Avaliação da Aplicabilidade de Inferência Difusa em Sistema de Controle Embarcado para Irrigação

Valtinei Andrade de Souza*
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N
Barbalho, Salvador, Bahia
valtinei.souza@ifba.edu.br

Frederico Jorge Ribeiro Barboza†
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N
Barbalho, Salvador, Bahia
fredericojorge@ifba.edu.br

RESUMO

Técnicas de Inteligência Artificial são tidas como tipicamente demandantes de muitos recursos computacionais, principalmente de capacidade de processamento e memória. Esta crença deriva do fato de algumas como as de busca heurística e de IA evolutiva, de fato, apresentarem uma solução de compromisso entre estes recursos, requerendo ora mais memória e ora mais processamento, para que uma solução aceitável seja encontrada. Assim o uso de técnicas de IA em sistemas embarcados microcontrolados seria, em tese, não factível. Este trabalho apresenta um controlador construído usando-se uma técnica de IA, a inferência difusa, que foi embarcado em *Arduino*. O trabalho demonstra que o uso de algumas técnicas de IA, em particular o raciocínio difuso, pode na verdade ser embarcado em dispositivos com recursos computacionais limitados. Argumentamos assim, que este artigo apresenta duas contribuições: a primeira, a solução prototipada de controle difuso para irrigação, a segunda uma avaliação da aplicabilidade da inferência *fuzzy* em sistemas embarcados microcontroaldos.

Palavras Chaves

Controlador *Fuzzy*, Microcontroladores, *Arduino*, Controle Irrigação, Inteligencia Artificial e Sistemas Embarcados

1. INTRODUÇÃO

Sistemas embarcados tem por objetivo executar tarefas específicas e promover interações de maneira contínua com o ambiente, através de sensores e atuadores [4]. Como exemplo podem, ser citados o acionamento da ignição eletrônica ou um sistema multimídia de um veículo, um forno micro-ondas, uma máquina para preparo de café, dentre outros.

O ponto em comum nestes sistemas é o emprego da tecnologia para solução de tarefas especificas. Todo projeto que utiliza sistemas embarcados possui em sua plataforma de *hardware* uma unidade de processamento, ou seja, um circuito integrado fixado a uma placa eletrônica, capaz de processar informações do *software* gravado na própria unidade de processamento. Neste conceito de aplicação o *software* é chamado de *firmware* [4].

Dentre as principais características que contribuem para o uso dos sistemas embarcados, destacam-se: o baixo custo final do produto; a flexibilidade em mudanças no projeto para expansão de funcionalidades, resultando no aumento de algumas linhas de código; a adaptabilidade ao se adequar as necessidades dos usuários; e a capacidade computacional, independente de sua operação. Quanto as suas limitações, pode-se salientar o número reduzido de recursos de processamento e memória e a necessidade de ter um baixo consumo de energia e maior autonomia [4]. Comparado aos computadores de modo geral, que consomem *megabytes* de *RAM* (*Random Access Memory*) para um determinado *software*, e utilizam *gigabytes* para armazenamento do sistema operacional e dos programas, um sistema embarcado pode usar menos de 256 *bytes* de *RAM* e seu *firmware* consume alguns milhares de *bytes*.

Uma aplicação comum dos sistemas embarcados é no uso de algoritmos para sistemas dinâmicos acoplados, de modo que cada sistema influencia no comportamento do outro. A esta classe de sistemas chama-se sistemas de controle.

Sistemas de controle embarcados são aplicados, por exemplo, na indústria automobilística para o monitoramento de sensores que verificam a cada momento o estado dos componentes, podendo ativar atuadores ou indicar uma ação a ser realizada pelo condutor. Outas aplicações de controle embarcado incluem o monitoramento de motores, de processos químicos, de controladores de voos, de aplicações aeroespaciais, de sistemas de telecomunicações e internet [4].

Um dos principais problemas em projetos embarcados para automação e controle, é a dificuldade de modelagem para aplicações não lineares, complexas, com múltiplos parâmetros de entrada ou onde o conhecimento associado ao problema é imperfeito, impossibilitando o uso de um modelo matemático preciso. Esta ultima situação envolve o que chama-se, na literatura de sistemas baseados em conheci-

*Aluno do curso de Análise e Desenvolvimento de Sistemas (ADS).

†Mestre em Mecatrônica e Professor do Curso de Análise e Desenvolvimento de Sistemas.

mento, de incerteza. As informações em um sistema baseado em conhecimento podem variar de completamente perfeitas à completamente imperfeitas. Contudo deve-se observar que mesmo quando exposto a situações em que o conhecimento é impreciso, ainda assim o homem faz uso de algum modelo adequado para apoio a sua tomada de decisão. Em Inteligência Artificial(IA) chama-se esse processo de heurística. A heurística pode ser definida como métodos, critérios ou princípios para decidir uma ação a ser tomada com base no conhecimento existente nem sempre preciso [5].

De fato, em determinadas aplicações de controle pode ser muito difícil ou até mesmo impossível expressar o conhecimento heurístico através de um modelo analítico(equação) ou da lógica clássica. Nestas situações, o *expertise* humano está descrito através do uso de termos imprecisos, típicos da linguagem humana.

A lógica difusa é o modelo mais tradicional para o tratamento e raciocínio sobre informações imprecisas e vagas. Este modelo consiste em associar níveis de veracidade a proposições. Na lógica clássica, a veracidade das informações é definida de forma binária obtendo-se resultados como verdadeiro ou falso. Em oposição, na lógica difusa uma proposição pode ser parcialmente verdadeira e ao mesmo tempo parcialmente falsa. Isto é adequado, por exemplo para se representar informações relacionadas a temperatura atmosférica. Neste caso pode-se afirmar que um dia está muito quente, sem que a esta proposição só possa se atribuir os valores verdadeiro ou falso [2] [9].

Técnicas de IA costumam fazer uso massivo de recursos computacionais. Este é o caso por exemplo do uso de processadores nas técnicas de busca heurísticas e do uso de memória na computação evolutiva. A inferência *fuzzy* faz uso de avaliação paralela das regras da base de conhecimento codificada em termos de proposições da lógica difusa. Assim, considera-se que controladores difusos robustos precisem de processamento paralelo e operadores especiais [17]. Algumas abordagens até aqui utilizadas, incluem: expandir o conjunto de instruções dos microcontroladores dedicados a lógica difusa, como por exemplo, o modelo de 16 bits da família *68HC12* produzido pela *Motorola*; utilizar co-processador que coopere com o microcontrolador, modelo produzido por *Togai InfraLogic* e *Toshiba*; uso de um *hardware* de alta velocidade de processamento, introduzindo mecanismos de controle eficaz em arquitetura paralela [19]. Todas as publicações citadas, enfatizaram a necessidade de *hardware* de alto poder de processamento como a principal demanda de um controle difuso.

O problema a ser tratado neste estudo, é avaliar a aplicabilidade tecnológica dos microcontroladores em sistemas embarcados para controle *fuzzy*, propondo uma solução que atenda aos requisitos desejados, considerando as limitações de processamento e memória dos microcontroladores convencionais. O principal aspecto a ser avaliado é a viabilidade de aplicar *Arduino* para projetar um controlador difuso para gerenciamento de sistemas de irrigação. Este controlador fará uso de conhecimento heurístico codificado através de regras difusas, que relacionem a temperatura atmosférica e a umidade do solo, de modo a adequar a vazão de água sobre a área controlada.

Este artigo está organizado em seções onde serão apresentados os principais conceitos necessários para construção de um controlador difuso. A seção 2 apresenta os principais conceitos e características dos microcontroladores. A segui são discutidos tipos, características e arquitetura do *Arduino*. São também detalhadas as principais características da lógica difusa como representação da incerteza, conjunto *fuzzy*, controle *fuzzy*, *Fuzzy Control Language*, fuzificação, inferência e defuzificação. Na seção 3 são apresentados os trabalhos relacionados. A seção 4 descreve o desenvolvimento do controlador descrevendo a metodologia aplicada em cada estágio de controle *fuzzy*. A seção 5 discorrendo sobre a estrutura de *software* e *hardware* aplicada, especificação, operadores, implementação e mapeamento para o *sketch*. A seção 6 explana a avaliação dos resultados obtidos através de testes funcionais e não-funcionais, comparando estes resultados com os obtidos em *softwares* com *jFuzzyLogic* e o *Infuzzy*. Nas seções 7 e 8 são apresentadas as considerações finais e os trabalhos futuros.

2. REFERENCIAL TEÓRICO

2.1 Microcontroladores

A automação de processos vem sofrendo diversas transformações ao longo do tempo impulsionados pelo desenvolvimento da microeletrônica. Dentre os dispositivos que contribuíram para esta evolução, destacam-se o papel fundamental dos microcontroladores. Hoje amplamente empregados no controle de equipamentos domésticos e industriais, são componentes que encapsulam na sua arquitetura, além da *CPU* (*Central Processing Unit*), elementos como memória *ROM* (*Read Only Memory*) e *RAM*, temporizadores, contadores, *PWM* (*Pulse-Width Modulation*), canais de comunicação e conversores analógicos/digitais [11].

Quando relacionado a *CPU*, os microcontroladores tem um poder de processamento inferior aos microprocessadores. O conjunto de instruções dos mesmos limita-se a informações simples, possui uma frequência de *clock* mais baixa e espaço de endereçamento de memória limitado. Apesar destas limitações, quando comparado com os microprocessadores, surge um novo campo de aplicabilidade dos mesmos, para utilização em sistemas controlados, que exigem menor complexidade e menor custo, diferentes dos microprocessadores utilizados em sistemas que exigem processamento mais complexo. Os microcontroladores são amplamente difundidos para uso em aplicações de controle industrial, semáforos, inversores eletrônicos, controladores lógicos programáveis e eletrodomésticos em geral [11].

A programação dos microcontroladores é executada de forma mais simples quando comparada com a dos microprocessadores. São acessados de forma padronizada, facilitando o desenvolvimento de soluções automatizadas para resolução de problemas específicos. Apesar de ser de fácil programação e não exigir maiores complexidades, é fundamental o conhecimento amplo dos componentes de *hardware* que serão conectados ao circuito eletrônico, pois os mesmos poderão fornecer entradas e saídas para sistemas automatizados distintos [11] [4].

2.1.1 Arduino

O *Arduino* pode ser descrito como uma placa eletrônica gerenciada por um microcontrolador que disponibiliza entra-

das e saídas digitais e analógicas para utilização de sensores e atuadores, dentre outros componentes eletrônicos. O *hardware* possui um microcontrolador, um cristal oscilador, alimentação de 3 e 5 volts, possuindo ainda pinos de entradas e saídas para conexão de circuitos externos [14].

A placa eletrônica *Arduino* consiste de um microcontrolador *Atmel AVR* de 8 *bits*, composta de componentes eletrônicos os quais facilitam a comunicação com circuitos externos e facilitam a programação do *software* embarcado. Um outro aspecto importante é a capacidade de expansão de suas funcionalidades, através de módulos expansivos chamados de *Shields*. O *Arduino* utiliza uma série de chips *megaAVR*, dentre estes destacam-se o *ATmega8*, o *ATmega168* e o *ATmega1280*. Contudo, existem alguns modelos similares que utilizam outros processadores [8] [14].

A plataforma computacional *Arduino* pode interagir através da atuação de sensores programados como entrada de um sistema de monitoramento ou por meio de *software* para acionamento de um controle especifico [8]. Um sistema embarcado com *Arduino* pode acionar mecanismos em uma planta industrial, assim como coletar dados para um sistema online de monitoramento de sensores, gerando referências para compor informações estatísticas do objeto de estudo.

Na Tabela 1, é apresentado um comparativo entre 7 modelos de *Arduino*. Nele são apresentados características relacionadas ao processamento e memória. É possível verificar que a capacidade de memória *Flash* varia entre 32 *KB* e 512 *KB*, *SRAM(Static Random Access Memory)* varia de 2 *KB* a 96 *KB*, *EEPROM* de 1 *KB* a 4 *KB* e *Clock* varia de 16 *MHz* a 84 *MHz* [14] [15].

