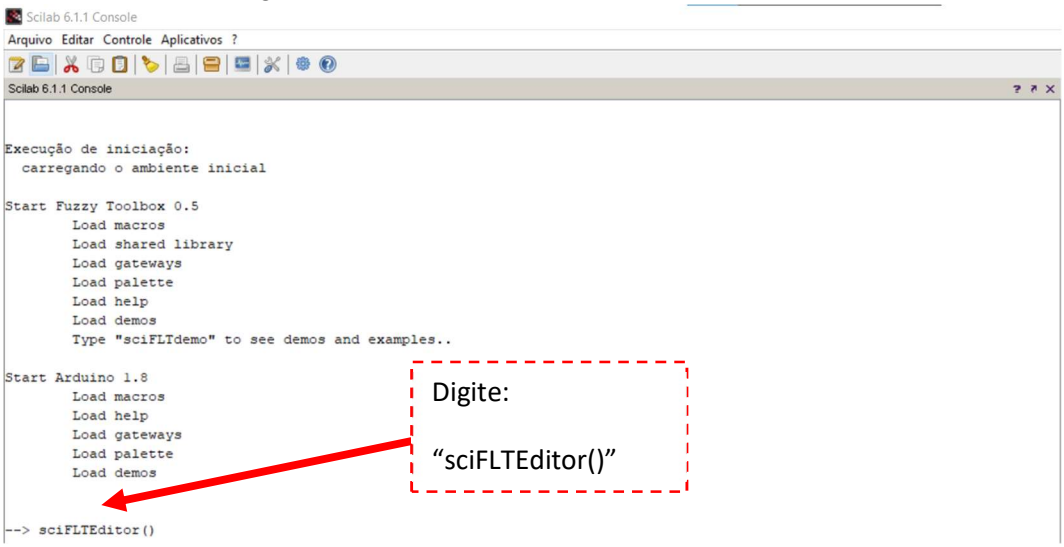
Figura 3 – Exibindo as Categorias



Fonte: Dos Autores.

Com a instalação da extensão "Fuzzy Logic Toolbox" realizada, acesse o Console do Scilab e execute o comando "sciFLTEditor()" de acordo com as convenções de maiúsculas e minúsculas. Esse comando é responsável por inicializar o "sciFLT fls Editor", uma ferramenta que permite a construção de sistemas de lógica fuzzy.

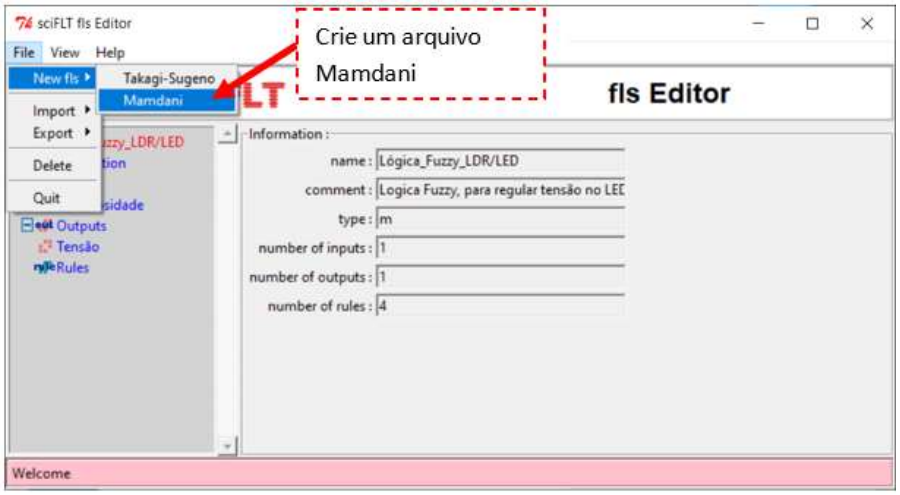
Figura 4 – Abrindo a ferramenta sciFLT Editor



Fonte: Dos Autores.

Ao inicializar a ferramenta "sciFLT fls Editor", podemos proceder com a criação de um novo arquivo, o qual será utilizado para simular a lógica fuzzy desejada. Para tanto, devemos acessar o menu "File" > "New fls" > "Mandani", a fim de selecionar a opção de criação de um novo arquivo do tipo Mandani.

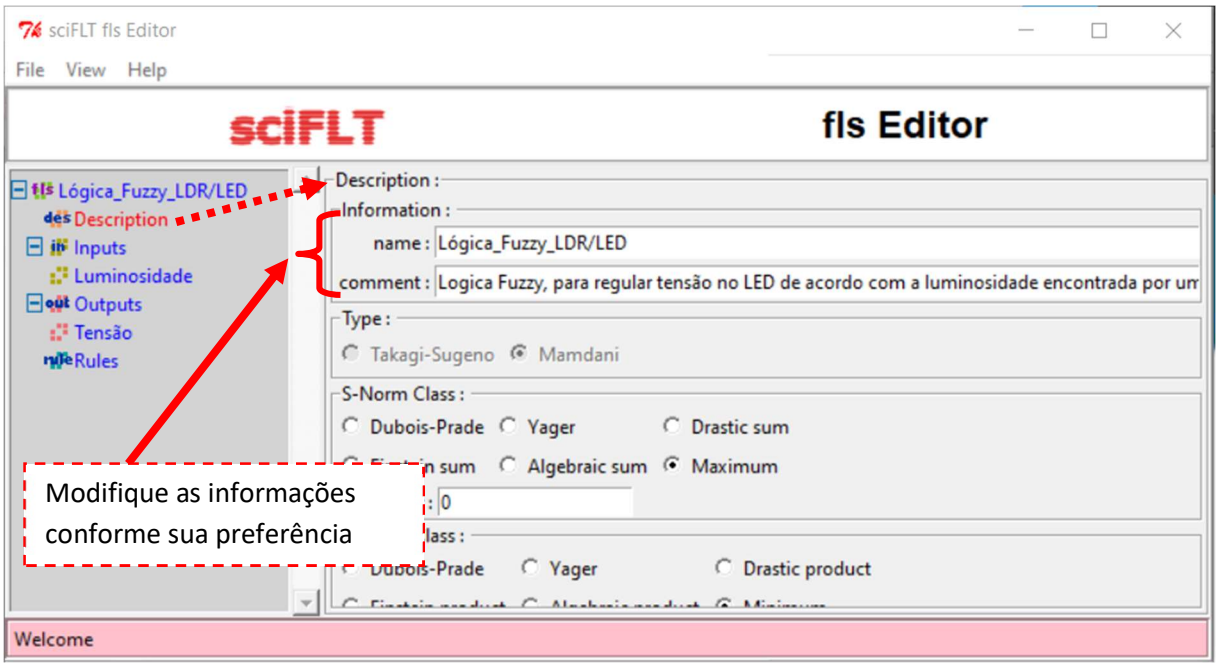
Figura 5 – Criando arquivo



Fonte: Dos Autores.

Após a criação do arquivo de simulação da lógica fuzzy, é possível identificá-lo por meio da alteração do seu nome e comentário, caso necessário.

Figura 6 – Identificando o Projeto



Fonte: Dos Autores.

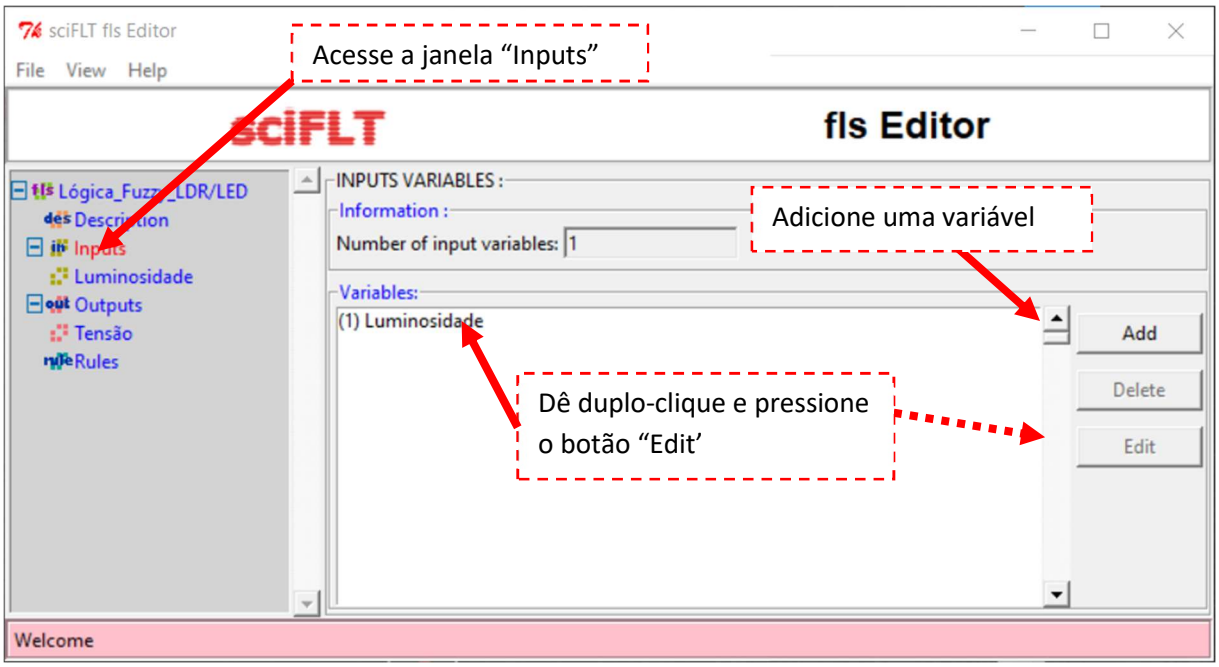
Uma das etapas fundamentais da lógica Fuzzy é o processo de Fuzzificação, o qual consiste em transformar as entradas do sistema em variáveis linguísticas, estabelecendo uma correlação entre as entradas e um conjunto de valores. Após o processo e Fuzzificação configuramos as saidas conforme o processo de Defuzzificação. Para correlacionar as entradas com a saída, regras de inferência são necessárias, as quais serão configuradas em sequência.

