PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

CEATEC

FACULDADE DE ENGENHARIA ELÉTRICA

TIAGO TREVISANI GANSELLI

IMPLEMENTAÇÃO DE SISTEMAS BASEADOS EM REGRAS NEBULOSAS POR MÉTODO MATRICIAL EM DISPOSITIVOS EMBARCADOS

PUC-CAMPINAS

2014

TIAGO TREVISANI GANSELLI

IMPLEMENTAÇÃO DE SISTEMAS BASEADOS EM REGRAS NEBULOSAS POR MÉTODO MATRICIAL EM DISPOSITIVOS EMBARCADOS

Dissertação apresentada como exigência para obtenção do Título de Mestre em Engenharia Elétrica, ao Programa de Pós Graduação Stricto Sensu em Engenharia Elétrica, Pontifícia Universidade Católica de Campinas.

Orientador: Prof. Dr. Alexandre de Assis Mota

PUC-CAMPINAS

2014

Ficha Catalográfica
Elaborada pelo Sistema de Bibliotecas e
Informação - SBI - PUC-Campinas

t005.1
G199i

Ganselli, Tiago Trevisani.
Implementação de sistemas baseados em regras nebulosas por método matricial em dispositivos embarcados / Tiago Trevisani Ganselli. - Campinas: PUC-Campinas, 2014.
130p.

Orientador: Alexandre de Assis Mota.
Dissertação (mestrado) – Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Pós-Graduação em Engenharia Elétrica.
Inclui bibliografia.

1. Algoritmo de computador. 2. Matrizes (Matemática). 3. Programação linear. 4. Sistemas difusos. 5. Lógica difusa. I. Mota, Alexandre de Assis. II. Pontifícia Universidade Católica de Campinas. Centro de Ciências Exatas, Ambientais e de Tecnologias. Pós-Graduação em Engenharia Elétrica. III. Título.

22.ed. CDD – t005.1

TIAGO TREVISANI GANSELLI

IMPLEMENTAÇÃO DE SISTEMAS BASEADOS EM REGRAS NEBULOSAS POR MÉTODO MATRICIAL EM DISPOSITIVOS EMBARCADOS

Dissertação apresentada ao Curso de Mestrado Profissional em Gestão de Redes de Telecomunicações do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas como requisito parcial para obtenção do título de Mestre em Gestão de Redes de Telecomunicações.

Área de concentração: Gestão de Redes e Serviços

Orientador: Prof. Dr. Alexandre de Assis Mota

Dissertação defendida e aprovada em ____ de _____ de _____ pela Comissão Examinadora constituída dos seguintes professores:

_____

Prof. Dr. Alexandre de Assis Mota

Orientador da Dissertação e Presidente da Comissão Examinadora

Pontifícia Universidade Católica de Campinas (PUC-Campinas)

_____

Profª. Drª. Lia Toledo Moreira Mota

Pontifícia Universidade Católica de Campinas (PUC-Campinas)

_____

Prof. Dr. Anibal Tavares de Azevedo

Universidade Estadual de Campinas (UNICAMP)

Dedico este trabalho aos pesquisadores do Brasil. O homem cresce ao aprender. O aprendizado nos mostra um objetivo a ser perseguido. O homem, quando não estuda, não cresce.

AGRADECIMENTOS

Agradeço aos meus colegas de classe e professores pelo empenho, suporte e companheirismo nestes anos de estudos e superação. Em especial, agradeço ao colega Daniel Braga pela parceria em todos os momentos do curso, desde a realização dos testes iniciais até os momentos finais da redação desta Dissertação;

À professora Lia Mota pelas observações e opiniões, sempre bem vindas, que me ajudaram a meditar sobre os muitos detalhes que teimam em passar despercebidos;

Ao meu orientador, Alexandre Mota, pela tranquilidade, paciência e metodologia de ensino excepcional que constantemente promove o debate e interação entre todos os alunos (de todos os níveis) do grupo de pesquisa. Tê-lo como orientador possibilitou alcançar um nível de agregação de conhecimento que eu jamais imaginava ser capaz;

À PUC Campinas, pelo auxílio da bolsa de estudos e pelo excelente ambiente proporcionado pelas instalações e laboratórios do Campus I;

Ao CPqD, pelas horas de trabalho cedidas para realização de tarefas indispensáveis para o mestrado, como a qualificação e defesa;

Ao colega Jefferson Capovilla por compartilhar, de forma natural e dedicada, o seu conhecimento sobre Linux, essencial para a construção deste trabalho;