2.2 Fuzzy

2.2.1 Representação da Incerteza

Quando uma pessoa é tratada como velha ou jovem, um objeto como largo ou curto, um questão a ser considerada na caracterização desta pessoa ou objeto é a imprecisão da informação. As fronteiras dos conjuntos de classificação não estão muito bem definidas. Contudo os seres humanos costumam lidar com estas informações diariamente apesar desta imprecisão. Esta forma de raciocínio humano é fundamentada em aproximações e rodeados de suposições e incertezas.

A teoria dos conjuntos nebulosos é o modelo mais aplicados no tratamento da informação imprecisa. Este modelo, introduzido por Zadhe tem objetivo permitir níveis de pertinência de uma elemento de uma dado conjunto, ou seja possibilita um elemento de pertencer com maior ou menor intensidade ao dado conjunto. Basicamente isto se faz quando o grau de pertinência de um elemento ao conjunto, que na teoria dos conjuntos clássicos assume valores 0 ou 1, passa a ser dado por um valor no intervalo dos números [0,1] [3].

2.2.2 Teoria dos Conjuntos Nebulosos

A teoria dos conjuntos *crisp* ou tradicionais, define claramente o conceito de pertinência de um elemento a um determinado conjunto. Se é definido um conjunto A em um universo U, os elementos de U pertencem ou não pertencem ao conjunto A, podendo ser definido como [1]:

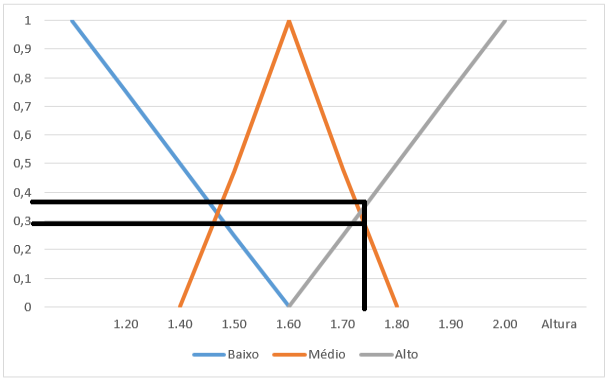

Figure 1: Função de pertinência lógica fuzzy.

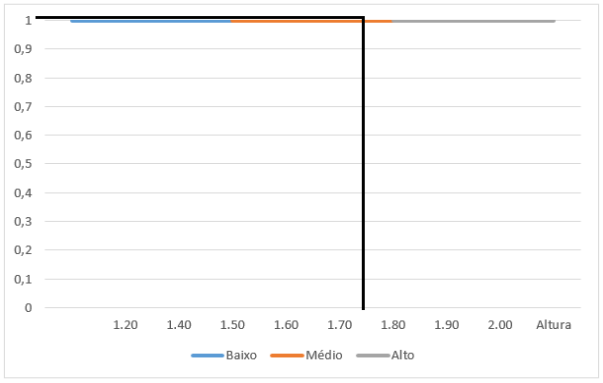

Figure 2: Função de pertinência lógica convencional.

$$\text{f (x)} = \begin{cases} 1 & \text{se somente se x} \in A; \\ 0 & \text{se somente se x} \notin A. \end{cases}$$

A teoria dos conjuntos nebulosos possui uma caracterização mais ampla que contempla infinitos valores no intervalo de [0,1]. Se definido um conjunto *fuzzy* A em um universo U tem-se uma função de pertinência $\mu A(x)$: $U \rightarrow [0,1]$. A mesma define quanto x é compatível com A. Se $\mu A(x) = 1$ então x é totalmente compatível A; caso $\mu A(x) = 0$ então x é totalmente incompatível com A, se $0 < \mu A(x) < 1$ então x é parcialmente compatível A, com compatibilidade $\mu A(x)$ [1] [2].

Assim, por exemplo, quando é realizada uma análise de altura utilizando conjuntos nebulosos é possível ter graus de pertinência. Considerando como exemplo uma pessoa de 1,75, pode-se dizer que ela pertença ao conjunto $\mu medio(x)$ com compatibilidade 0,29 e $\mu alta(x)$ com compatibilidade 0,38 (Figura 1). Entretanto quando essa análise utiliza-se da teoria dos conjuntos clássicos teria apenas uma única definição médio ou alto, ou seja, total compatibilidade ou incompatibilidade com os conjuntos (Figura 2).

2.2.3 Operações em Conjuntos Fuzzy

Na teoria clássica dos conjuntos existem propriedades e diversas operações. Da mesma maneira ocorre com os conjuntos nebulosos. Operações como união, intersecção, complemento são fundamentais para os sistemas que utilizam lógica

Arduino	Microcontroller	Flash Memory	SRAM	EEPROM	Clock Speed
Uno	ATmega328	32KB	2KB	1KB	16 MHz
Mega 2560	ATmega2560	256KB	8 KB	4 KB	16 MHz
Leonardo	ATmega32u4	32 KB	2.5 KB	1 KB	16 MHz
Due	AT91SAM3X8E	512 KB	96KB	-	84MHz
Yún	ATmega32u4	32 KB	2.5 KB	1KB	16MHz
Ter	ATmega32u4	32 KB	2.5 KB	1KB	16MHz
Micro	ATmega32u4	32 KB	2.5 KB	1KB	16MHz

Tabela 1: Modelos de Arduinos.

$t-norma$	$t-conorma$	nome
$min(a,b)$	$max(a,b)$	Zadeh
$a\,.\,b$	$a+b-ab$	probabilista
$max(a+b-1,0)$	$min(a+b,1)$	Lukasiewicz
$\begin{cases} a, & \text{se } b=1 \\ b, & \text{se } a=1 \\ 0, & \text{senão} \end{cases}$	$\begin{cases} a, & \text{se } b=0 \\ b, & \text{se } a=0 \\ 1, & \text{senão} \end{cases}$	Weber

Figure 3: Modelos t-norma e t-conorma.

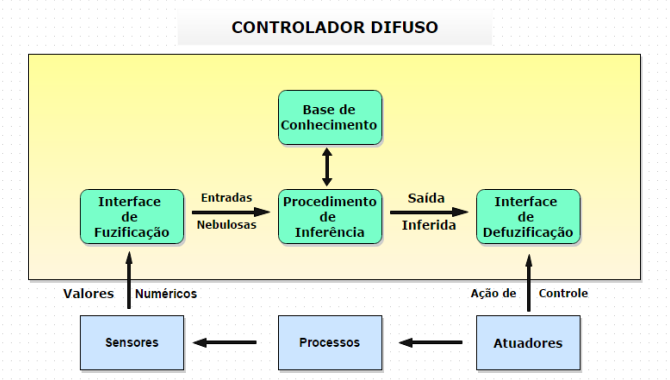

Figure 4: Controlador Fuzzy.

nebulosa. As operações básicas destes conjuntos foram definidas por Zadeh [1].

A intersecção entre conjuntos pode ser representada com D $= A \cap B$, ou seja, representa uma operação E, que verifica o nível de compatibilidade de um elemento aos dois conjuntos *fuzzy*. A operação de união pode ser descrita como E $= A \cup B$,ou seja, representa uma operação OU, que verifica o nível de compatibilidade de um elemento a qualquer um dos conjuntos *fuzzy*. Nos conjuntos nebulosos a intersecção é representada por uma família de operadores denominados t – *normas*, e a união é representados por uma família de operadores chamados de t – *conormas* ou s – *normas* [1] [2]. Uma t-norma é uma função $T : [0,1]^2 \rightarrow [0,1]$ que é comutativa, associativa e monotônica e 1 é um elemento neutro $T(a,1) = a$. A função t – *conorma* possui as mesmas propriedades, mas com 0 sendo o elemento neutro $\perp(a,0) = a$.

Na Figura 3 são representadas os principais modelos de funções que definem t- norma e t-conorma.

2.2.4 *Raciocínio e Controle Fuzzy*

Um controlador *fuzzy* basicamente é utilizado para modelar ações de controle a partir de uma base de dados construída substancialmente do conhecimento de especialistas, diferente dos controladores convencionais nos quais o controle é realizado através de modelagem matemática. O grande propósito dos controladores nebulosos, é aproveitar o conhecimento de especialistas de maneira a simplificar os projetos nos quais o emprego das tecnologias convencionais, não atendem por serem custosas ou complexas para desenvolvimento [2] [12].

As estratégias de controle nebuloso teve como percussor *E. H. Mamdani*. Posteriormente surgiram diversas pesquisas neste campo de estudo. [2] [16].

O controlador difuso modela ações a partir do conhecimento de especialistas, para gerenciamento de processos sofisticados, permite a construção do conhecimento com o objetivo de tratar aspectos vagos da informação como a imprecisão e a incerteza. A estrutura de um controlador *fuzzy* descrito na Figura 4 é formada basicamente de componentes como: interface de fuzificação, base de conhecimento, procedimento de inferência e a interface de defuzificação. Diversos modelos foram propostos na literatura porém o modelo mais comum de representação demonstra a disposição dos módulos e o fluxo das informações [2] [10].

Na fase de projeto do controlador difuso é fundamental a definição de alguns parâmetros fixos fundamentais com base no conhecimento dos especialistas ou através de experimentos. São parâmetros imutáveis em condições normais como: o número de variáveis de entrada, o número de variáveis de saída, recursos de operações sobre os dados de entrada, variáveis linguísticas, funções de pertinência, intervalos de discretização e normalização, estrutura da base de regras e conjunto básico de regras [16].

FCL

O *FCL*(*Fuzzy Control Language*) é um padrão de programação utilizado no controle difuso. As especificações da sintaxe *FCL* são definidas pela *International Electrotechnical Commission (IEC)* através do documento *IEC 61131-7*. Este padrão de modelagem de controle difuso é utilizado em *softwares* como *jFuzzyLogic* para definição das variáveis linguísticas, termos linguísticos, fuzificação, inferência e defuzificação [21]. A partir do arquivo *.fcl* são realizadas ações

de controle com base no domínio estabelecido. A especificação do *FCL* utilizado no protótipo está descrito na seção 10.1.

Variáveis Linguísticas

Uma variável linguística pode ser definida por uma 4-upla (X; Ω; T(X);M), onde X é o nome da variável, Ω é o universo de discurso de X, T(X) é um conjunto de nomes para valores de X, e M é uma função que associa uma função de pertinência a cada elemento de T(X). São chamados de termos linguísticos, indistintamente, tanto os elementos de T(X) quanto suas funções de pertinência [2].

As variáveis linguísticas apresentam valores de maneira informal, ou seja, os valores são descritos por palavras ou sentenças ao invés de números. Cada valor linguístico de uma variável é associado a um conjunto nebuloso no universo do discurso. Por exemplo a variável linguística *temperatura* pode assumir valores *baixa*, *média*, *alta* e *muito_alta*. Estes valores são descritos conforme os conjuntos nebulosos associados a funções de pertinência. As variáveis linguísticas normalmente são definidas através de termos primários (alto, baixo, pequeno, médio, grande), e modificadores (muito, pouco, levemente, extremamente) [16].

Os valores empregados nas variáveis linguísticas possuem uma descrição linguística muito utiliza por seres humanos com base em suas experiências. O emprego de variáveis linguísticas permite o tratamento de informações complexas não analisadas facilmente através dos modelos matemáticos convencionais. A principal função da variável linguística é caracterizar de maneira imprecisa fenômenos complexos.

Funções de Pertinência

As funções de pertinência ou *membership function* representam o conhecimento do especialista e as propriedades semânticas são utilizadas para caracterizar um conjunto nebuloso. A escolha da função impacta diretamente no desempenho do controlador *fuzzy*. A função de pertinência de cada conjunto difuso é um aspecto de grande importância para operação, ou seja, a escolha da função e de sua forma devem seguir critérios de acordo com o universo que estejam trabalhado [2].