2.1.2 CONFIGURAÇÃO DOS PARÂMETROS DE ENTRADA

Para cada entrada, é necessário configurar o parâmetro do range de atuação, que é composto por um intervalo de valores nítidos, permitindo, dessa forma, a mensuração das entradas. Com base na base de conhecimento do problema proposto, é atribuído a cada entrada um subconjunto denominado conjunto difuso, o qual é definido pelos parâmetros do range de atuação e da função de pertinência.

Para fazer a configuração das entradas, precisamos acessar a janela “Inputs” e adicionar todas entradas do problema proposto, para esse exemplo adicionaremos apenas uma entrada “Luminosidade” que representará a luminosidade encontrada no ambiente. Ao adicionar a entrada, selecionamos a mesma e editamos os valores dando duplo-clique e clicando no botão “Edit” que nos levará até o campo de edição da variável.

Figura 7 – Adicionando Variáveis de Entrada

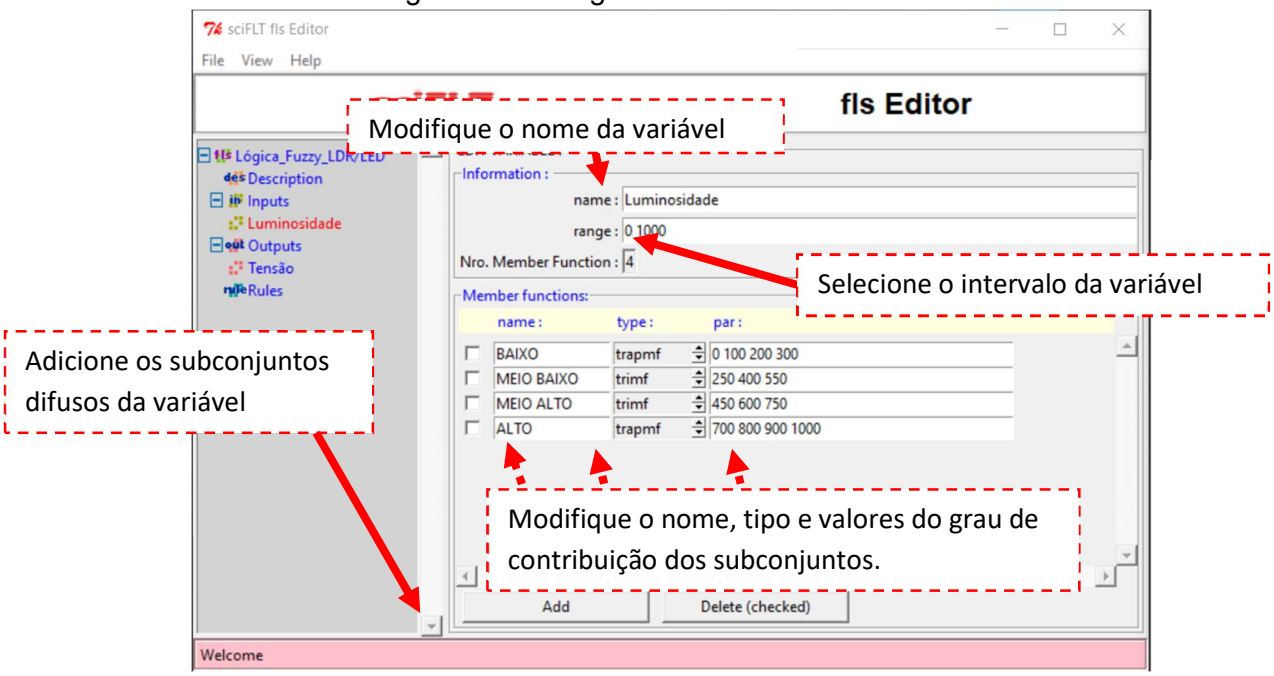


Fonte: Dos Autores.

Nesta janela, você poderá acionar quantas variáveis de entrada você julgar necessário para simular o problema proposto. Cada variável deverá ser configurada individualmente com base na base de conhecimento do seu processo, definindo parâmetros para os conjuntos difusos.

Na janela de edição da variável de entrada, foi realizada a configuração do intervalo de valores nítidos, permitindo a mensuração da entrada "Luminosidade" em um intervalo de 0 a 1000. De acordo com a base de conhecimento do problema proposto, foram atribuídos quatro subconjuntos à entrada criada, denominados conjuntos difusos, que são definidos pelos parâmetros do range de atuação e da função de pertinência.

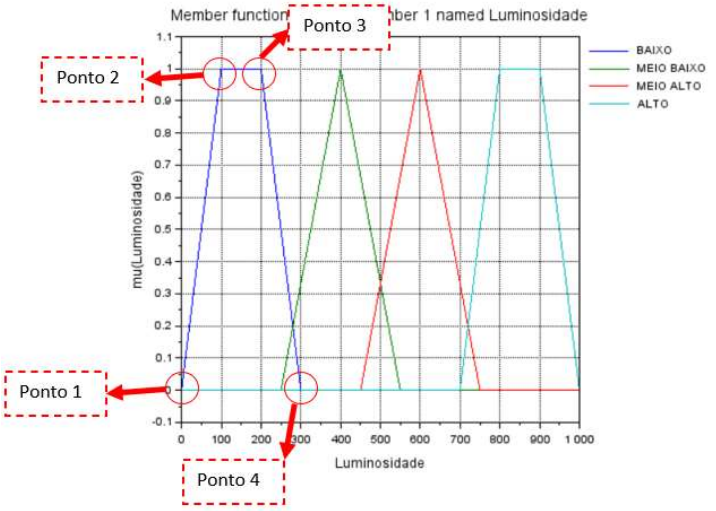
Figura 8 – Configurando Variável de Entrada



Fonte: Dos Autores.

Os subconjuntos "BAIXO" e "ALTO" foram identificados na visualização dos conjuntos difusos, sendo que o conjunto difuso correspondente recebeu uma função de pertinência trapmf (função trapezoidal), já os subconjuntos "MEIO BAIXO" e "MEIO ALTO" receberam uma função de pertinência trimf (função triangular). Para cada função de pertinência, é determinado um conjunto de valores no qual é medido seu grau de contribuição e a quantidade de pontos a serem inseridos no subconjunto. Por exemplo, uma função triangular requer três pontos, enquanto a trapezoidal requer quatro pontos.

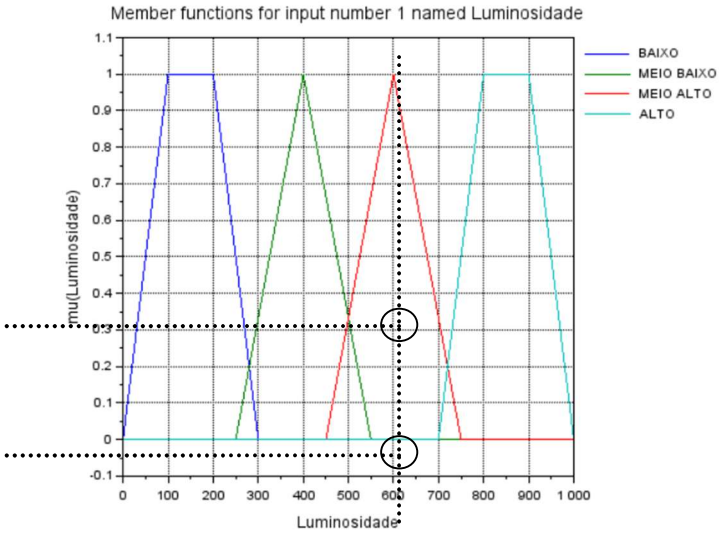
Figura 9 – Pontos de um Subconjunto do Conjunto “Luminosidade”



Fonte: Dos Autores.

Podemos observar o conjunto difuso da variável “Luminosidade” por meio de um gráfico no qual representa visualmente o grau de pertinência da função. Onde no eixo Y temos o grau de pertinência representando com um range de 0:1, e no eixo X o range de entrada da Luminosidade conforme a configuração previamente realizada de 0:1000. Para visualizarmos o contexto do conjunto difuso, utilizamos a variável de entrada “Luminosidade” plotando um gráfico e posteriormente traçamos uma reta horizontal afim de observar as funções de pertinência que influenciarão na saída no instante desejado, conforme a Figura 10.

Figura 10 – Conjunto Difuso “Luminosidade”



Fonte: Dos Autores.

Foi selecionado o instante de Luminosidade de 740 para exemplificar, e assim, observamos que nesse instante a reta cruzou as funções de pertinência “MEIO ALTO” e “ALTO” e assim

podemos visualizar o grau de pertinência de cada funções, nitidamente notamos que o grau de influência na resposta será maior da função “ALTO” (próximo a 0,4 ou 40%) do que da função “MEIO ALTO” (próximo a 0,08 ou 8%) contudo será considerada as duas funções para a resposta neste instante.