Ao meu amigo Ricardo Carmo, pelo constante incentivo ao crescimento pessoal e intelectual, o que possibilitou que muitos objetivos fossem alcançados no decorrer dos últimos anos;

À minha noiva, Fabiana Mansano, pelo suporte, paciência, compreensão, companheirismo e amor compartilhados comigo durante toda a minha jornada na Engenharia, até este momento, e nos muitos que estão por vir;

Ao meu pai, Antonio, por ser para mim um modelo do que eu busco me tornar, incentivando sempre a busca pela realização profissional, intelectual e familiar de forma íntegra e justa para com o próximo;

Finalmente, agradeço à minha mãe, Maria Silvia, pelo suporte incondicional ao bem estar do nosso Lar, dedicando todo o seu esforço para sempre nos entregar as melhores condições de vida possíveis. Sem a sua ajuda, eu jamais chegaria onde estou.

"Se la gente sapesse quanto ho lavorato per raggiungere il mio dominio, non sembrerebbe così meraviglioso, dopo tutto."

“Se as pessoas soubessem o quanto trabalhei para adquirir a maestria, ela não pareceria tão maravilhosa, afinal.”

Michelangelo
(1475 - 1564)

RESUMO

Com a crescente necessidade de dispositivos com maior capacidade de processamento e menor consumo energético faz-se necessário o uso de algoritmos otimizados, permitindo o máximo aproveitamento dos recursos disponíveis na aplicação. Neste trabalho foi realizada a implementação do Método Matricial para execução de cálculos usando Lógica Nebulosa em dispositivos embarcados, tornando possível a tomada de decisão local nas mais diversas aplicações. Foram desenvolvidos códigos para o Scilab, Arduino e para a distribuição de Linux embarcado OpenWRT, que foram testados em dispositivos reais através da comparação com o código original do Método Matricial e com o estudo de caso da Anomalia da MAC em redes IEEE 802.11. Os resultados obtidos indicam que o Método Matricial é compatível com o uso em sistemas embarcados, sendo necessária a análise e configuração específica de cada aplicação para que o melhor desempenho seja alcançado. Concluiu-se que o balanceamento entre a tomada de decisão e a precisão do resultado é essencial para realizar o cálculo com o menor consumo de recursos possível. Espera-se que outros trabalhos façam uso dos algoritmos criados, a fim de auxiliar na tomada de decisão em dispositivos embarcados nas inúmeras aplicações emergentes.

Palavras-Chave — Método Matricial, Lógica Nebulosa, Fuzzy, Dispositivos Embarcados, Algoritmos embarcados.

ABSTRACT

It is known that the need for devices with higher processing capacities and low power consumption is increasing, making algorithm optimization necessary to allow the maximum utilization of the application's resources. In this work, the Matrix Method was implemented in embedded systems to solve Fuzzy calculations, allowing the decision-making process to be included in several applications. Code was developed for Scilab, Arduino, and the embedded Linux distribution OpenWRT, being tested in real devices through the comparison with the original Matrix Method algorithm implementation and the case study of the MAC anomaly in IEEE 802.11 networks. Results show that the Matrix Method is compatible for use in embedded systems, and the analysis and specific configuration of each application are necessary for the best performance to be achieved. Conclusion shows that the balance between the decision-making and the result precision is essential to lower resource consumption to the maximum. It is expected that other studies make use of the created algorithms, assisting the decision-making process in embedded systems for the countless emerging applications.

Key Words — Matrix Method, Fuzzy Logic, Embedded Devices, Embedded Algorithms.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AP	– *Access Point*
ASCII	– *American Standard Code for Information Interchange*
CSMA/CA	– *Carrier Sense Multiple Access with Collision Avoidance*
DCF	– *Distributed Coordination Function*
EDCA	– HCF/MCF *Contention Access*
FLS	– *Fuzzy Logic System*
FPU	– *Floating Point Unit*
HCCA	– HCF *Controlled Access*
HCF	– *Hybrid Coordination Function*
IEEE	– *Institute of Electrical and Electronics Engineers*
IoT	– *Internet of Things*
MAC	– *Medium Access Control*
MB/s	– Mega Bytes por Segundo
MCCA	– MCF *Controlled Access*
MCF	– *Mesh Coordination Function*
MIPS	– *Microprocessor without Interlocked Pipeline Stages*
OSI	– *Open Systems Interconnection*
PCF	– *Point Coordination Function*
PHY	– *Physical Layer*
QoS	– *Quality of Service*
RF	– Radio Frequência
SNR	– *Signal to Noise Relation*
SO	– Sistema Operacional
SoC	– *System on Chip*
TI	– Tecnologia da Informação
WLAN	– *Wireless Local Area Network*
WMAN	– *Wireless Metropolitan Area Network*
WPAN	– *Wireless Personal Area Network*