As funções de pertinências podem assumir diferentes formas de representações, visando desenhar um contexto de utilização. Estas funções podem ser definidas a partir da experiência e da perspectiva do especialista, porém corriqueiramente é comum a utilização da forma triangular, trapezoidal e gaussiana.

Interface de Fuzificação

Durante a etapa de fuzificação é realizada a identificação das variáveis de entrada e verificado o grau de pertinência do valor numérico com os conjuntos existentes. Quando estes valores são transformados em conjuntos nebulosos, os mesmos tornam-se instâncias das variáveis linguísticas. É a etapa responsável pela tradução dos valores entrada provenientes de sensores ou dispositivos computacionais em valores nebulosos. Os valores numéricos devem estar contidos no universo do discurso e pertencer ao intervalo estabelecido

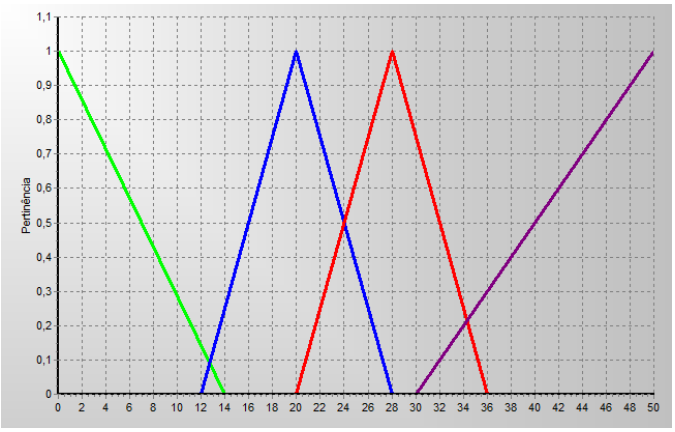

Figure 5: Gráfico de temperatura.

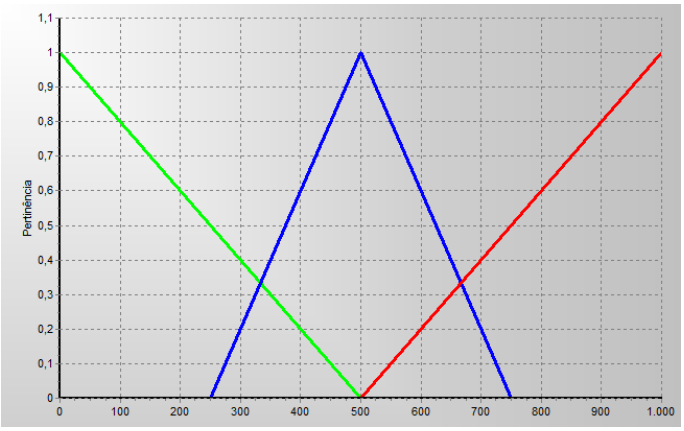

Figure 6: Gráfico de umidade do solo.

por especialistas [10].

Na etapa de fuzificação é realizada a atribuição ou cálculo para representar o grau de participação em um ou mais, conjuntos *fuzzy* [16]. Conforme Figura 5 é verificado função de pertinência de *temperatura* e *umidade* visto na Figura 6 onde cada valor de entrada tem um grau de participação em cada um desses conjuntos. O grau de adesão é determinada por uma função de pertinência, que é definido com base na experiência ou intuição. Quando o sistema estiver em operação estas funções de pertinência não mudam [2].

Exemplificando, considere a leitura do sensor de temperatura com valor em 24 °C tem-se as seguintes pertinências dos termos linguísticos:

- Valor entrada *temperatura* = 24;
- Pertinência *baixa* = 0,00;
- Pertinência *media* = 0,50;
- Pertinência *alta* = 0,50;
- Pertinência *muito_alta* = 0,00.

Neste estágio o valor numérico 24 é transformado em grau de pertinência de entrada de um ou mais conjuntos. Na fase seguinte do controle difuso, todos estes termos linguísticos passam pelo processo de inferência conforme o grau de pertinência de cada termo.

Base de Conhecimento

A base de conhecimento é composta por dados e regras para caracterização das estratégias de controle. As definições sobre a discretização e normalização do universo de discurso e funções de pertinências ficam armazenados na base de dados. A base de regra é formada por estruturas condicionais SE <premissa> ENTÃO <conclusão>. O processamento destas regras juntamente com os dados de entrada são realizados pelo procedimento de inferência, nesta etapa são realizadas ações de controle com o estado do sistema, executando operações de implicação[5]. O emprego de regras em controladores *fuzzy* devem contemplar o máximo de combinações possíveis. As proposições são relacionadas através de conectivos lógicos E/OU. Podendo ter múltiplas entradas e múltiplas saídas ou poderá ter múltiplas entradas e única saída [2] [16].

Procedimento de Inferência

A etapa do procedimento de inferência obtém-se da base de conhecimento a contribuição do especialista no processo a ser controlado. A partir destes dados utiliza-se as regras de inferência. Neste estágio será realizada a lógica de tomada de decisão, para análise de todos os prováveis conjuntos de pertinência de cada variável a ser controlada.

SE peso é médio E altura é mediana ENTÃO condição é normal;

Conforme exemplo peso e altura são variáveis controladas podendo assumir valores pertencentes aos respectivos conjuntos nebulosos médio e mediana, de acordo com conjunto pertencente de cada variável. É importante observar que toda análise é feita em relação as variáveis controladas quantificadas com valores *fuzzy*. Em função do grau de compatibilidade de cada regra, é necessário realizar o processo de agregação de acordo com os valores obtidos em cada regra de inferência, obtendo-se uma ação de controle global [2].

Nesta etapa é possível optar por diversos modelos encontrados na literatura como os modelos clássicos, compreendendo os trabalhos de *Mamdani* e *Larsen*, e os modelos de interpolação, empregado nas publicações de *Takagi-Sugeno* e *Tsukamoto*.

O controle nebuloso por interpolação dispensa a definição de funções de implicação e operadores para a inferência. Neste modelo são utilizadas funções monotônicas diferentes para cada regra. Os principais modelos de interpolação foram projetados por *Takagi-Sugeno* e *Tsukamoto*. O modelo de *Sugeno* consiste em definir cada regra como uma função das variáveis linguísticas de entrada. O resultado de cada regra é um valor numérico não *fuzzy*. Este modelo não utiliza função de implicação especifica, as saídas são resultantes do cálculo das médias ponderadas obtidas com respostas das regras. Este modelo apresenta um único valor de inferência

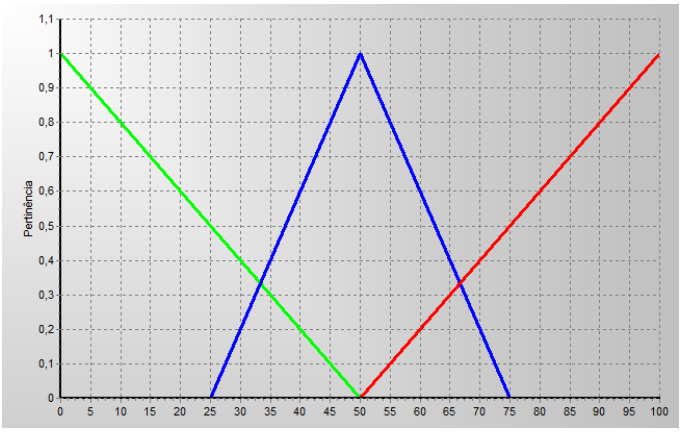

Figure 7: Gráfico de irrigação.

$\mu_i \neq 0$ e os demais com pertinência zero $\mu_i = 0$, não sendo necessária a etapa de defuzificação [7].

Os modelos de controles nebulosos clássicos são baseados em funções de implicação e operações de composição para definição de um único conjunto de saída. O resultado de cada regra define um valor de pertinência de cada termo linguístico comumente representados por conjuntos nebulosos convexos como funções em forma triangular e trapezoidal [2].

Com base no modelo de *Mamdani* na etapa de inferência, são tomadas as decisões à partir da base de regras, após a fase fuzificação onde são determinados os valores de pertinência de cada conjunto, neste estágio são aplicadas as regras SE <premissa> E <premissa> ENTÃO <condição> usando a operação de inferência proposta por *Zadhe* [1] minimo(t-norna), para todas combinações possíveis. Após verificar o valor de pertinência dos conjuntos a exemplo da variável *irrigacao* conforme Figura 7 é aplicado a operação de agregação através máximo(t-conorma). Por exemplo se os valores de pertinência de temperatura e umidade forem respectivamente 1.00 e 0,22:

SE ($temperatura = alta$) E ($umidade = encharcado$) ENTÃO ($irrigacao = fechado$);

SE ($temperatura$= 1,00) E ($umidade$ = 0,22) ENTÃO ($fechado$ = 0,22);

SE ($temperatura$ = $alta$) E ($umidade$ = $umido$) ENTÃO ($irrigacao$ = $gotejando$);

SE ($temperatura$ = 1,00) E ($umidade$ = 0,56) ENTÃO $irrigacao$ = 0,56);

t-norma = min($temperatura$ = 1,00, $umidade$ = 0,22) = 0,22;

t-norma = min($temperatura$ = 1,00,$umidade$ = 0,56) = 0,56;

t-conorma = max($fechado$)= max(0,22, 0,00) = 0,22;

t-conorma = max($gotejando$)= max(0,56, 0,00) = 0,56;

Neste caso, após a operação de agregação os resultados de pertinências obtidos foram 0,22 e 0,56 para os respectivos conjuntos *aberto* e *gotejando*. Na última fase do processo é realizada a conversão destes conjuntos difusos em saída discreta.

Interface de Defuzificação

Após a etapa de inferência é realizada a fase de defuzificação que consiste na operação de converter os valores difusos de saída em valores numéricos. Assim a defuzificação é a transformação inversa que traduz a saída do domínio *fuzzy* para o domínio discreto. Existem diversos métodos para defuzificação escolhidos de acordo com o sistema utilizado. Os principais são primeiro dos máximo, método da média dos máximos, método de centro de área [2].

- Primeiro dos Máximos(SOM): O valor de saída é identificado conforme o ponto que o grau de pertinência da variável linguística atinja o primeiro valor máximo;
- Método da Média dos Máximos(MOM): A saída é obtida calculando a média entre os dois elementos extremos do universo do discurso que correspondem aos maiores valores da função de pertinência do conjunto *fuzzy* de saída;
- Método de Centro de Área ou Centro de Gravidade(COA): O valor de saída é o valor no universo que corresponde ao centro da área da função de distribuição de possibilidade da ação de controle;

No estágio de defuzificação são empregadas técnicas para resolver problemas como imprecisão e conflitos. Esta etapa é fundamental por duas razões: a primeira é decifrar o significado das ações vagas, tais como *gotejando*. A segunda resolve conflitos entre as ações concorrentes, como *fechado* e *gotejando*. O objetivo desta etapa é traduzir estes termos linguísticos em valores numéricos.

O cálculo centro de gravidade consiste na razão entre o somatório de "u" vezes o seu grau de pertinência pelo somatório do grau de pertinência. No cálculo do centro de gravidade é fundamental identificar o baricentro ou centroide da figura plana [1]:

$$X_{CG=}\frac{\sum_{i=1}^{i=N} u_i.\mu_u(u_i)}{\sum_{i=1}^{i=N} \mu_u(u_i)}; \quad (1)$$

Onde:

- u_i = Valor do eixo das abscissas;
- μ_u = Valor da pertinência de u_i;
- **N** = Limite do somatório;

Na Figura 8 é possível observar um exemplo do resultado da etapa de defuzificação utilizando o método do centro da gravidade. Inicialmente são definidos os passos onde largura está diretamente relacionada ao grau de precisão do sistema, quanto menor a amplitude mais preciso será o resultado [6]. O conjunto *fechado* possui um grau de pertinência 0,22 e o conjunto *gotejando* possui um grau de pertinência igual a 0,53 e o passo é igual a 2,5.

$$X_{CG} = \frac{0*0,22+2.5+...+30*0,22+32,5*0,3+...75*0}{0,22+0,22+...+0,22+0,3+...0}; \quad (2)$$

Portanto o valor de saída por exemplo para acionamento de uma válvula de irrigação seria o X_{CG} que representa o centro de gravidade conforme Figura 8 indicando a regulagem da válvula de saída.