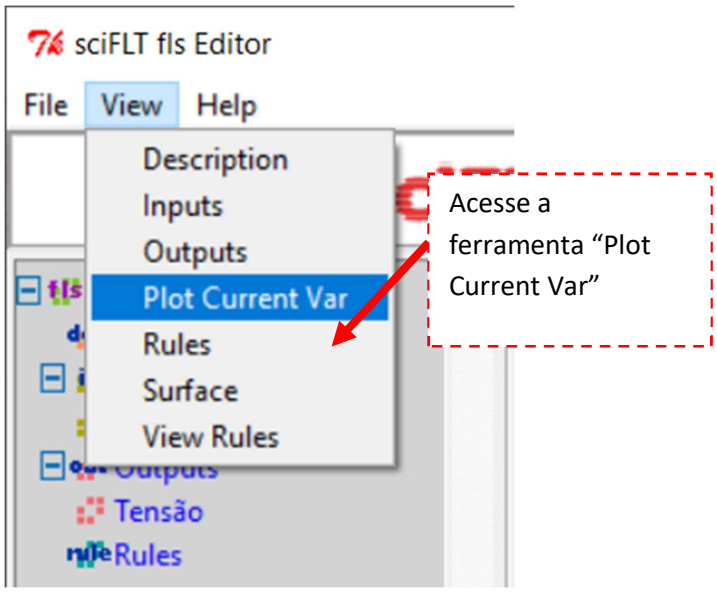
2.1.3 EXIBIÇÃO DO GRÁFICO DAS VÁRIAVEIS

Para exibir o gráfico das variáveis (tanto variável de entradas quanto variáveis de saídas), selecionamos a variável desejada e em seguida no menu “View” selecionamos a ferramenta “Plot Currente Var” para exibir o gráfico da variável selecionada.

Figura 11 – Selecionando a Variável para Exibir o Gráfico

Fonte: Dos Autores.

Figura 12 – Acessando a Ferramenta “Plot Currente Var”



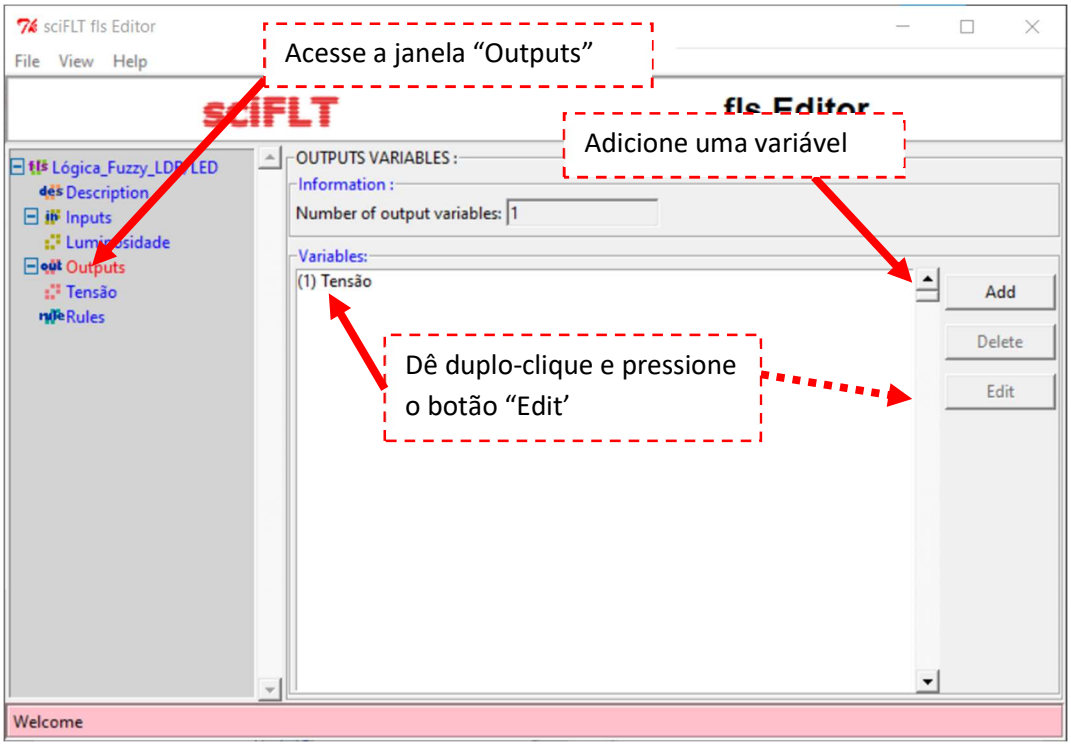
Fonte: Dos Autores.

2.1.4 CONFIGURAÇÃO DOS PARÂMETROS DE SAÍDA.

Para cada saída, também é necessário configurar o parâmetro do range de atuação, composto por um intervalo de valores nítidos. Com base na base de conhecimento do problema proposto, atribuímos um subconjunto (conjunto difuso) para cada saída.

Para fazer a configuração das saídas seguimos os mesmos passos utilizados nas configurações das entradas, acessando a janela “Outputs” e adicionar todas saídas do problema proposto, para esse exemplo adicionaremos apenas uma saída “Tensão” que representará o nivel de tensão aplicado no LED. Ao adicionar a saída, selecionamos a mesma e editamos os valores dando duplo-clique e clicando no botão “Edit” que nos levará até o campo de edição da variável.

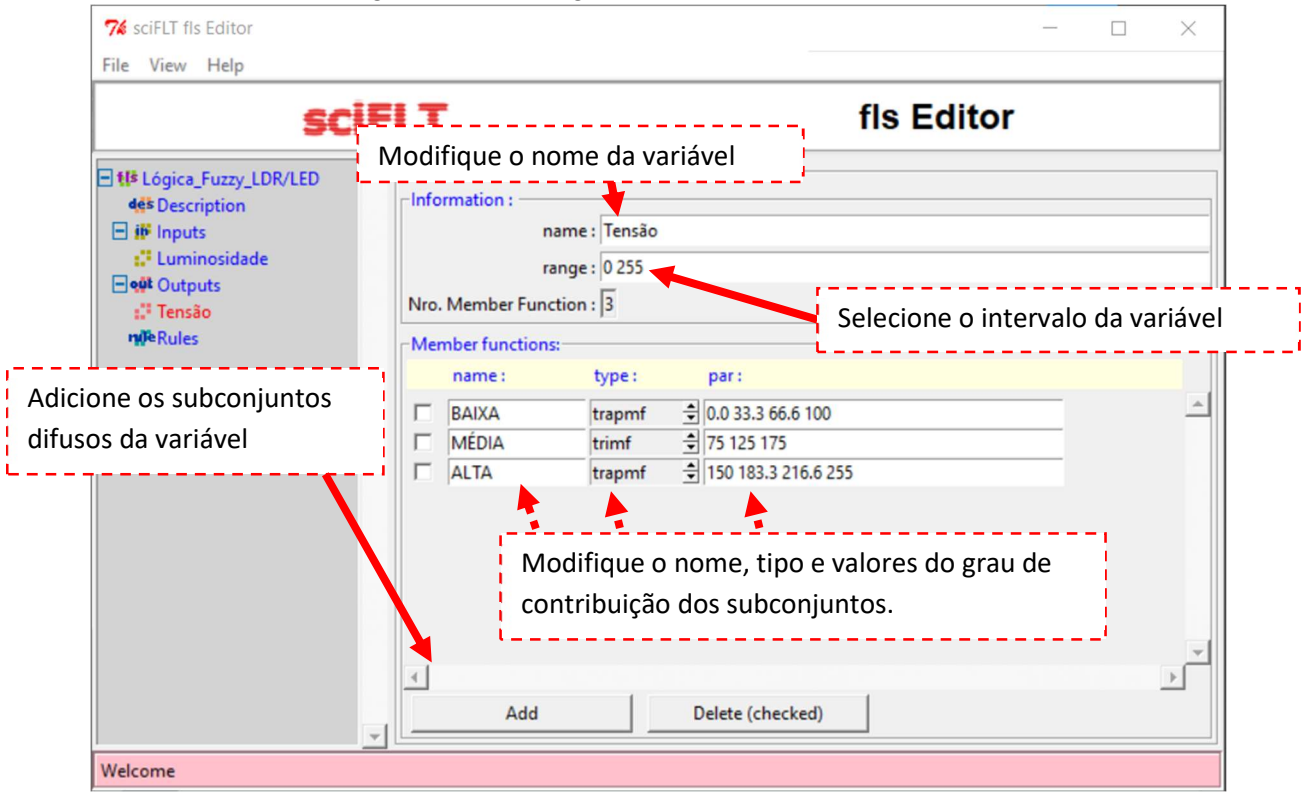
Figura 13 – Adicionando Variáveis de Saída



Fonte: Dos Autores.

Similar as variáveis de entrada, na janela das variáveis de saída, também possibilitam a criação de diversas variáveis, para exemplificar seu problema da melhor forma. Também se faz necessário uma configuração individual para cada variável conforme a base de conhecimentos do seu processo.