SUMÁRIO

1. INTRODUÇÃO

É crescente a necessidade do uso de dispositivos embarcados nas atividades do dia-a-dia. Estes aparelhos, geralmente pequenos e com reduzida capacidade de processamento, podem passar despercebidos aos olhos pouco atentos, porém eles estão se tornando cada vez mais numerosos. Sistemas embarcados podem ser encontrados em carros, eletrodomésticos, redes de transmissão de energia elétrica, pontos de pagamento com cartão, câmeras, portas eletrônicas, alarmes residenciais, e em muitos outros equipamentos do cotidiano.

Com a produção de dispositivos com tarefas específicas, surgem novas aplicações que não eram viáveis até então. Juntamente com as aplicações específicas, há a necessidade de *softwares* especializados que exigem algoritmos de maior complexidade. Parte desta área de pesquisa envolve a portabilidade de programas para dispositivos de baixo processamento e baixo consumo energético, seja para o avanço tecnológico (como em redes *Smart Grid* ou sensores sem fio) quanto para a solução de problemas já existentes.

1.1. Justificativa para o desenvolvimento do trabalho

Este trabalho analisa o caso da ofensividade [1] em redes IEEE 802.11 [2], problema este que está presente de modo intrínseco nas redes Wi-Fi, muito populares em todo o mundo. Há estudos, como os presentes em [3] e [4], que apresentam diferentes maneiras de se abordar e mitigar o problema, destacando-se entre eles um método de Lógica Fuzzy baseado em conjuntos matriciais, denominado Método Matricial [5].

Apesar de o Método Matricial ser utilizado originalmente para gestão de TI (Tecnologia da Informação) ele não foi implementado e testado em um ambiente real com dispositivos embarcados e de baixo poder de processamento. Este trabalho se justifica pela expansão do conceito de uso do Método Matricial para uso em redes de sensores, Internet das Coisas (IoT), entre outros, como forma de processamento de dados e gerência do sistema através da tomada de decisão local e autônoma.

Com a possibilidade de uso em diversas aplicações, as implementações do Método Matricial criadas neste trabalho foram realizadas para o estudo de caso específico da ofensividade em redes IEEE 802.11, já citado anteriormente. O problema da ofensividade é real e afeta uma das redes sem fio mais populares, o Wi-Fi. Deste modo, este trabalho busca aplicar o Método Matricial no auxílio à tomada de decisão na mitigação do problema da ofensividade em redes IEEE 802.11.

1.2. Objetivo do trabalho

O objetivo deste trabalho é a implementação do Método Matricial em dois tipos distintos de dispositivos embarcados. O primeiro deles é o Arduino [6], que representa o uso do algoritmo em produtos simples, de baixo poder de processamento e com baixa disponibilidade de energia. O segundo dispositivo é um roteador sem fio com suporte ao Linux embarcado [7], contendo um processador diferente do Arduino, além de maior disponibilidade de energia.

Este estudo busca entregar ao dispositivo embarcado o poder de tomada de decisão para gerência do sistema no qual ele está envolvido a partir da execução do Método Matricial. Em ambos os casos a aplicação de destino é a mitigação de ofensores em redes IEEE 802.11.

1.3. Resultados esperados

Espera-se, inicialmente, que a implementação do Método Matricial seja possível tanto para o Arduino quanto para o Linux embarcado. Ambos os *softwares* precisam realizar corretamente os cálculos e interagir com as matrizes de modo que os dados permaneçam concisos durante a execução do algoritmo.

Uma vez que as implementações estiverem completas, espera-se que, mesmo havendo divergência nos resultados numéricos finais (devido a possíveis diferenças no tratamento dado pelos processadores aos diferentes tipos de variáveis utilizados), a tomada de decisão não se altere. Por se tratar de um sistema Fuzzy pode haver diferença nos valores dos cálculos, contanto que o resultado linguístico final permaneça inalterado, como será explicado adiante.