3. TRABALHOS CORRELACIONADOS

3.0.5 Correlatos Controle Fuzzy

A solução *fuzzy* proposta por *Changuel* [23] utiliza a *AMSC* projeto de *hardware* especifico para controle clássico, onde o algoritmo foi modificado para um controle *fuzzy*. Já *Reyneri* [24] propõe sistema embarcado *fuzzy* que utiliza 2 microcontroladores, *FPGA (Field Programmable Gate Array)*, memória externa, o projeto demanda um *hardware* robusto com um neuro-controlador *fuzzy* baseado em um chip de fluxo de pulso neural. O sistema mescla controladores neurais analógicos e *fuzzy* para controlar diferentes sistemas mecânicos. O projeto proposto por *Fodor* [25] apresenta um controlador difuso embarcado utilizando um processador Th4S320C25 que possui *hardware* especifico para gravação de software, com licença proprietária de alto custo, além de necessitar de memória externa *SRAM*, *EPROM*.

O projeto referido por *Raji* [26] trata-se de um controle *fuzzy* embarcado desenvolvido com *Chip Neuron 3180* solução de *hardware* projetada com 3 *CPUs* de 8*bits RAM* e *EEPROM*. A *IDE* e *hardware* para programação proprietário apresenta alto custo de projeto. A pequisa publicada por *Kumar* [27] propõe um sistema para controle difuso aplicado no controle da velocidade de motores utilizando, modelo teórico simulado no *software* MATLAB.

O projeto apresentado nestes artigo se diferencia dos trabalhos correlacionados de controle *fuzzy* embarcados citados nesta subseção por ser uma protótipo que utiliza *hardware* livre de baixo custo, e apresenta um bom desempenho apesar limitações de processamento e memória. Os projetos apresentados possuem *hardwares* robustos com maior custo de projeto [23] [24], possui *hardware* especifico para gravação de *software* no processador [23] [24] [26], ou modelo teórico simulado [27].

3.0.6 Correlatos Controle Fuzzy Aplicado em Sistemas de Irrigação

Zhang [9] desenvolveu uma solução de controle *fuzzy* buscando mais eficiência na gestão da irrigação, pois o mesmo considera os métodos tradicionais e automatizados por agendamento ineficientes por não considerar aspectos como as respostas não lineares dos sensores utilizados. O sistema

Autor	Controle Fuzzy	Teórico	Outras Técnicas de IA	Embarcado	Controle de Irrigação	Hardware/IDE Livre
Zhang	Sim	Não	Não	Não	Sim	Não
Bahat	Sim	Sim	Não	Não	Sim	Não
Changuel	Sim	Não	Não	Sim	Não	Não
Reyneri	Sim	Não	Sim	Sim	Não	Não
Fodor	Sim	Não	Não	Sim	Não	Não
Raji	Sim	Não	Sim	Sim	Não	Não
Kumar	Sim	Sim	Não	Não	Não	Não
Protótipo	Sim	Não	Não	Sim	Sim	Sim

Tabela 2: Comparativo de trabalhos correlacionados.

de gerenciamento relatado por ele não utiliza nenhum operador, apenas um computador com arquitetura mínima de microprocessador 386, em um software com cerca de 1000 linhas de código desenvolvido na linguagem C.

Bahat [13] define o controlador de irrigação como o cérebro do sistema de irrigação, pois supervisiona água despejada nas plantas permitindo resultados otimizados consumindo o mínimo de água. Em seu modelo denominado como circuito fechado de controle com parâmetros de entrada como: umidade do solo, temperatura, radiação, velocidade do vento, umidade do ar e salinidade. A proposta não foi implementada, apenas realizado simulações no *MATLAB*.

A proposta apresentada neste trabalho se difere das propostas de *Zhang* e *Bahat* por ser um controlador embarcado, hardware livre, de baixo custo. Enquanto as soluções propostas usam um computador para o controle *fuzzy* [9], e proposta teórica [13] simulada no *MATLAB*.

4. DESENVOLVIMENTO DO CONTROLADOR

Este trabalho apresenta uma solução de controle *fuzzy* embarcado em *Arduino* aplicada a processos de irrigação. O controle projetado tem como variáveis linguísticas a *temperatura*, a *umidade* e a *irrigacao*, sendo que as duas primeiras variáveis representam valores de entrada a última o valor de atuação. O controlador implementado segue o método de *Mamdani* [7]. A especificação em FCL do controlador encontra-se no apêndice 10.1. Esta seção apresentará os detalhes de controlador implementado.

4.1 Variáveis Linguísticas

Na implementação do controle *fuzzy* objeto desse trabalho foram utilizadas três variáveis linguísticas.

4.1.1 Temperatura

Esta variável linguística representa a *temperatura* atmosférica do local a ser irrigado. Por senso comum, deve-se observar que quanto mais quente, maior deve ser a vazão do sistema de irrigação. Isto deve-se a noção que quanto maior a temperatura do ar atmosférico maior a evaporação do solo irrigado. O domínio da variável *temperatura* vai de [0; 50] graus. Os termos linguísticos utilizados foram *baixa*, *media*, *alta* e *muito_alta*. A Figura 5 exibe os gráficos das funções de pertinência dos termos linguísticos da varável *temperatura*.

A seguir é apresentada a especificação da 4-upla que define a variável linguística *temperatura*:

X: *temperatura*
Ω: [0; 50]
T(X): *baixa*, *media*, *alta*, *muito_alta*
M(T(X)):$T(X) \rightarrow (\mu(y) \rightarrow [0;1]) =$ (*baixa*,$\mu_{baixa}(t)$), (*media*, $\mu_{media}(t)$), (*alta*, $\mu_{alta}(t)$), (*muito_alta*, $\mu_{muito\_alta}(t)$)

onde,

$$\mu_{baixa}(t) = \begin{cases} -0.07142857142857t + 1, & \text{se } t \geq 0 \text{ e } t < 16 \\ 0 & \text{se } t \geq 14 \text{ e } t \leq 50 \end{cases}$$

$$\mu_{media}(t) = \begin{cases} 0.125t - 1.5, & \text{se } t \geq 12 \text{ e } t < 20 \\ -0.125t + 3.5, & \text{se } t \geq 20 \text{ e } t < 28 \\ 0 & \text{se } t \geq 28 \text{ e } t \leq 50 \\ 0 & \text{se } t \geq 0 \text{ e } \leq 12 \end{cases}$$

$$\mu_{alta}(t) = \begin{cases} 0.125t - 2.5, & \text{se } t \geq 20 \text{ e } t < 28 \\ -0.125t + 4.5, & \text{se } t \geq 28 \text{ e } t < 36 \\ 0 & \text{se } t \geq 36 \text{ e } t \leq 50 \\ 0 & \text{se } t \geq 0 \text{ e } t \leq 20 \end{cases}$$

$$\mu_{muito\_alta}(t) = \begin{cases} 0.05t - 1.5, & \text{se } t \geq 30 \text{ e } t \leq 50 \\ 0 & \text{se } t \geq 0 \text{ e } t \leq 30 \end{cases}$$

Umidade

A variável linguística *umidade* do solo ou teor em água é definida como a massa da água contida em uma amostra de solo dividido pela massa de solo seco, sendo expressa em quilogramas de água por quilogramas de solo. Por senso comum, deve-se observar que quanto menor a concentração de água, maior deve ser a vazão do sistema de irrigação. Isto deve-se a noção do solo ser considerado um grande reservatório, cuja a quantidade de água armazenada varia com a umidade. O domínio da variável *umidade* vai de [0; 1000]. Os termos linguísticos utilizados foram *seco*, *umido* e *encharcado*. A Figura 6 exibe os gráficos das funções de pertinência dos termos linguísticos da varável *umidade*.

A seguir é apresentada a especificação da 4-upla que define a variável linguística *umidade*:

Nome da variável: *umidade*
Universo discurso: [0; 1000]
Temos linguísticos: *seco*, *umido*, *encharcado*
M(T(X)):$T(X) \rightarrow (\mu(y) \rightarrow [0;1]) =$ (*seco*, $\mu_{seco}(t)$),

(*umido*, $\mu_{umido}(t)$), (*encharcado*, $\mu_{encharcado}(t)$)

onde,

$$\mu_{seco}(t) = \begin{cases} -0.002t + 1, & \text{se } t \geq 0 \text{ e } t < 500 \\ 0 & \text{se } t \geq 500 \text{ e } t \leq 1000 \end{cases}$$

$$\mu_{umido}(t) = \begin{cases} 0.004t - 1, & \text{se } t \geq 250 \text{ e } t < 500 \\ -0.004t + 3, & \text{se } t \geq 500 \text{ e } t < 750 \\ 0 & \text{se } t \geq 750 \text{ e } t \leq 1000 \\ 0 & \text{se } t \geq 0 \text{ e } t \leq 250 \end{cases}$$

$$\mu_{encharcado}(t) = \begin{cases} 0.002t - 1, & \text{se } t \geq 500 \text{ e } t \leq 1000 \\ 0 & \text{se } t \geq 0 \text{ e } t \leq 500 \end{cases}$$

Irrigação

A variável linguística *irrigacao* é definida como a vazão da água a ser despejada no solo. Por senso comum, deve-se observar que quanto mais quente a temperatura e menor for a umidade do solo, maior deve ser a vazão do sistema de irrigação. A vazão de água para irrigação se dá de forma percentual, influenciada por parâmetros físicos da natureza como temperatura e umidade do solo, variáveis que interferem diretamente o processo de gotejamento de água no solo. Quando essas condições mudam a quantidade de água utilizada na irrigação também mudam. O domínio da variável *irrigacao* vai de [0; 100]. Os termos linguísticos utilizados foram *fechado*, *gotejando*, *aberto*. A Figura 7 exibe os gráficos das funções de pertinência dos termos linguísticos da varável *irrigacao*.

A seguir é apresentada a especificação da 4-upla que define a variável linguística *irrigacao*:

Nome da variável: *irrigacao*
Universo discurso: [0; 100]
Temos linguísticos: *fechado*, *gotejando*, *aberto*
Função de pertinência: M = {(*fechado*, $\mu_{fechado}(t)$), (*gotejando*, $\mu_{gotejando}(t)$), (*aberto*, $\mu_{aberto}(t)$)}

onde,

$$\mu_{seco}(t) = \begin{cases} -0.02t + 1, & \text{se } t \geq 0 \text{ e } t < 50 \\ 0 & \text{se } t \geq 50 \text{ e } t \leq 100 \end{cases}$$

$$\mu_{gotejando}(t) = \begin{cases} 0.04t - 1, & \text{se } t \geq 25 \text{ e } t < 50 \\ -0.04t + 3, & \text{se } t \geq 50 \text{ e } t < 75 \\ 0 & \text{se } t \geq 75 \text{ e } t \leq 100 \\ 0 & \text{se } t \geq 0 \text{ e } t \leq 25 \end{cases}$$

$$\mu_{aberto}(t) = \begin{cases} 0.02t - 1, & \text{se } t \geq 50 \text{ e } t \leq 100 \\ 0 & \text{se } t \geq 0 \text{ e } t \leq 50 \end{cases}$$

4.2 Fuzificação

A interface de fuzificação faz a identificação dos valores das variáveis de entrada, as quais caracterizam o estado do sistema e normaliza no universo do discurso padronizado (*temperatura* [0; 50] e *umidade* [0; 1000]). Estes valores são transformados de entrada *crisp* em conjuntos nebulosos, para que possam se tornar instâncias de variáveis linguísticas.