Figura 14 – Configurando Variáveis de Saída



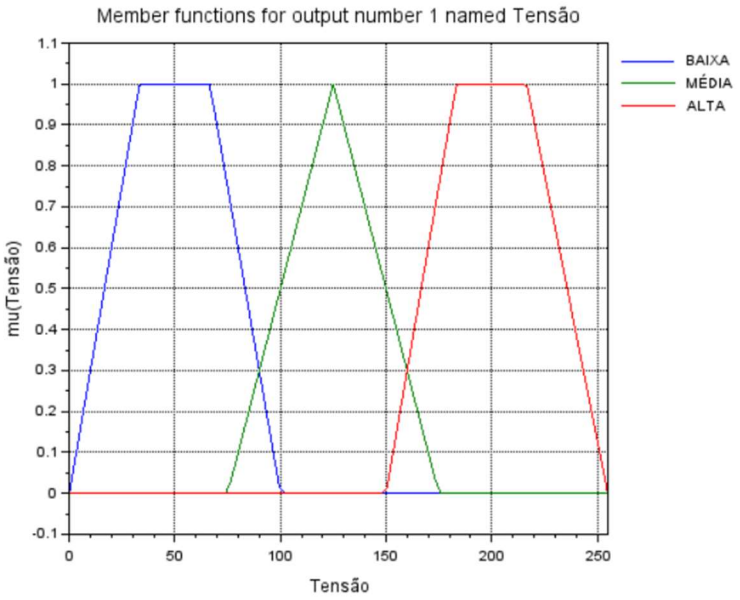
Fonte: Dos Autores.

Na janela de edição da variável de saída, foi realizada a configuração do intervalo de valores nítidos, permitindo a mensuração da saída "Tensão" em um intervalo de 0 a 255, o que não representa um sinal de tensão de 0 a 255 volts, e sim, o sinal poderá ser enviado a um microcontrolador fazer os devidos controles de tensão.

De acordo com a base de conhecimento do problema proposto, foram atribuídos três conjuntos difusos à saída criada. Os subconjuntos "BAIXA" e "ALTA" receberam uma função de pertinência trapmf (função trapezoidal), já o subconjunto "MÉDIA" recebeu uma função de pertinência trimf (função triangular). Sendo assim, temos 4 pontos para os subconjuntos “BAIXA” e “ALTA” e 3 pontos para o subconjunto “MÉDIA”.

Utilizando a ferramenta “Plot Currente Var” conseguimos exibir o gráfico da variável de saída no qual facilita a configuração dos valores do grau de contribuição para cada subconjunto.

Figura 15 – Conjunto Difuso de Saída “Tensão”



Fonte: Dos Autores.

Podemos observar o conjunto difuso da variável “Saida” conforme o gráfico acima no qual representa visualmente o grau de pertinência de cada subconjunto da função. Onde no eixo Y temos o grau de pertinência representando com um range de 0:1, e no eixo X o range de entrada da Tensão conforme a configuração previamente realizada de 0:255.

2.1.5 DEFINIÇÃO DAS REGRAS, CORRELAÇÃO ENTRE ENTRADA E SAÍDA.

As devidas correlações entre as entradas e as saídas devem ser feitas de acordo com o estabelecimento de algumas regras, essas regras devem ser seguidas conforme a base de conhecimento do processo. Se uma determinada ação for correspondida, a saída estará de acordo com a sua regra criada para ela.

Para criar essas regras, devemos acessar a janela “Rules” que possibilitará a criação das regras de correlação. Inicialmente precisamos selecionar na variável de entrada (destacada no software pela cor azul) qual subconjunto será atribuído a essa regra e posteriormente selecionar na variável de saída (destacada no software pela cor roxa) qual subconjunto será atribuído a essa regra. Temos a opção também de selecionar, quando mais de uma variável de entrada, a opção “AND (traduzindo para o português: “E”) ou OR (traduzindo: “OU”) para assim, trazer várias opções de correlações para as entradas com a saída.

Figura 16 – Configuração das Regras de Correlação

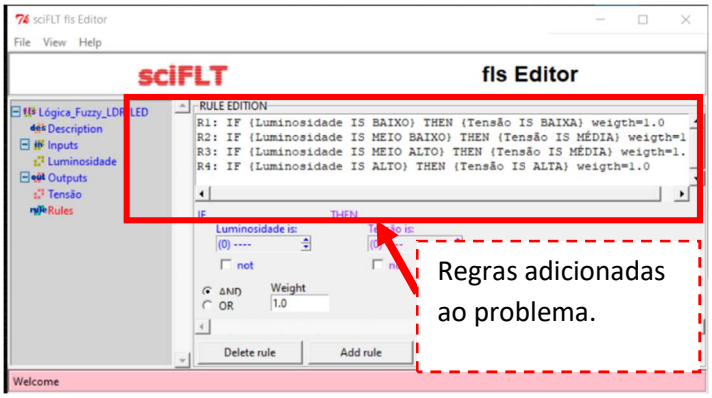
Fonte: Dos Autores.

Para essa situação problema, utilizamos apenas 4 regras de decisão, sendo elas:

- **R1**: Se “Luminosidade” for “*BAIXO*” então “Tensão” será “*BAIXA*;
- **R2**: Se “Luminosidade” for “*MEIO BAIXO*” então “Tensão” será “*MÉDIA*;
- **R3**: Se “Luminosidade” for “*MEIO ALTO*” então “Tensão” será “*MÉDIA*;
- **R4**: Se “Luminosidade” for “*ALTO*” então “Tensão” será “*ALTA*;

O modelo pode contemplar a quantidade de regras necessária para se aproximar da base de conhecimento existente do problema.

Figura 17 – Regras de Correlação



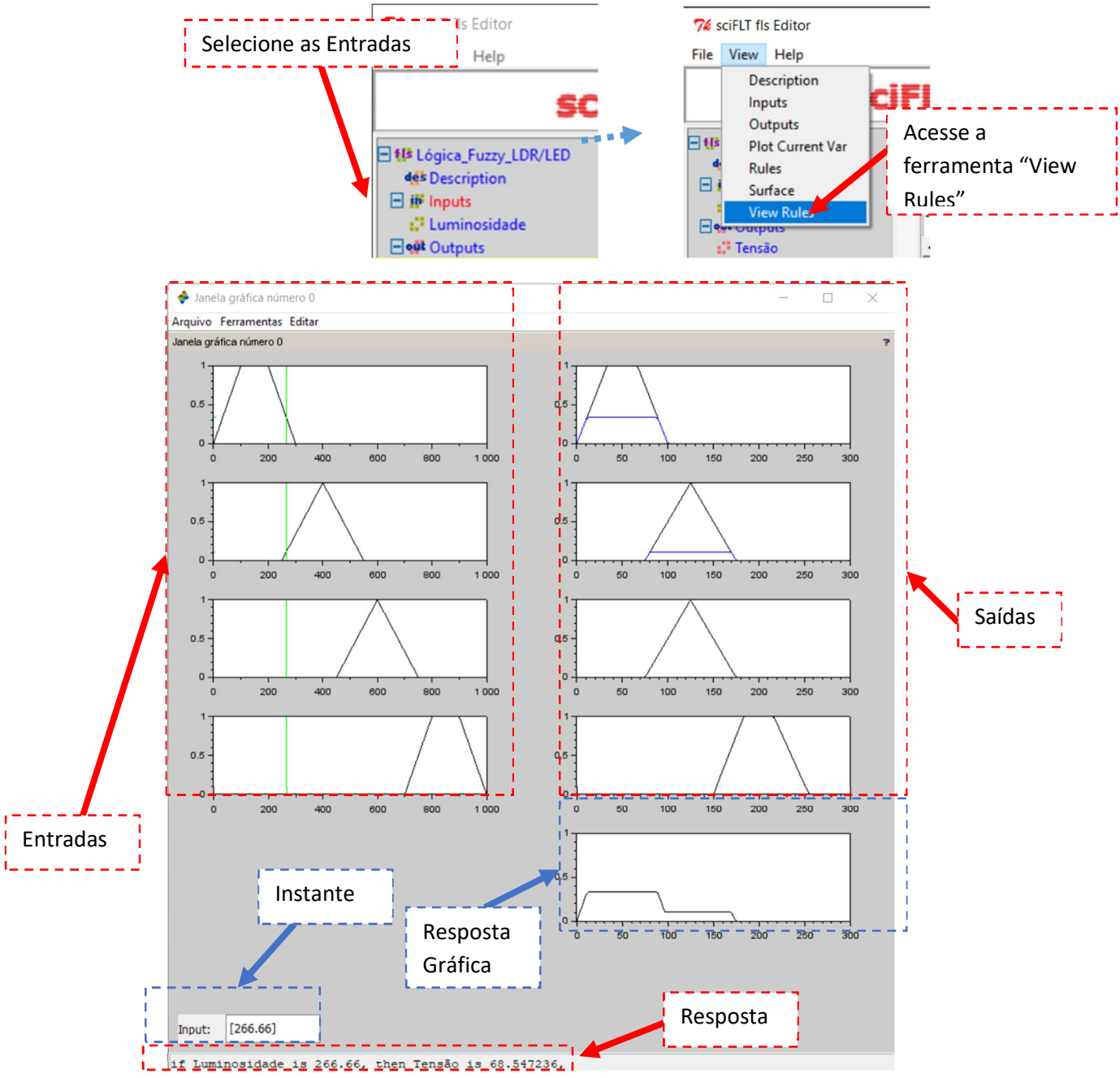
Fonte: Dos Autores.