A seguir são apresentados os cálculos aplicados ao processo de fuzificação para verificar pertinência dos valores numéricos de entrada:

Temperatura

Para transformação dos valores de entrada da variável linguística *temperatura* visto na Figura 5. Identificou-se os pontos (x, y) das funções de pertinência da variável linguística:

1. $\mu_{tri}(x; a, b, c) = max(min(x - a/b - a, c - x/c - b), 0)$;
2. $\mu_{tri}(x; a, b) = min(((a - x)/a)1)$;

Exemplos:

- $\mu_{baixa}(x; a) = min(((14 - x)/14), 1)$;
- $\mu_{media}(x; a, b, c) = max(min(x - 12/20 - 12, 28 - x/c - 20), 0)$;
- $\mu_{alta}(x; a, b, c) = max(min(x - 20/28 - 20, 36 - x/c - 28), 0)$;
- $\mu_{muito\_alta}(x; a, b) = min(((30 - x)/50 - 30)1)$;

Umidade do Solo

Para transformação dos valores de entrada da variável linguística *umidade* visto na Figura 6. Identificou-se os pontos (x, y) das funções pertinência da variável linguística:

1. $\mu_{tri}(x; a, b, c) = max(min(x - a/b - a, c - x/c - b), 0)$;
2. $\mu_{ram}(x; a, b) = min(((a - x)/a)1)$;

Exemplos:

- $\mu_{seco}(x; a) = min(((500 - x)/500), 1)$;
- $\mu_{umido}(x; a, b, c) = max(min(x - 250/500 - 250, 750 - x/750 - 500), 0)$;
- $\mu_{encharcado}(x; a, b) = min(((500 - x)/1000 - 500), 1)$;

4.3 Base de Conhecimento

A base de conhecimento consiste de uma base de dados construída com dados experimentais. A base de regras foi definida de acordo com estrutura do tipo: SE <premissa> E <premissa> ENTÃO <conclusão> conforme *FCL* do controlador que encontra-se no apêndice 10.1. Para este controlador foram definidas 12 regras. Estas com duas entradas (*temperatura* e *umidade*) e uma única saída (irrigacao). As premissas são relacionadas por conectivos lógicos dados pelo operador de conjunção (E).

4.4 Inferência

Nesta etapa aplicou-se o modelo clássico de inferência publicado por *Mamdani*. Visto que o mesmo aplica os modelos de operações t-norma e t-conorma, conforme descrito na subseção 2.2.4. A opção por este modelo deve-se ao fato de uma apresentar implementação mais adequados a base de conhecimento projetada. A t-norma min (a, b) é utilizado pra obtenção do valor dos temos linguísticos da variável *irrigacao* conforme exemplo:

SE *temperatura* = *alta* E *umidade* = *umido* ENTÃO irigacao = *gotejando*;

t-norma = min(*alta* = 1,00, *umido* = 0,50) = *gotejando* = 0,50;

Após obtenção dos diferentes valores de pertinência dos termos linguísticos aplica-se a t-conorma definida como o max (a, b), ou seja, o valor máximo do termo linguístico será o maior valor de pertinência da função de saída, conforme exemplo:

t-conorma = max (0,50, 0,00) = 0,50;

4.5 Defuzificação

O método aplicado no estágio de defuzificação foi o centro de gravidade, um dos que por, precisar de passos iterativos, para gerar uma aproximação, demandam mais computação e linhas de código. Para esta técnica foi aplicado um passo que gera uma saída com baixa granularidade (0.01), ou seja, no intervalo de 0 a 100 que correspondem os valores do eixo x da função da saída que terá os seguintes valores de x (0, 0.01, 0.02, 0.03, ..., 100) multiplicados pelo eixo y correspondente ao valor de pertinência identificado conforme Figura **??**. Após calculados os valores de pertinência é realizado o cálculo da razão do somatório do produto (passo*pertinência) dividido pelo somatório da pertinência. O resultado representa o percentual de abertura da válvula de irrigação.

5. IMPLEMENTAÇÃO DO CONTROLADOR

5.1 Mapeamento para o Sketch

O controle difuso referido foi implementado utilizando a linguagem de programação *C*, para codificação das entradas do sistema, as principais funções de pertinências, regras e saídas do sistemas. Foram utilizadas três bibliotecas *Arduino*, a *LiquidCristal.h* que permite a placa *Arduino* controlar um *LiquidCristal* monitor *LCD*. A biblioteca *DHT11.h* do sensor de temperatura que suporta uma função para leitura de temperatura, a função *read()* verifica a transmissão de dados, além de uma função de espera, com *DHT11.h* é possível a leitura de vários sensores desde que estejam em pinos separados. Outra biblioteca aplicada ao projeto foi a *Servo.h* que permite uma placa *Arduino* controle servo motores. Esta biblioteca permite o controle e posicionando em vários ângulos, geralmente entre 0 ° e 180 ° [14].

Para codificação da interface de fuzificação foram aplicados as operações apresentadas na seção 4.0.8. No Algoritmo 1 é demonstrado o trecho de código da função de fuzificação da variável linguística *umidade*. Na inferência como verificado no trecho de código Algoritmo 2, os valores obtidos através da operação t-norma são armazenados em vetores que representam cada termo linguístico da variável irrigação, após aplicação de todas regras cada vetor é percorrido aplicando a operação t-conorma para extração do valor de maior pertinência de cada termo linguístico. Por fim a defuzificação foi codificada conforme método do centro de gravidade apresentado na seção 4.0.11. No Algoritmo 3 mostra o trecho e código deste estágio.

Algoritmo 1: Codificação da etapa fuzificação

```
// Cálculo pertinência de leitura de umidade conjunto seco.
double funcaoSeco(double leitura_umi){
return min(1,((a_umi − leitura_umi)/(a1_umi)));
}
// Cálculo pertinência de leitura de umidade conjunto
umido.
double funcaoUmido (double leitura_umi) {
calc_A = ((leitura_umi − a2_umi)/(b2_umi − a2_umi));
calc_B = ((c2_umi − leitura_umi)/(c2_umi − b2_umi));
return max((min(calc_A,calc_B)), 0 );
}
// Cálculo pertinência de leitura de umidade conjunto
encharcado.
double funcaoEncharcado (double leitura_umi) return
min(1,((leitura_umi-a3_umi)
/(b3_umi-a3_umi)));
}
```

Algoritmo 2: Codificação da inferência

```
void tconormaAberto(){
for(int i=0;i<=cont2;i++){
aberto=max(aberto,vet_aberto[i]); }
}
void tconormaFechado(){
for(int i=0;i<=cont;i++){
fechado=max(fechado,vet_fechado[i]); }
}
void tconormaGotejando(){
for(int i=0;i<=cont1;i++){
gotejando=max(gotejando,vet_gotejando[i]); }
}
```

5.2 Hardware Projetado

O protótipo desenvolvido consiste num sistema de gerenciamento de irrigação. Neste trabalho foi utilizado a placa microcontrolada *Arduino Uno* visto na Figura 9 baseada no *ATmega328 8bit AVR* [15]. Possui 14 pinos digitais de entrada/saída, 6 entradas analógicas, uma conexão *USB*, um conector de alimentação. Para utilização da placa *Arduino* basta utilização alimentação via *USB* no computador, uma fonte *DC* ou uma bateria externa.

Dentre outros motivos para escoalha *Uno* destacam-se a facilidade para comunicação com outros *Arduinos* e outros microcontroladores através da interface de comunicação serial e digital fornecido pelo microcontrolador. Não é necessário nenhum tipo de drive externo, o software *Arduino* inclui um monitor serial que permite a simulação simples do *software* em execução na prototipagem *Uno* [14].

Algoritmo 3: Codificação da defuzificação

```
// Cálculo de pertinência de cada valor pertencente ao
conjunto fechado.
double funcaoFechado(double x){
return max(-x/50 + 1, 0);
}
// Cálculo de pertinência de cada valor pertencente ao
conjunto gotejando.
double funcaoGotejando(double x){
return max(min((x/25)-1, -x/25+3),0);
}
// Cálculo de pertinência de cada valor pertencente ao
conjunto aberto.
double funcaoAberto(double x){
return max(x/50-1, 0);
}
// Retorna pertinência de cada valor do conjunto fechado.
double pertFechado(double x, double pert){
return min(funcaoFechado(x), pert);
}
// Retorna pertinência de cada valor do conjunto
gotejando.
double pertGotejando(double x, double pert){
return min(funcaoGotejando(x), pert);
}
// Retorna pertinência de cada valor do conjunto aberto.
double pertAberto(double x, double pert)
return min(funcaoAberto(x), pert);
```

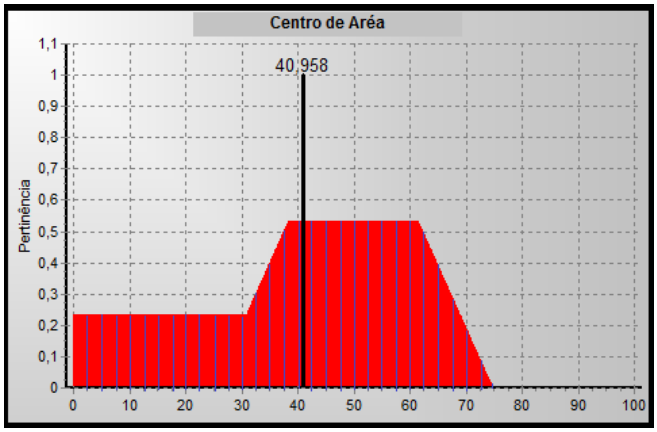

Figure 8: Gráfico de defuzificação.

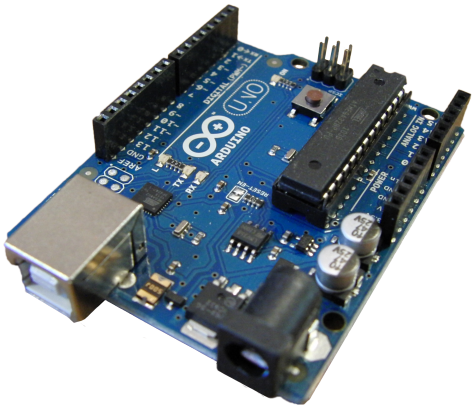

Figure 9: Arduino Uno

Os principais sensores utilizados no projeto são: sensor de temperatura *DHT11* e sensor de umidade do solo *BOT-06668 Moisture Sensor*, além *display LCD* 16x6 e um servo motor de parábola adaptado numa válvula registro com abertura com ângulo de 90 ° para o controle da vazão. Para modelagem do projeto os valores de entrada do controlador difuso são providos por sensores de umidade do solo e temperatura, gerando uma saída defuzificada para acionamento da válvula de vazão de água. A umidade do solo pode ser medida através do chamado de potencial hídrico do solo, ou seja, o volume de água no solo é diretamente proporcional a condutividade elétrica, os sensores de umidade do solo utilizados nesta pesquisa, medem a resistência do solo, ou seja, a variação de resistência depende da quantidade de água no solo [8].

Outra variável a ser analisada é a temperatura ambiente, tendo em vista quanto maior a temperatura maior a perda de água por evaporação, o sensor utilizado no projeto irá através da temperatura capturada gerar valores de entrada para o *Arduino*. No processo de defuzificação será gerado um valor de saída que representa a porcentagem de abertura da válvula para o gotejamento da água.

O *BOT-06668* Figura 10 utiliza duas sondas para passar a corrente através do solo, em seguida lê-se a resistência para obter o nível de umidade. Quanto maior o nível de água a eletricidade é conduzida mais facilmente (menor resistência), enquanto que o solo seco conduz eletricidade mal (mais resistência). Esse sensor é composto por três pinos que compõem saída analógica, GND e alimentação [20].

O *DHT11*, Figura 11 trata-se de um sensor de temperatura e umidade do ar, porém nesta pesquisa só serão utilizados seus recursos de leitura de temperatura do ambiente. Ele usa um sensor capacitivo de umidade e um termistor para medir o ar circulante [22]. Os dados de leitura *DHT11* devem ser medidos uma vez a cada 2 segundos, possui 3 pinos para conexão com *Arduino* o alimentação 3 ou 5V, dados e GNV, e possui uma margem de 2%.

5.3 Funcionamento do Protótipo

Figure 10: BOT-06668.

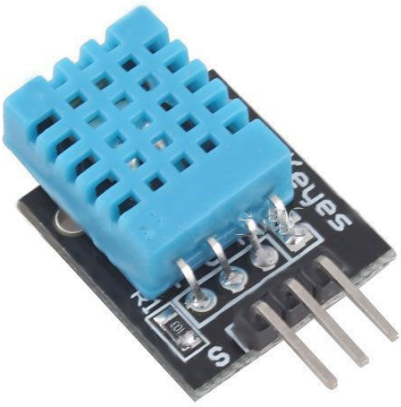

Figure 11: DHT11.

Figure 12: Protótipo do Controlador Fuzzy Arduino.

Na Figura 12 é apresentado o protótipo funcional do controlador projetado. A placa de circuito eletrônico projetada juntamente com a placa microcontrolada foi fixada a um compartimento feito de material reciclável de informática. Na parte superior do invólucro foi fixado o sensor de temperatura *DHT11* visando uma leitura mais precisa e sem interferências. Na lateral do compartimento estão dispostas a conexão *USB*, conector de fonte DC e um potenciômetro para ajuste do *display*. Para conexão com sensor de umidade do solo *BOT-06668* e a válvula proporcional existem cabos de dados e alimentação.

Durante o funcionamento do sistema o sensor de temperatura fixado ao compartimento do controlador, capta a temperatura atmosférica fornecendo uma das entradas discretas. Em paralelo o sensor de umidade fixado ao solo conforme Figura 12, realiza a leitura para fornecer a segunda entrada discreta ao microcontrolador de posse desses dados inicia-se o processo de controle *fuzzy* que irá fornecer uma saída discreta para acionamento da válvula proporcional de vazão de água. Esta válvula foi projetada utilizando um registro convencional de 1/4 de volta controlada por um servo motor de parábola. Para trabalhar em conjunto com a válvula o *software* foi projetado para acionamento do servo motor no ângulo de 90° proporcionando uma abertura de 1/4 de volta. Conforme o valor de saída do controlador a válvula abrirá proporcionalmente liberando água para irrigação.

6. VALIDAÇÃO E ANÁLISE DE RESULTADOS

6.1 Avaliação de Recursos

A avaliação dos recursos são definidas com base em requisitos não-funcionais visando verificar atributos não relacionados as funcionalidades do sistema, ou seja, são verificados requisitos como desempenho do produto correspondente ao uso da memória. Para análise destes requisitos investigou-se os impactos do controle *fuzzy* no microcontrolador. Para os experimentos modificou-se as configuração das variáveis linguísticas de entrada e saída. Na execução destas medições foram reduzidos os números de termos linguísticos das funções de pertinência de entrada e saída conforme Tabela 3. Na verificação destes cenários foi aplicada a ferramenta *AVR-size* para verificar em cada cenário proposto o impacto do consumo de memória na inicialização do código fonte no *Arduino Uno* que possui 2 *KB* de *SRAM* e 32 *KB* de memória *Flash*.

Em experimentos realizados com a solução final proposta os resultados são julgados satisfatórios observando as limitações do *Arduino Uno* quando comparado a outros modelos. Em verificações executadas neste panorama o resultado final medido demonstrou o consumo de 531 *bytes* de memória *SRAM* e 12030 *bytes* memória *Flash*.

Cenário 1: para esta medição foi retirando um termo linguístico da variável *temperatura*. O resultado alcançado foi a redução de 10 linhas de código que impactaram diretamente nas fases de fuzificação e inferência. Neste cenário o consumo de memória *SRAM* foi de 527 *bytes* e 11698 *bytes* de memória *Flash*.

Cenário 2: nesta conjuntura foi retirado um termo linguístico de saída da variável linguística *irrigacao*. A conclusão foi a subtração de 25 linhas de código, impactando nas fases de inferência e defuzificação. Com utilização de 503 *bytes* de *SRAM* e 11358 *bytes* de memória *Flash*.

Cenário 3: a análise neste instante foi realizada com base na subtração de um termo de cada variável linguística de entrada. Houve redução de 20 linhas de código, afetando a fuzificação e inferência. Quanto memória *SRAM* utilizou-se 523 *bytes* e 11388 *bytes* de memória *Flash*.

Cenário 4: neste experimento subtraiu-se um termo de cada variável de entrada e um termo da variável linguística de saída. Observou-se uma redução 44 linhas de código, neste cenário apenas a etapa de saída não foi impactada. A utilização de memória *SRAM* foi de 495 *bytes* e 10776 *bytes* de memória *Flash*.

Com base nos experimentos apresentados na Tabela 4 utilizando o *Arduino Uno*, foi possível constatar que a subtração ou adição de termos linguísticos de entrada e saída não afetaram de forma significativa o consumo inicial de memória *SRAM* e *Flash*. Para o controle *fuzzy* projetado a porcentagem de memória consumida foi de 25,92% de *SRAM* e 36,71% de memória *Flash* disponível.

A subtração de um termo linguístico representou uma diminuição 4 *bytes* de *SRAM* de 332 *bytes* de memória *Flash*, já a retirada de um termo de saída subtraiu 28 *bytes* de *SRAM* e 672 *bytes* da *Flash*. Após análise destes dados é possível inferir que o *Arduino* é uma boa solução para o controle proposto, podendo também ser aplicado em controles que requerem um maior número de temos linguísticos de entrada e saída, considerando o baixo percentual do consumo de memória na adição de termos linguísticos. Em um cenário com o modelo *Arduino Mega 2560* que possui 8 *KB* de *SRAM* e 256 *KB* de memória *Flash* sendo 8 *KB* usado no *bootloader*, o consumo de memória teria um percentual ainda menor de 6,48% de *SRAM* e 4,58% da memória *Flash*.

6.2 Descrição do Teste Funcional

No teste funcional foi avaliado o comportamento do protótipo em comparação com outros *softwares* para modelagem de controles difusos, aplicando os mesmos dados de entrada, para verificação dos valores de saída. Para realização dos testes foi desenvolvido um *software* na linguagem de programação *Java* utilizando a biblioteca *Random*, com o objetivo de gerar trinta valores de entrada pertencentes a variável *temperatura* no intervalo de 0 a 50 e trinta valores de entrada relacionados a variável *umidade* compreendendo valores ente 0 e 1000 conforme apêndice Tabela 5. Os resultados das saídas foram comparados com valores obtidos em *softwares* como o *InFuzzy* e *jFuzzyLogic*. Os testes funcionais aplicados justificam-se, devido a utilização de um cálculo de aproximação de área na etapa de defuzificação.

6.2.1 Controlador Fuzzy Arduino x jFuzzyLogic

Conforme experimentos descritos na Tabela 5 os dados de entrada gerados aleatoriamente foram inseridos no controlador e os resultados obtidos comparados com valores obtidos utilizando a ferramenta *jFuzzyLogic*, que fornece uma biblioteca completa *open source* para implementação de modelos para controles difusos [28]. Para realização dos testes sugeridos, foi construído um *FCL* descrito no apêndice 10.1, este padrão é utilizado na aplicação *jFuzzyLogic* para desenho das funções de pertinência de entrada, saída e lógica de inferência. Após configuração do *software* como os mesmos valores de entrada, foi possível verificar, que os resultados de saída obtidos no *jFuzzyLogic* foram compatíveis obtidos no protótipo.

6.2.2 Controlador Fuzzy Arduino x InFuzzy

O *InFuzzy* foi segundo *software* utilizado para confrontar os valores de entrada obtidos aleatoriamente, o mesmo é designado para modelagem de sistemas difusos [6]. Para estes testes aplicou-se os mesmos valores de entrada usados no primeiro teste funcional, os resultados obtido conforme ilustrado na Tabela 5, não apresentaram a mesma regularidade apresentada no primeiro teste funcional, dos trinta valores analisados no *InFuzzy*, 8 apresentaram falha na etapa de inferência resultando em falha na saída. Os 22 valores restantes apresentaram valores compatíveis aos do controlador.

6.2.3 Avaliação dos Testes Funcionais

Para análise resultados obtidos no teste funcional os valores de saída foram coletados e se usou o teste t-pareado conforme apêndice 10.2 para se verificar se as diferenças nos resultados de saída são estatisticamente relevantes. O que vimos foi que se gerarmos mais 30 valores no *jFuzzyLogic* e no protótipo, em 95% das vezes a diferença entre a média dos valores colhidos na saída estará entre 0.01634366 a 0.02298968. No caso do *InFuzzy* a amplitude foi maior considerando que 8 amostras do *InFuzzy* apresentaram problemas, neste contexto se gerarmos mais 30 valores, em 95% das vezes a diferença entre a média dos valores no protótipo e no *InFuzzy* colhidos na saída estará entre -5.462094 a 5.392760. Após análise detalhada do *InFuzzy* foi possível detectar uma falha no processo de inferência do mesmo.

Quando comparados o *InFuzzy* e o *jFuzzyLogic* se geradas mais 30 valores, em 95% das vezes a diferença e entre a média dos valores colhidos na saída estará entre -5.481102 a 5.372435. Conforme testes funcionais realizados com o *InFuzzy* e *jFuzzyLogic* possível inferir que os resultados obtidos com o controlador proposto demostram resultados compatíveis com os obtidos com *jFuzzyLogic* que não apresentou nenhum tipo de falha.

7. CONCLUSÃO

Fases	Código completo	Cenário 1	Cenário 2	Cenário 3	Cenário 4
Entrada de Dados	6	6	6	6	6
Fuzificação	85	78	84	71	70
Inferência	56	53	37	50	31
Defuzificação	31	31	26	31	27
Saída	16	16	16	16	16
Outros	22	22	22	22	22

Tabela 3: Detalhamento de linhas de código por cenários.

Memória	Código completo	Cenário 1	Cenário 2	Cenário 3	Cenário 4
Memória SRAM	531	527	503	523	495
Memória FLASH	12030	11698	11358	11388	10776
Total de Bootloader	512	512	512	512	512
Total de memória utilizada	12561	12225	11861	11911	11271
SRAM não utilizada	1517	1521	1545	1525	1553
FLASH não utilizada	20226	20558	20907	20868	21480

Tabela 4: Uso de memória em bytes.

As tecnologias utilizadas nos sistemas embarcados estão sempre em constante evolução proporcionando novas formas e soluções para o aprimoramento de controles. Esta evolução possibilita aplicar conceitos de Inteligência Artificial para a melhoria dos processos e resoluções de problemas não suportados por teorias matemáticas convencionais. Utilizando os conhecimentos da lógica difusa, uma das principais áreas de conhecimentos da IA, foi desenvolvido um protótipo embarcado no *Arduino* que disponibiliza uma plataforma livre e flexível. Este trabalho demonstra os benefícios provenientes da aplicação de controle difuso num cenário de irrigação.

Os conhecimentos de controle difuso nortearam todo o processo de construção do produto, desde as leituras dos valores discretos através de sensores, fuzificação dos valores em formas linguísticas e a aplicação das regras de inferência. Finalmente, na etapa de defuzificação é gerada uma saída discreta para acionamento da válvula de vazão para irrigação.