2.1.6 EXIBIÇÃO DOS GRÁFICOS DAS REGRAS.

Com a definição dos parâmetros das entradas, saídas e regras; o próximo passo é analisar a regras. Para exibir os gráficos das regras, com as variaveis de entradas

selecionada, no menu “View” selecionamos a ferramenta “View Rules” e assim será exibida a janela gráfica.

Figura 18 – Acessando a Ferramenta “View Rules” e Exibição da Janela Gráfica



Fonte: Dos Autores.

2.2 EXPLICAÇÃO DE UMA LOGICA FUZZY INTRODUZIA NO ARDUINO.

Este código implementa um sistema de lógica fuzzy para controlar a velocidade de um objeto com base em sua distância de um obstáculo. A lógica fuzzy é uma técnica de inteligência artificial que lida com a incerteza e a imprecisão em sistemas de controle.

Tabela 2 – Código Proposto para Análise

Aplicação de uma Logica Fuzzy em Arduino

```
#include <Fuzzy.h>


// Instanciando um objeto Fuzzy
Fuzzy *fuzzy = new Fuzzy();


void setup()
{
  // Configurando a saída Serial
  Serial.begin(9600);
  // Definindo uma semente aleatória
  randomSeed(analogRead(0));


  // Instanciando um objeto FuzzyInput
  FuzzyInput *distance = new FuzzyInput(1);
  // Instanciando um objeto FuzzySet
  FuzzySet *small = new FuzzySet(0, 20, 20, 40);
  // Incluindo o FuzzySet no FuzzyInput
  distance->addFuzzySet(small);
  // Instanciando um objeto FuzzySet
  FuzzySet *safe = new FuzzySet(30, 50, 50, 70);
  // Incluindo o FuzzySet no FuzzyInput
  distance->addFuzzySet(safe);
  // Instanciando um objeto FuzzySet
  FuzzySet *big = new FuzzySet(60, 80, 80, 80);
  // Incluindo o FuzzySet no FuzzyInput
  distance->addFuzzySet(big);
  // Incluindo o FuzzyInput no Fuzzy
  fuzzy->addFuzzyInput(distance);


  // Instanciando um objeto FuzzyOutput
  FuzzyOutput *speed = new FuzzyOutput(1);
  // Instanciando um objeto FuzzySet
  FuzzySet *slow = new FuzzySet(0, 10, 10, 20);
  // Incluindo o FuzzySet no FuzzyOutput
  // Incluindo a FuzzyRule no Fuzzy
  fuzzy->addFuzzyRule(fuzzyRule01);


  // Criando a regra "SE a distância é segura ENTÃO a velocidade é média"
  // Instanciando um objeto FuzzyRuleAntecedent
  FuzzyRuleAntecedent *ifDistanceSafe = new FuzzyRuleAntecedent();
  // Criando um FuzzyRuleAntecedent com apenas um FuzzySet
  ifDistanceSafe->joinSingle(safe);
  // Instanciando um objeto FuzzyRuleConsequent
  FuzzyRuleConsequent *thenSpeedAverage = new FuzzyRuleConsequent();
  // Incluindo um FuzzySet neste FuzzyRuleConsequent
  thenSpeedAverage->addOutput(average);
  // Instanciando um objeto FuzzyRule
  FuzzyRule *fuzzyRule02 = new FuzzyRule(2, ifDistanceSafe, thenSpeedAverage);
  // Incluindo a FuzzyRule no Fuzzy
  fuzzy->addFuzzyRule(fuzzyRule02);


  // Criando a regra "SE a distância é grande ENTÃO a velocidade é alta"
  // Instanciando um objeto FuzzyRuleAntecedent
  FuzzyRuleAntecedent *ifDistanceBig = new FuzzyRuleAntecedent();
  // Criando um FuzzyRuleAntecedent com apenas um FuzzySet
  ifDistanceBig->joinSingle(big);
  // Instanciando um objeto FuzzyRuleConsequent
  FuzzyRuleConsequent *thenSpeedFast = new FuzzyRuleConsequent();
  // Incluindo um FuzzySet neste FuzzyRuleConsequent
  thenSpeedFast->addOutput(fast);
  // Instanciando um objeto FuzzyRule
  FuzzyRule *fuzzyRule03 = new FuzzyRule(3, ifDistanceBig, thenSpeedFast);
  // Incluindo a FuzzyRule no Fuzzy
  fuzzy->addFuzzyRule(fuzzyRule03);
}
```

```
  speed->addFuzzySet(slow);

  // Instanciando um objeto FuzzySet

  FuzzySet *average = new FuzzySet(10, 20, 30, 40);

  // Incluindo o FuzzySet no FuzzyOutput

  speed->addFuzzySet(average);

  // Instanciando um objeto FuzzySet

  FuzzySet *fast = new FuzzySet(30, 40, 40, 50);

  // Incluindo o FuzzySet no FuzzyOutput

  speed->addFuzzySet(fast);

  // Incluindo o FuzzyOutput no Fuzzy

  fuzzy->addFuzzyOutput(speed);



  // Criando a regra Fuzzy "SE distância = small ENTÃO velocidade =
slow"

  // Instanciando um objeto FuzzyRuleAntecedent

  FuzzyRuleAntecedent *ifDistanceSmall = new
FuzzyRuleAntecedent();

  // Criando um FuzzyRuleAntecedent com um único FuzzySet

  ifDistanceSmall->joinSingle(small);

  // Instanciando um objeto FuzzyRuleConsequent

  FuzzyRuleConsequent *thenSpeedSlow = new
FuzzyRuleConsequent();

  // Incluindo um FuzzySet neste FuzzyRuleConsequent

  thenSpeedSlow->addOutput(slow);

  // Instanciando um objeto FuzzyRule

  FuzzyRule *fuzzyRule01 = new FuzzyRule(1, ifDistanceSmall,
thenSpeedSlow);
```

```
void loop()

{

  // Obtendo um valor aleatório

  int input = random(0, 80);

  // Imprimindo algo

  Serial.println("\n\n\nEntrance: ");

  Serial.print("\t\t\tDistance: ");

  Serial.println(input);

  // Definindo o valor aleatório como entrada

  fuzzy->setInput(1, input);

  // Executando a Fuzzificação

  fuzzy->fuzzify();

  // Executando a Defuzzificação

  float output = fuzzy->defuzzify(1);

  // Imprimindo algo

  Serial.println("Result: ");

  Serial.print("\t\t\tSpeed: ");

  Serial.println(output);

  // esperando 12 segundos

  delay(12000);

}
```

Fonte: eFLL/arduino_simple_sample.ino at master · alvesoaj/eFLL · GitHub

Na inicialização do programa, são criados dois objetos FuzzyInput e FuzzyOutput, que representam as variáveis de entrada e saída do sistema, respectivamente. Cada objeto é composto por 3 objetos FuzzySet, que definem conjuntos fuzzy que descrevem o comportamento da variável. São criadas então três FuzzySet para o FuzzyInput "distância" e três para o FuzzyOutput "velocidade".

Sendo elas:

FuzzyInput (distance):

small = (0, 20, 20, 40);

safe = (30, 50, 50, 70);

big = (60, 80, 80, 80).

FuzzyOutput (speed):

slow = (0, 10, 10, 20);

average = (10, 20, 30, 40);

fast = (30, 40, 40, 50).

Em seguida, são definidas regras fuzzy que relacionam as variáveis de entrada e saída. As regras fuzzy são definidas pelo objeto FuzzyRule, que contém um FuzzyRuleAntecedent (que descreve as condições da regra) e um FuzzyRuleConsequent (que descreve a ação a ser tomada se a condição for verdadeira). São definidas três regras que relacionam cada FuzzySet do FuzzyInput "distância" a um FuzzySet do FuzzyOutput "velocidade".

Sendo elas:

SE a distância = small ENTÃO a velocidade = slow;

SE a distância = safe ENTÃO a velocidade = média;

SE a distância = big ENTÃO a velocidade = alta.

No loop principal, é gerado um valor aleatório (de 0 a 80) para a entrada e, em seguida, é realizado o processo de fuzzificação, que determina em que grau cada FuzzySet da variável de entrada é verdadeiro para o valor de entrada. Então, é feita a defuzzificação, que determina a saída do sistema com base nas regras definidas e nos graus de verdade dos FuzzySets. O valor da saída é impresso no monitor serial.

A lógica fuzzy tem diversas aplicações em tecnologia, como em sistemas de controle, reconhecimento de padrões, previsão de séries temporais, sistemas especialistas, entre outros. Ela é particularmente útil em sistemas que lidam com imprecisão ou incerteza, onde as regras de controle não são facilmente expressas por meio de lógica booleana.

Tabela 3 – Código Comentado

Aplicação de uma Logica Fuzzy em Arduino Comentada	
#include <Fuzzy.h> Fuzzy *fuzzy = new Fuzzy();	Inicia-se incluindo a biblioteca “Fuzzy.h” responsavel pelas configurações dos parametros de entrada e saida, bem como, as regras respobsaveis pela fuzzificação e deffuzyficação.

```
void setup()

{


  Serial.begin(9600);

  randomSeed(analogRead(0));



  FuzzyInput *distance = new FuzzyInput(1);

  FuzzySet *small = new FuzzySet(0, 20, 20, 40);

  distance->addFuzzySet(small);

  FuzzySet *safe = new FuzzySet(30, 50, 50, 70);

  distance->addFuzzySet(safe);

  FuzzySet *big = new FuzzySet(60, 80, 80, 80);

  distance->addFuzzySet(big);

  fuzzy->addFuzzyInput(distance);
```

Criado um numero aleatorio para a entrada (RandomSeed).

Configurações das entradas conforme conhecimento do processo.

Neste exemplo (Ao lado), temos três entradas:

small = (0, 20, 20, 40);

safe = (30, 50, 50, 70);

big = (60, 80, 80, 80).

```
FuzzyOutput *speed = new FuzzyOutput(1);

  FuzzySet *slow = new FuzzySet(0, 10, 10, 20);

  speed->addFuzzySet(slow);

  FuzzySet *average = new FuzzySet(10, 20, 30, 40);

  speed->addFuzzySet(average);

  FuzzySet *fast = new FuzzySet(30, 40, 40, 50);

  speed->addFuzzySet(fast);

  fuzzy->addFuzzyOutput(speed);
```

Configurações das saídas conforme conhecimento do processo.

Neste exemplo (Ao lado), temos três saídas:

slow = (0, 10, 10, 20);

average = (10, 20, 30, 40);

fast = (30, 40, 40, 50).

```
  FuzzyRuleAntecedent *ifDistanceSmall = new
FuzzyRuleAntecedent();

  ifDistanceSmall->joinSingle(small);

  FuzzyRuleConsequent *thenSpeedSlow = new
FuzzyRuleConsequent();

  thenSpeedSlow->addOutput(slow);

  FuzzyRule *fuzzyRule01 = new FuzzyRule(1, ifDistanceSmall,
thenSpeedSlow);

fuzzy->addFuzzyRule(fuzzyRule01);




  FuzzyRuleAntecedent *ifDistanceSafe = new FuzzyRuleAntecedent();

  ifDistanceSafe->joinSingle(safe);
```

Definição das Regras do sistema Fuzzy, de acordo com os parâmetros de entrada com os parâmetros de saída.

Neste exemplo (Ao lado), temos três regras:

- SE a distância = small ENTÃO a velocidade = slow;
- SE a distância = safe ENTÃO a velocidade = média;
- SE a distância = big ENTÃO a velocidade = alta.

FuzzyRuleConsequent *thenSpeedAverage = new FuzzyRuleConsequent();
 	
thenSpeedAverage->addOutput(average);
 	
FuzzyRule *fuzzyRule02 = new FuzzyRule(2, ifDistanceSafe, thenSpeedAverage);
 	
fuzzy->addFuzzyRule(fuzzyRule02);
 	
FuzzyRuleAntecedent *ifDistanceBig = new FuzzyRuleAntecedent();
 	
ifDistanceBig->joinSingle(big);
 	
FuzzyRuleConsequent *thenSpeedFast = new FuzzyRuleConsequent();
 	
thenSpeedFast->addOutput(fast);
 	
FuzzyRule *fuzzyRule03 = new FuzzyRule(3, ifDistanceBig, thenSpeedFast);
 	
fuzzy->addFuzzyRule(fuzzyRule03);
 	
}	

void loop()
 	
{
 	
int input = random(0, 80);
 	
Serial.println("\n\n\nEntrance: ");
 	
Serial.print("\t\t\tDistance: ");
 	
Serial.println(input);
	Início da Função LOOP no qual é responsável por rodar o código incessantemente.
	
	Impressão do sinal de Entrada.

fuzzy->setInput(1, input);
 	
fuzzy->fuzzify();
 	
float output = fuzzy->defuzzify(1);
	Fuzzificação e Defuzzificação do sinal de entrada para o sinal de saída.

Serial.println("Result: ");
 	
Serial.print("\t\t\tSpeed: ");
 	
Serial.println(output);
 	
delay(12000);
 	
}
	Impressão do sinal de Saida com um tempo de espera de 12s para cada loop.

Fonte: Dos autores.

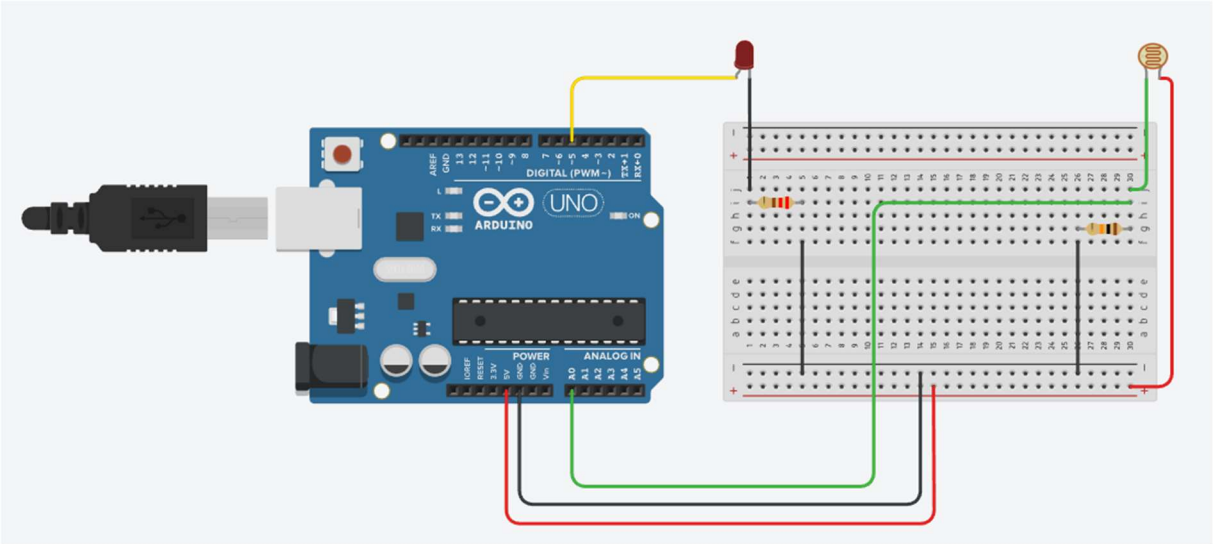
2.3 APLICAÇÃO PRÁTICA DA LÓGICA FUZZY EM UM ARDUINO.

Levando como base o código explicado acima foi realizada uma aplicação prática da lógica fuzzy utilizando uma placa de prototipagem eletrônica de código aberto Arduino. Para assim visualizarmos de forma pratica o processo de fuzzificação e defuzzificação. Utilizando a biblioteca “Fuzzi.h” conseguimos traduzir a simulação realizada no Scilab dentro do Arduino, onde, controlamos a luminosidade em um LED variando sua tensão conforme o valor de entrada de luminosidade no sensor LDR. Para a montagem do sistema, foi utilizado os seguintes componentes:

- 1 Arduino UNO;
- 1 Diodo Emissor de Luz (LED) Difuso de 5mm Vermelho;
- 1 Resistor de 0,25W com resistência de 220 Ω;
- 1 Resistor de 0,25W com resistência de 10K Ω;
- 1 Sensor de Luminosidade LDR 5mm.

Inicialmente o sistema foi montando em uma protoboard (placa de ensaio que serve como um protótipo de um aparelho eletrônico, onde facilita as ligações de componentes) para otimizar os testes iniciais e posteriormente montar o circuito em uma placa universal.

Figura 19 – Montagem do circuito na Protoboard



Fonte: Dos autores.

Após a montagem do circuito na protoboard, foi aplicado uma adaptação do código explicado acima implementando-a no Arduino, trazendo os parâmetros de acordo com os componentes utilizados. Como podemos ver na tabela abaixo.

Tabela 3 – Código Aplicado no Arduino para Controle de Tensão no LED

Aplicação Prática Realizada no Arduino	

#include <Fuzzy.h>
 	
#define PIN_PWM 5
 	
Fuzzy *fuzzy = new Fuzzy();
 	
int Luminosidade_LDR = 0;
 	
void setup()
 	
{
 	
pinMode (A0, INPUT);
 	
pinMode (5, OUTPUT);
 	
Serial.begin(9600);
	Inicia-se incluindo a biblioteca “Fuzzy.h”
	
	Também, definindo a variável “define” como PIN_PWM 5
	
	Definindo a variável Luminosidade_LDR como inteiro com valor 0
	
	Iniciando as configurações como o pino A0 do arduino como Entrada e o pino 5 do arduino como Saida.

FuzzyInput *Luminosidade = new FuzzyInput(1); FuzzySet *BAIXO = new FuzzySet(0, 100, 200, 300); Luminosidade->addFuzzySet(BAIXO); FuzzySet *MEIO_BAIXO = new FuzzySet(250, 400, 400, 550); Luminosidade->addFuzzySet(MEIO_BAIXO); FuzzySet *MEIO_ALTO = new FuzzySet(450, 600, 600, 750); Luminosidade->addFuzzySet(MEIO_ALTO); FuzzySet *ALTO = new FuzzySet(600, 800, 900, 1000); Luminosidade->addFuzzySet(ALTO); fuzzy->addFuzzyInput(Luminosidade);	Configurações das entradas conforme conhecimento do processo. Para essa aplicação usamos quatro entradas: BAIXO = (0, 100, 200, 300); MEIO_BAIXO = (250, 400, 400, 550); MEIO_ALTO = (450, 600, 600, 750); ALTO = (600, 800, 900, 1000).
FuzzyOutput *Tensao = new FuzzyOutput(1); FuzzySet *BAIXA = new FuzzySet(0, 30, 61, 100); Tensao->addFuzzySet(BAIXA); FuzzySet *MEDIA = new FuzzySet(75, 125, 125, 175); Tensao->addFuzzySet(MEDIA); FuzzySet *MEDIA2 = new FuzzySet(75, 125, 125, 175); Tensao->addFuzzySet(MEDIA2); FuzzySet *ALTA = new FuzzySet(150, 183, 217, 255); Tensao->addFuzzySet(ALTA); fuzzy->addFuzzyOutput(Tensao);	Configurações das saídas conforme conhecimento do processo. Para essa aplicação usamos quatro saídas: BAIXA = (0, 30, 61, 100); MEDIA = (75, 125, 125, 175); ~Para relacionar com Meio_Baixo MEDIA2 = (75, 125, 125, 175). ~Relacionar com Meio_Alto ALTA = (150, 183, 217, 255).