Após avaliação do consumo de recursos do protótipo, verificou-se a viabilidade de aplicar microcontroladores em controles difusos, o que poderia ser um fator complicador do projeto, considerando as limitações de processamentos e memória deste modelo, quando comparado *hardwares Arduino* mais robustos, como verificado na Tabela 1. De posse destas informações, constatou-se que é possível embarcar um controle *fuzzy* em um *Arduino*, tendo em vista o consumo de processamento e memória utilizado no protótipo. Os testes funcionais também demostraram que os valores obtidos nas populações de amostras testadas, são estatisticamente compatíveis com as amostras simuladas na ferramenta *jFLogicLogic*.

Por fim, conclui-se que o uso da tecnologia de controle *fuzzy* embarcado é uma boa solução para controle de sistemas automatizadas. Por apresentar vantagens como: a simplificação do modelo de processos, o tratamento das imprecisão inerentes aos sensores utilizados, a facilidade na especificação das regras de controle em uma linguagem próxima da natural, satisfação de múltiplos objetivos de controle e a facilidade de incorporação do conhecimento de especialistas humanos. Os resultados despertam possibilidades de criação de sistemas de controles mais robustos em virtude do percentual de consumo de recursos, na adição o subtração de termos linguísticos de entrada e saída. O produto final desenvolvido, forneceu os subsídios necessários para validarmos a aplicação destas técnicas no controle de irrigação, assim como em outros sistemas automatizados.

8. TRABALHOS FUTUROS

Como proposta para trabalhos futuros, pode-se pensar na implementação da geração automática de código para *Arduino* através da configuração do *FCL*. A proposta é utilizar os princípios aplicados em *softwares* como *JFuzzLogic* para simulação de controle *fuzzy*, visando aplicá-los na geração do *sketch .ino* utilizado no *Arduino*.

9. REFERÊNCIAS

[1] L. A. Zadeh. Fuzzy Sets, Information and Control,1965.

[2] S. Sandri and C. Correia. Lógica Nebulosa. V Escola de Redes Neurais, São José dos Campos – SP, Julho 1999.

[3] G. Bittencourt. Inteligencia Artificial: Ferramentas e Teorias. 2º ed, Editora Da UFSC, 2001.

[4] S. R. Ball. Embedded Microprocessor Systems: Real World Design, 2002.

[5] F. A. C Gomide and P. G Cohn and M. L. Marques. Automação, Controle e Inteligência Artificial, Parte I: Uma Visão Integrada, 2005.

[6] E. L. Posselt. INFUZZY - Ferramenta para Desenvolvimento de Aplicações de Sistemas Difusos, Santa Cruz do Sul, Abril de 2011.

[7] M. Andrade and M. A. P. Jacques. Estudo Comparativo de Controladores de Mamdani e Sugeno para Controle do Tráfego em Interseções Isoladas, Brasília, 1999.

[8] M. McRoberts. Arduino Básico. 1ª ed, Novatec Editora, São Paulo – SP, Setembro 2011.

[9] Q. Zhang, C. Hwa and K.Tilt. Application of Fuzzy Logic in na Irrigation Control System, IEEE International Conference on Industrial Technology,

1996.

[10] M. R. B G. Vale. Análise comparativa do desempenho de um Controlador Fuzzy acoplado a um PID Neural sintonizado por um Algoritmo Genético com Controladores Inteligentes Convencionais, Natal – RN 2007.

[11] R. Zelenovsky. Emprego de Microcontroladores na Cadeira de Arquitetura de Computadores, Brasília – DF, 2004.

[12] L. M. Rodrigues and G.P. Dimuro. Utilizando Lógica Fuzzy para Avaliar a Qualidade de uma Compra Via Internet, Programa de Pós-Graduação em Modelagem Computacional Universidade Federal do Rio Grande, Porto Alegre - RS, 2011.

[13] R. Bahat and G. Inbar and O. Yaniv and M. Schneider. A fuzzy irrigation controller system, Engineering Applications of Artificial Intelligence, 2000.

[14] Arduino Uno. http://arduino.cc/en/Main/arduinoBoardUno. Acessado em 01/07/2014.

[15] Microcontroller ATmega328. http://www.atmel.com/Images/doc8161.pdf Acessado em 10/06/2014.

[16] F. A. C. Gomide and R. R. Gudwin and R. Tanscheit. Conceitos Fundamentais da Teoria de Conjuntos Fuzzy, Lógica Fuzzy e Aplicações, Campinas - SP, 2001.

[17] I. Baturone and F. J. Moreno-Velo and V. Blanco, and Joaquín Ferruz. Design of Embedded DSP-Based Fuzzy Controllers for Autonomous Mobile Robots, Lógica Fuzzy e Aplicações, IEEE Transactions on Industrial Eletronics, VOL. 55, NO. 2, Fevereiro 2008.

[18] Chia-Feng Juang and Jung-Shing Chen. Water Bath Temperature Control by a Recurrent Fuzzy Controller andIts FPGA Implementation, IEEE Transactions on Industrial Eletronics,VOL. 53, NO. 3, Junho 2006.

[19] M. Sasaki and F.Ueno and T. Inoue. 7.5MFLIPS Fuzzy Microprocessor Using SIMD and Logic-in-Memory Structure, IEEE Conference Publications, Kumamoto, Kurokami 2-39-1, 860 Japan, 1993.

[20] Soil Moisture Sensor. http://www.geeetech com/garden-tool-soil-moisture-sensor-p-598. html. Acessado em 11/07/2014

[21] Fuzzy Control Language. http://ffll.source forge.net/fcl.html. Acessado em 21/08/2014.

[22] D-Robotics. DHT11 Humidity e Temperature Sensor, Janeiro 2010.

[23] A. Changuel and R. Rolland and A. A. Jerraya. Design of an Adaptive Motors Controllerbased on Fuzzy Logic using Behavioural Synthesis Design Automation Conference, Europa 1996.

[24] L.M. Reyneri and M. Chiaberge and L. Zocca. CINTIA: A Neuro-Fuzzy Real Time Controller for Low Power Embedded Systems, Torino 1995.

[25] D.Fodor and J. Vass and Z.Katona. Fuzzy Logic Based Vector Controlled AC Drive Using Embedded DSP-Controller, Optimization of Electrical and Electronic Equipments, Proceedings of the 6th International Conference on (Volume:2), Brasov 1998.

[26] R. Raji and Y.G. Srinivasa and A. Kistner. Development of a Neuron chip based Smart Embedded Fuzzy Logic Controller with Intelligent Control Strategies for Process Control, Fuzzy Systems, The 10th IEEE International Conference on Volume: 3, 2001.

[27] N.S. Kumar and V. Sadasivam and K. Prema. Design and Simulation of Fuzzy Controller for Closed Loop Control of chopper fed Embedded DC drives, Power System Technology, PowerCon 2004. International Conference on Volume:1, 2004.

[28] P. Cingolani and J. Alcalá-Fdez. jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming, International Journal of Computational Intelligence Systems, Vol. 6, 2013.

10. APÊNDICE

10.1 FCL

```
FUNCTION_BLOCK irrigador

VAR_INPUT
    temperatura : REAL;
    umidade : REAL;
END_VAR

VAR_OUTPUT
    irrigacao : REAL;
END_VAR

FUZZIFY temperatura
    TERM baixa := (0, 1) (14, 0);
    TERM media := (12, 0) (20, 1) (28, 0);
    TERM alta := (20, 0) (28, 1) (36, 0);
    TERM muito_alta := (30, 0) (50, 1);
END_FUZZIFY

FUZZIFY umidade
    TERM seco := (0, 1) (500, 0);
    TERM umido := (250, 0) (500, 1) (750, 0);
    TERM encharcado := (500, 0) (1000, 1);
END_FUZZIFY

DEFUZZIFY irrigacao
    TERM fechado := (0,1) (50,0);
    TERM gotejando := (25,0) (50,1) (75,0);
    TERM aberto := (50,0) (100,1);
    METHOD : COG;
    DEFAULT := 0;
END_DEFUZZIFY

RULEBLOCK No1
    AND : MIN;
    ACT : MIN;
    ACCU : MAX;
    RULE 1 : IF temperatura IS baixa AND umidade
             IS seco THEN irrigacao IS gotejando;
    RULE 2 : IF temperatura IS media AND umidade
             IS seco THEN irrigacao IS aberto;
    RULE 3 : IF temperatura IS alta AND umidade
             IS seco THEN irrigacao is aberto;
    RULE 4 : IF temperatura IS muito_alta AND
             umidade IS seco THEN irrigacao
IS aberto;
    RULE 5 : IF temperatura IS baixa AND umidade
```

```
            IS umido THEN irrigacao IS fechado;
    RULE 6 : IF temperatura IS media AND umidade
            IS umido THEN irrigacao IS gotejando;
    RULE 7 : IF temperatura IS alta AND umidade
            IS umido THEN irrigacao is gotejando;
    RULE 8 : IF temperatura IS muito_alta AND
            umidade IS umido THEN irrigacao
is aberto;
    RULE 9 : IF temperatura IS baixa AND umidade
            IS encharcado THEN irrigacao IS
fechado;
    RULE 10 : IF temperatura IS media AND umidade
             IS encharcado THEN irrigacao IS
fechado;
    RULE 11 : IF temperatura IS alta AND umidade
             IS encharcado THEN irrigacao IS
 fechado;
    RULE 12 : IF temperatura IS muito_alta AND
             umidade IS encharcado THEN irrigacao
 IS gotejando;


END_RULEBLOCK

END_FUNCTION_BLOCK
```

10.2 Teste T Pareado

```
R version 3.1.1 (2014-07-10) -- "Sock it to Me"
Copyright (C) 2014 The R Foundation for
Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO
 WARRANTY.
You are welcome to redistribute it under certain
 conditions.
Type 'license()' or 'licence()' for distribution
 details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in
publications.

  Type 'demo()' for some demos, 'help()' for
on-line help, or
  'help.start()' for an HTML browser interface to
  help.
  Type 'q()' to quit R.

> amostraJFuzzy = c(43.78, 61.56, 19.67, 37.76,
  79.3, 82.94, 36.15, 19, 79.76, 79.56, 60.46,
27.09, 18.34, 30.84, 80.52, 81.91, 19.31, 35.03,
18.80, 81.60, 50, 80.86, 79.30, 50, 50.84, 49.77,
80.52, 80.57, 44.98, 81.80)
> summary(amostraJFuzzy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.34   35.31   50.42   54.73   80.33   82.94
> amostraPrototipo = c(43.8, 61.58, 19.69, 37.78,
  79.32,82.96, 36.18, 19.03, 79.78, 79.58, 60.48,
27.12, 18.37, 30.86, 80.55, 81.94, 19.33, 35.05,
18.81, 81.63,50, 80.88, 79.32, 50, 50.85, 49.77,
80.55, 80.59,m44.99, 81.82)
> summary(amostraPrototipo)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.37   35.33   50.42   54.75   80.36   82.96
> t.test(amostra1JFuzzy, amostraPrototipo, paired
  = TRUE)
Error in t.test(amostra1JFuzzy, amostraPrototipo,
   paired = TRUE) :
   object 'amostra1JFuzzy' not found
> t.test(amostraJFuzzy, amostraPrototipo, paired =
 TRUE)

            Paired t-test

data:  amostraPrototipo and amostraJFuzzy
t = 12.1043, df = 29, p-value = 7.34e-13
alternative hypothesis: true difference in means
is not equal to 0
95 percent confidence interval:
 0.01634366 0.02298968
sample estimates:
mean of the differences
            0.01966667

> amostraInFuzzy = c(43.79, 61.58, 19.69, 37.78,
  79.32,82.96, 49.99, 34.32, 50, 50, 60.48, 27.12,
18.37, 49.99,80.55, 81.93, 19.33, 49.99, 49.99,
81.62, 63.38, 80.88,79.32, 50, 50, 49.77, 50,
80.59, 60.91, 50)
> summary(amostraInFuzzy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.37   49.82   50.00   54.79   75.33   82.96
> summary(amostraJFuzzy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.34   35.31   50.42   54.73   80.33   82.94
> summary(amostraPrototipo)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.37   35.33   50.42   54.75   80.36   82.96
> t.test(amostraPrototipo, amostraJFuzzy, paired
  = TRUE)

        Paired t-test

data:  amostraPrototipo and amostraJFuzzy
t = 12.1043, df = 29, p-value = 7.34e-13
alternative hypothesis: true difference in means
is not equal to 0
95 percent confidence interval:
 0.01634366 0.02298968
sample estimates:
mean of the differences
            0.01966667
> t.test(amostraPrototipo, amostraInFuzzy, paired
  = TRUE)

        Paired t-test

data:  amostraPrototipo and amostraInFuzzy
t = -0.0131, df = 29, p-value = 0.9897
alternative hypothesis: true difference in means
is not equal to 0
95 percent confidence interval:
```

```
 -5.462094  5.392760
sample estimates:
mean of the differences
            -0.03466667

> t.test(amostraJFuzzy, amostraInFuzzy, paired
  = TRUE)

        Paired t-test

data:  amostraJFuzzy and amostraInFuzzy
t = -0.0205, df = 29, p-value = 0.9838
alternative hypothesis: true difference in means
is not equal to 0
95 percent confidence interval:
 -5.481102  5.372435
sample estimates:
mean of the differences
            -0.05433333

>
```