FuzzyRuleAntecedent *ifLuminosidadeBAIXO = new FuzzyRuleAntecedent();
 	
ifLuminosidadeBAIXO->joinSingle(BAIXO);
 	
FuzzyRuleConsequent *thenTensaoALTA = new FuzzyRuleConsequent();
 	
thenTensaoALTA->addOutput(ALTA);
 	
FuzzyRule *fuzzyRule01 = new FuzzyRule(1, ifLuminosidadeBAIXO, thenTensaoALTA);
 	
fuzzy->addFuzzyRule(fuzzyRule01);
 	
FuzzyRuleAntecedent *ifLuminosidadeMEIO_BAIXO = new FuzzyRuleAntecedent();
 	
ifLuminosidadeMEIO_BAIXO->joinSingle(MEIO_BAIXO);
 	
FuzzyRuleConsequent *thenTensaoMEDIA = new FuzzyRuleConsequent();
 	
thenTensaoMEDIA->addOutput(MEDIA);
 	
FuzzyRule *fuzzyRule02 = new FuzzyRule(2, ifLuminosidadeMEIO_BAIXO, thenTensaoMEDIA);
 	
fuzzy->addFuzzyRule(fuzzyRule02);
 	
FuzzyRuleAntecedent *ifLuminosidadeMEIO_ALTO = new FuzzyRuleAntecedent();
 	
ifLuminosidadeMEIO_ALTO->joinSingle(MEIO_ALTO);
 	
FuzzyRuleConsequent *thenTensaoMEDIA2 = new FuzzyRuleConsequent();
 	
thenTensaoMEDIA2->addOutput(MEDIA2);
 	
FuzzyRule *fuzzyRule04 = new FuzzyRule(4, ifLuminosidadeMEIO_ALTO, thenTensaoMEDIA2);
 	
fuzzy->addFuzzyRule(fuzzyRule04);
 	
FuzzyRuleAntecedent *ifLuminosidadeALTO = new FuzzyRuleAntecedent();
 	
ifLuminosidadeALTO->joinSingle(ALTO);
 	
FuzzyRuleConsequent *thenTensaoBAIXA = new FuzzyRuleConsequent();
 	
thenTensaoBAIXA->addOutput(BAIXA);
 	
FuzzyRule *fuzzyRule03 = new FuzzyRule(3, ifLuminosidadeALTO, thenTensaoBAIXA);
 	
fuzzy->addFuzzyRule(fuzzyRule03);
 	
}
	Definição das Regras do sistema Fuzzy, de acordo com os parâmetros de entrada com os parâmetros de saída.
	
	Para essa aplicação usamos quatro regras:
	
	• SE a Luminosidade = BAIXO então o nivel de tensão = ALTA;
	
	• SE a Luminosidade = MEIO_BAIXO então o nível de tensão = MEDIA;
	
	• SE a Luminosidade = MEIO_ALTO então o nível de tensão = MEDIA2;
	
	• SE a Luminosidade = ALTO então o nível de tensão = ALTA.

void loop() {	Início da Função LOOP no qual é responsável por rodar o código incessantemente.

Luminosidade_LDR = analogRead(A0); int input = Luminosidade_LDR;	Definição da variável Luminosidade_LDR como o valor coletado pela porta A0. Definindo variável inteira input como o valor da variável Luminosidade_LDR
Serial.println("\nEntrada: "); Serial.print("Luminosidade: "); Serial.println(input);	Impressão do sinal de Entrada.

fuzzy->setInput(1, input);
fuzzy->fuzzify();
int output = fuzzy->defuzzify(1);
	Fuzzificação e Defuzzificação do sinal de entrada para o sinal de saída.

Serial.println("Saída: "); Serial.print("Tensão: "); Serial.println(output);	Impressão do sinal de Saida.

analogWrite(PIN_PWM, output);
delay (50);
 	
}
	Jogando no pino 5 (PIN_PWM) a variável output que é o sinal defuzzificado, onde varia entre 0:255, e assim, proporcionalmente alimentará o LED com 0:5V.

Fonte: Dos autores.

3 RESULTADOS

Ao aplicar a lógica fuzzy no software Scilab para simular o controle de tensão em um LED com base na iluminação do ambiente, foi obtido um resultado eficiente contudo com poucas oscilações de saída regulação do sinal de tensão no LED, variando o sinal próximo de 50 a 200, contudo proporcionando uma adaptação satisfatória à variação das condições de iluminação.

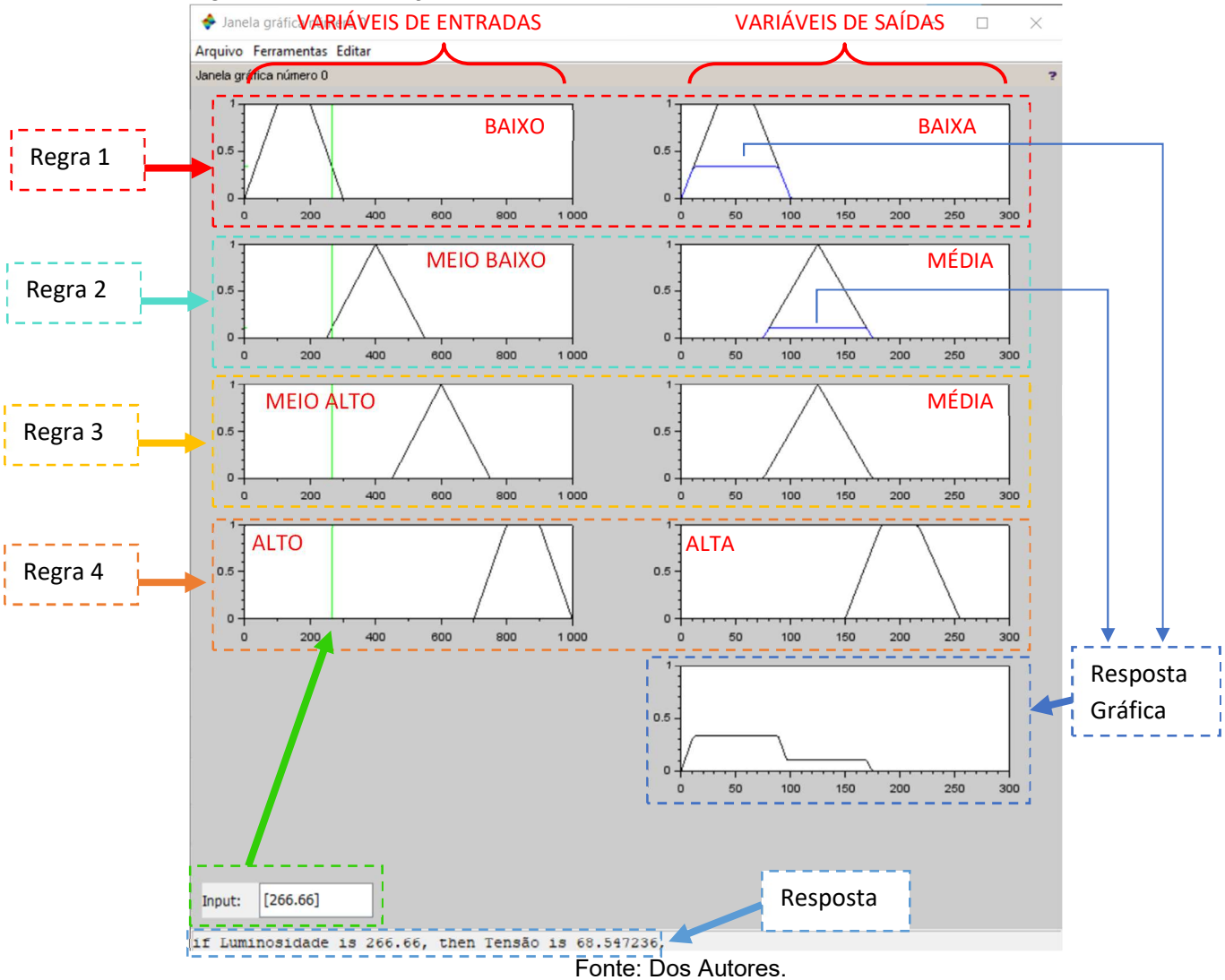
Quando visualizados a prática aplicada no Arduino conseguimos interpretar melhor a relação entre entrada e saída, que representa o processo de fuzzyificação e

defuzzificação. Também podemos notar as poucas oscilações de saída também encontradas no simulador conforme os parâmetros e regras aplicadas.

3.1 RESULTADOS NA SIMULAÇÃO PELO SOFTWARE SCILAB V.6.1.1.

O nível de tensão aplicado no LED é definido conforme a variável ILUMINAÇÃO, e assim, a saída analisa os graus de pertinências da variável e se comporta de acordo com o grau de pertinência mais influente.

Figura 20 – Correlação entre Entrada e Saída de acordo com o Instante Selecionado



Fonte: Dos Autores.

E assim, representando o problema proposto, o nível de tensão aplicado no LED se mostra proporcional a relação das entradas, e faz uso do método de centro de massa para definir o valor final, conforme apresentado na figura, sendo assim, temos um valor atribuído

na saída de 68.54, para uma resposta em que a iluminação possui um valor de 266.66. Nesta situação, existe a aplicação de duas das regras feitas.

Ou seja, para uma entrada onde a iluminação é de 266.66 é correlacionado os subgrupos de entrada “MEIO ALTO” e “MEIO BAIXA” com o de saída “MÉDIA” e assim o resultado com um nível de sinal de tensão de 68,54. Realizou-se diversas simulações no software, a fim de validar as possibilidades do problema. A tabela abaixo contempla os valores das entradas e os respectivos resultado da saída, sendo possível observar a coerência entre os valores.

Tabela 4 – Simulações.

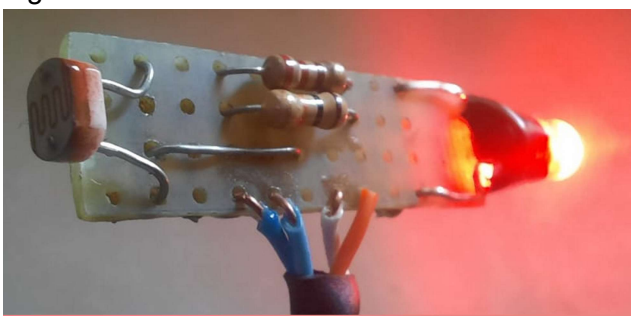
Simulação	Luminosidade	Tensão
1	66.66	49.99
2	133.33	49.99
3	200.00	49.99
4	266.66	68.54
5	333.33	124.99
6	400.00	124.99
7	466.66	124.99
8	533.33	124.99
9	600.00	124.99
10	666.66	124.99
11	733.33	183.77
12	800.00	201.43
13	866.66	201.43
14	933.33	201.73
15	990.00	202.23

Fonte: Dos Autores.

3.2 RESULTADOS NA APLICAÇÃO PRÁTICA UTILIZANDO O ARDUINO.

Uma vez que feito a simulação no software Scilab juntamente com a montagem do prototipo no Arduino aplicando o código descrito acima, seguimos com a montagem do circuito na placa universal para assim, obter uma melhor experiencia nos testes e análises realizadas.

Figura 21 – Circuito Montado na Placa Universal



Fonte: Dos Autores.

Quando comparamos os resultados das simulações realizadas no Scilab com a lógica aplicada no Arduino, conseguimos visualizar nitidamente a similaridade dos valores encontrados após a Defuzzificação, classificando as saídas em grupos:

Tabela 5 – Simulações Divididas em Grupos

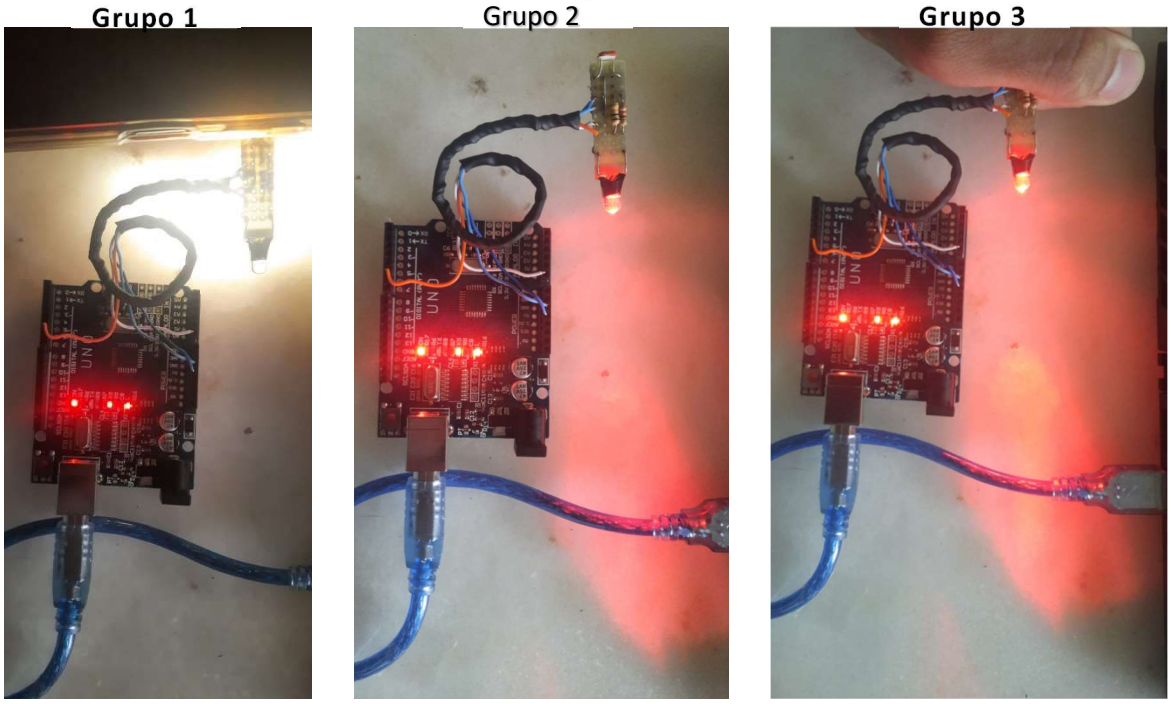
Simulação	Luminosidade	Tensão	
1	66.66	49.99	Grupo 1
2	133.33	49.99	
3	200.00	49.99	
4	266.66	68.54	
5	333.33	124.99	Grupo 2
6	400.00	124.99	
7	466.66	124.99	
8	533.33	124.99	Grupo 3
9	600.00	124.99	
10	666.66	124.99	
11	733.33	183.77	
12	800.00	201.43	

13	866.66	201.43
14	933.33	201.73
15	990.00	202.23

Fonte: Dos Autores.

Assim, reproduzindo as simulações na placa montada, conseguimos visualizar a resposta do LED conforme a luminosidade de acordo com as simulações realizadas, no primeiro teste utilizamos a lanterna de um celular para facilitar a elevação da iluminação ambiente, no segundo teste temos a luminosidade natural e no terceiro teste foi tampado o sensor LDR para reduzir a iluminação ambiente, como podemos ver na figura abaixo:

Figura 22 – Aplicação Prática da Logica Fuzzy



Fonte: Dos Autores.

4 CONCLUSÃO

Em conclusão, expressamos de forma clara e didática a lógica fuzzy e como ela pode ser aplicada em controle de iluminação, tanto por meio de simulações no software Scilab quanto por meio da implementação prática em uma placa eletrônica com um LDR e um LED no Arduino. A importância desse assunto para a indústria em geral é evidente, já que a eficiência energética é uma preocupação crescente em processos industriais. O controle de luminosidade em barracões de depósitos industriais, por exemplo, pode resultar em

economia de energia significativa, automatizando para um cenário onde a iluminação artificial possa ser controlada de acordo com a iluminação natural do ambiente.

É importante destacar que a lógica fuzzy pode ser aplicada em diversas áreas da indústria para otimizar processos e promover a eficiência energética. Por meio do controle de variáveis como temperatura, umidade e fluxo de materiais, é possível reduzir o desperdício de energia e recursos, aumentar a produtividade e melhorar a qualidade dos produtos. Além disso, a lógica fuzzy pode ser usada para sistemas de controle de tráfego, controle de processos químicos, entre outros. Em resumo, a lógica fuzzy é uma ferramenta poderosa para otimização de processos e para a promoção da eficiência energética em diversas aplicações industriais.

Este artigo pode mudar a perspectiva das pessoas sobre a utilização da lógica fuzzy. Ao apresentar exemplos práticos de como essa técnica pode ser aplicada, evidenciando como ela pode ser uma solução viável e eficiente em diversas áreas. Além disso, é importante destacar como a lógica fuzzy pode contribuir para a sustentabilidade, já que a eficiência energética é uma questão cada vez mais importante em todo o mundo. Em suma, este artigo é um convite para que mais pessoas se interessem e se envolvam no desenvolvimento de soluções inovadoras para os desafios do mundo moderno, em que a sustentabilidade é uma questão chave para o futuro.

REFERÊNCIAS

Sampaio, F., Cuenca, M., Rodrigues, R., & Silva, W. C. (2022). Metodologia para simulação da Lógica Fuzzy através do SCILAB SciFLTEditor. SENAI Londrina PR, 1(1), pp. 1-18.

Autodesk. (2023). Tinkercad [Simulação de Arduino, Módulo: eletrônica]. Recuperado de https://www.tinkercad.com/

TEDESCO, Patricia; GERMANO, Vasconcelos. Lógica Fuzzy. Recuperado em 24 de abril de 2023, de https://www.cin.ufpe.br/~if684/EC/aulas/Aula-logica-Fuzzy-SI.pdf.

ALVES, Otávio. eFLL - Fuzzy Logic Library for Arduino. Disponível em: https://github.com/alvesoaj/eFLL. Acesso em: 24 de abril de 2023.

ARDUINO. Software. Disponível em: https://www.arduino.cc/en/software. Acesso em: 27 de abril de 2023.

ARDUINO. Tutorials. Disponível em: https://www.arduino.cc/en/Tutorial/HomePage. Acesso em: 27 de abril de 2023.