10.3 Código Fonte do Contrlador

```
#include <LiquidCrystal.h>
#include <dht11.h>
#include <Servo.h>
#define dht_dpin A1
#define Luz_Fundo  7

// Definição dos valores das
Funções de Pertinência
#define a1_temp 14.0
#define a2_temp 12.0
#define b2_temp 20.0
#define c2_temp 28.0
#define a3_temp 20.0
#define b3_temp 28.0
#define c3_temp 36.0
#define a4_temp 30.0
#define b4_temp 50.0
#define a1_umi 500.0
#define a2_umi 250.0
#define b2_umi 500.0
#define c2_umi 750.0
#define a3_umi 500.0
#define b3_umi 1000.0
#define a1_val 50.0
#define a2_val 25.0
#define b2_val 50.0
#define c2_val 75.0
#define a3_val 50.0
#define b3_val 100.0

//Definição das Variáveis globais

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int cont,cont1,cont2;
Servo motor;
dht11 DHT11;
double centro_de_gravidade;
double calc_A, calc_B;
double calc_c, calc_d;
double seco;
double umido;
double encharcado;
double baixa;
double media;
double alta;
double muito_alta;
double fechado;
double gotejando;
double aberto;
double vet_aberto[6];
double vet_fechado[6];
double vet_gotejando[6];

void setup(){
  motor.attach(6);
  motor.write(90);
  dht11 DHT11;
  lcd.begin(16, 2);
  pinMode(Luz_Fundo,OUTPUT);
  digitalWrite(Luz_Fundo,HIGH);
  Serial.begin(9600);
  lcd.setCursor(0,1);
  lcd.print("Fuzzy Control");
  delay(10000);
  lcd.clear();

}


void loop(){

  lcd.setCursor(0,0);
  DHT11.read(dht_dpin);
  delay(2000);
  fuzificacaoTemperatura();
  fuzificacaoUmidade();
  inferencia();
  defuzificacao();

}

// Cálculo pertinência de letitura de
   umidade conjunto seco.
double funcaoSeco(double leitura_umi){
 return min(1,((a1_umi-leitura_umi)/(a1_umi)));
}

// Cálculo pertinência de letitura de umidade
   conjunto umido.
double funcaoUmido (double leitura_umi){
 calc_A=((leitura_umi-a2_umi)/(b2_umi-a2_umi));
 calc_B=((c2_umi-leitura_umi)/(c2_umi-b2_umi));
 return max((min(calc_A,calc_B)), 0 );
}

// Cálculo pertinência de letitura de umidade
   conjunto encharcado.
double funcaoEncharcado (double leitura_umi){
 return min(1,((leitura_umi-a3_umi)/
```

```
 (b3_umi-a3_umi)));
 }

// Metódo de Fuzificação Umidade
void fuzificacaoUmidade(){

  double leitura_umidade=0.0;
  leitura_umidade=(double)analogRead(0);

  int mostra_umi=(int)leitura_umidade;

  Serial.println("Umidade:");
  Serial.println(mostra_umi);
  lcd.print("Umidade do Solo");
  lcd.setCursor(0,1);
  lcd.print(mostra_umi);
  delay(3000);
  lcd.clear();

 seco=funcaoSeco(leitura_umidade);
 umido=funcaoUmido(leitura_umidade);
 encharcado=funcaoEncharcado(leitura_umidade);

}

// Cálculo pertinência de letitura de temperatura
   conjunto baixa.
double funcaoBaixa(double leitura){
  return min(1,((a1_temp-leitura)/(a1_temp)));

 }

// Cálculo pertinência de letitura de
   temperatura conjunto media.
double funcaoMedia(double leitura){
  calc_c=((leitura-a2_temp)/(b2_temp-a2_temp));
  calc_d=((c2_temp-leitura)/(c2_temp-b2_temp));
  return max((min(calc_c,calc_d)), 0 );
 }

// Cálculo pertinência de letitura de
   temperatura conjunto alta.
double funcaoAlta(double leitura){
  calc_c=((leitura-a3_temp)/(b3_temp-a3_temp));
  calc_d=((c3_temp-leitura)/(c3_temp-b3_temp));
  return max((min(calc_c,calc_d)), 0 );
 }

// Cálculo pertinência de letitura de
   temperatura conjunto muito_alta.
double funcaoMuitoAlta(double leitura){
  return min(1,((leitura-a4_temp)/(b4_temp-a4_temp)));
 }


// Metódo de Fuzificação Umidade
void fuzificacaoTemperatura(){

  double leitura_temp;
  leitura_temp=DHT11.temperature;

  int mostra_temp=(int)leitura_temp;
  Serial.println(" Temperatura ");
  Serial.print(leitura_temp);
  Serial.println("Celsius = ");
  lcd.setCursor(0,0);
  lcd.print("Temperatura");
  lcd.setCursor(0,1);
  lcd.print(mostra_temp);
  lcd.print(" C");
  delay(3000);
  lcd.clear();

  baixa=funcaoBaixa(leitura_temp);
  media=funcaoMedia(leitura_temp);
  alta=funcaoAlta(leitura_temp);
  muito_alta=funcaoMuitoAlta(leitura_temp);

}

void  tconormaAberto(){
   for(int i=0;i<=cont2;i++){
   aberto=max(aberto,vet_aberto[i]);
   }
}
void  tconormaFechado(){
   for(int i=0;i<=cont;i++){
   fechado=max(fechado,vet_fechado[i]);
   }
}
void  tconormaGotejando(){
   for(int i=0;i<=cont1;i++){
   gotejando=max(gotejando,vet_gotejando[i]);
   }

}

// Metódo de Inferência
void inferencia(){
  cont=0,cont1=0,cont2=0;
  aberto=0.0;
  fechado=0.0;
  gotejando=0.0;

 //Cálculo t-norma metódo min(a,b)definido por Zadhe
  vet_fechado[cont]=min(umido,baixa); cont++;
  vet_fechado[cont]=min(encharcado,baixa); cont ++;
  vet_fechado[cont]=min(encharcado,media);cont++;
  vet_fechado[cont]=min(encharcado,alta);cont++;
  vet_gotejando[cont1]=min(seco,baixa);cont1++;
  vet_gotejando[cont1]=min(\textit{,media);cont1++;
  vet_gotejando[cont1]=min(umido,alta);cont1++;
  vet_gotejando[cont1]=min(encharcado,muito_alta);
cont1++;
  vet_aberto[cont2]=min(seco,media); cont2++;
  vet_aberto[cont2]=min(seco,alta); cont2++;
  vet_aberto[cont2]=min(seco,muito_alta); cont2++;
  vet_aberto[cont2]=min(umido,muito_alta); cont2++;

  tconormaAberto();
  tconormaGotejando();
  tconormaFechado();

  Serial.println(aberto);
  Serial.println(fechado);
  Serial.println(gotejando);
```

```
}

// Cálculo de pertinência de cada valor
   pertencente ao ao conjunto fechado.
double funcaoFechado(double x){
 return max(-x/50 + 1, 0);
}
// Cálculo de pertinência de cada valor
   pertencente ao ao conjunto gotejando.
double funcaoGotejando(double x){
 return max(min((x/25)-1, -x/25+3),0);
}

// Cálculo de pertinência de cada valor
   pertencente ao ao conjunto aberto.
double funcaoAberto(double x){
 return  max(x/50-1, 0);
}

// Retorna pertinência de cada valor do
   conjunto fechado.
double pertFechado(double x, double pert){
 return min(funcaoFechado(x), pert);
}

// Retorna pertinência de cada valor do
   conjunto gotejando.
double pertGotejando(double x, double pert){
 return min(funcaoGotejando(x), pert);
}

// Retorna pertinência de cada valor do
   conjunto aberto.
double pertAberto(double x, double pert){
 return min(funcaoAberto(x), pert);
}

//Cálculo da defuzificação pelo metódo Centro
 de Gravidade.
double cog(double fPertFechado, double
          fPertGotejando,
double fPertAberto,double menorValor,
       double maiorValor,double passo){
 double atual = menorValor;
 double area = 0;
 double peso = 0;

  for(atual = menorValor; atual <= maiorValor;
atual+=passo){
double valorFechado = pertFechado(atual,
fPertFechado);
double valorUmido = pertGotejando(atual,
fPertGotejando);
double valorAberto = pertAberto(atual,
fPertAberto);
double valor = max(max(valorFechado,
valorUmido), valorAberto);
       peso += valor;
       area+= (valor * atual);
}
return area / peso;
}
```

```
 // Metódo de Defuzificação
void defuzificacao(){
   double saida;
    //Retorno do cálculo do Centro de Gravidade
   centro_de_gravidade= cog(fechado, gotejando,
 aberto, 0, 100, 0.01);

   saida=90-((90.0*centro_de_gravidade)/100.0);
   Serial.println("CG Total");
   Serial.println(centro_de_gravidade);
   motor.write(saida);
   Serial.println(saida);
   lcd.setCursor(0,0);
   lcd.print("Irrigacao");
   lcd.setCursor(0,1);
   lcd.print(centro_de_gravidade);
   lcd.print(" %");
   delay(3000);
   lcd.clear();
}
```

10.4 Tabela de Validação

Teste	Temperatura	Umidade	Controlador Proposto	jFuzzy	InFuzzy
1	27	584	43,8	43,78	43,79
2	33	371	61,58	61,56	61,58
3	28	749	19,69	19,67	19,69
4	17	648	37,78	37,76	37,78
5	15	160	79,32	79,3	79,32
6	27	80	82,96	82,94	82,96
7	11	376	36,18	36,15	49,99
8	2	773	19,03	19	34,32
9	39	290	79,78	79,76	50
10	38	69	79,58	79,56	50
11	31	368	60,48	60,46	60,48
12	20	716	27,12	27,09	27,12
13	17	865	18,37	18,34	18,37
14	6	367	30,86	30,84	49,99
15	20	250	80,55	80,52	80,55
16	30	167	81,94	81,91	81,93
17	19	756	19,33	19,31	19,33
18	10	370	35,05	35,03	49,99
19	6	588	18,81	18,8	49,99
20	25	97	81,63	81,6	81,62
21	12	221	50	50	63,38
22	23	232	80,88	80,86	80,88
23	33	129	79,32	79,3	79,32
24	42	847	50	50	50
25	39	746	50,85	50,84	50
26	21	503	49,77	49,77	49,77
27	40	145	80,55	80,52	50
28	18	248	80,59	80,57	80,59
29	3	276	44,99	44,98	60,91
30	43	162	81,82	81,8	50

Tabela 5: Validação de funcionamento.