Além disso, o consumo energético da execução do algoritmo deve ser compatível com dispositivos embarcados. De modo simplificado pode-se dizer que, em comparação com outras atividades desempenhadas pelo sistema o cálculo Fuzzy não deve se sobressair em termos de tempo de processamento. Especificamente no caso da ofensividade, espera-se que a tomada de decisão e o tempo de processamento sejam compatíveis com os requisitos do ambiente real.

Caso todos os requisitos sejam cumpridos, é de se esperar que a aplicação seja viável para ser utilizada em módulos comerciais. Espera-se que o tamanho do código compilado permita que seja realizada a instalação para uso em equipamentos de mercado, assim como a atualização remota de dispositivos em operação para a melhor absorção da solução pelos fabricantes.

1.4. Delimitação da pesquisa

Apesar da implementação do *software* estar preparada para ser portada para diversas arquiteturas, este trabalho se limitou em testar o Método Matricial no Arduino e no Linux Embarcado.

O estudo de caso da ofensividade em redes IEEE 802.11 foi realizado para testar as implementações com informações de um ambiente real. Além disso, os resultados apresentados podem levar ao uso deste trabalho para melhorar o desempenho de redes reais pela indústria, trazendo benefícios tanto para os fabricantes quando para os usuários finais.

Os resultados apresentados neste trabalho podem variar de acordo com a aplicação de destino. Como o Método Matricial envolve operações com matrizes, os cálculos serão afetados no caso de alteração nos valores de entrada e saída do sistema, ou nas suas características. Em certos casos, dependendo das matrizes do sistema, a aplicação do Método Matricial pode não ser viável, seja por alta necessidade de processamento, ou por baixa disponibilidade de energia. Além disso, aplicações muito específicas, ou com requisitos de alta taxa no fluxo de dados, podem não se beneficiar dos algoritmos criados. Para todos os casos de uso diferentes do utilizado neste trabalho, deverão ser realizados testes em todas as etapas do processo de criação do sistema Fuzzy para garantir a usabilidade do produto final.

1.5. Organização da dissertação

O Capítulo 1 apresenta uma introdução ao tema. Nele são apresentadas as justificativas para o desenvolvimento do trabalho, os seus objetivos, os resultados esperados, a delimitação da pesquisa realizada e a organização da dissertação.

O Capítulo 2 introduz o leitor às especificidades das redes IEEE 802.11, explicando o seu funcionamento e problemas existentes. Neste capítulo, é apresentada a questão da ofensividade, juntamente com o problema da ofensividade em redes IEEE 802.11, mostrando como ela age sobre as estações, as possíveis soluções existentes na literatura e como os ofensores podem ser identificados.

O Capítulo 3 introduz a teoria sobre os sistemas baseados em Lógica Nebulosa, discorrendo sobre a avaliação qualitativa a ser realizada, a calibração dos conjuntos e regras nebulosos, finalizando com a explicação de cada etapa do processo de cálculo. Também é apresentada a identificação de ofensores com lógica Fuzzy, o estudo de caso tratado neste trabalho, contendo uma breve explicação do seu ponto de partida, a sequência de análise de ofensores e a configuração do sistema nebuloso para a aplicação em questão.

O Capítulo 4 discorre sobre as aplicações embarcadas, mostrando o seu estado da arte, comentando sobre as plataformas Arduino, Scilab e OpenWRT, e terminando com a explicação do funcionamento do Método Matricial.

No Capítulo 5 está a metodologia, apresentando as atividades envolvidas na implementação do Método Matricial através do desenvolvimento dos *softwares* para o Scilab, Arduino e OpenWRT.

No Capítulo 6 estão apresentados os resultados obtidos, incluindo os testes de validação dos algoritmos, algumas informações importantes adquiridas, e os testes de desempenho dos sistemas embarcados.

O Capítulo 7 contém as conclusões e as perspectivas de trabalhos futuros.

2. REDES IEEE 802.11

O padrão IEEE 802.11 consiste em um documento elaborado pelo IEEE (*Institute of Electrical and Electronics Engineers*) para normatização das comunicações sem fio operando na faixa de frequência de 2.4GHz [2]. Esse padrão foi amplamente difundido com a popularização das Wi-Fi (marca registrada da Wi-Fi Alliance), uma implementação proprietária do padrão IEEE 802.11 que é utilizada em todo o mundo nos roteadores residenciais, de empresas, comércios e indústrias. Esse padrão se consolidou no mercado a partir do uso coletivo de computadores portáteis, porém sua aplicação foi alavancada globalmente com a popularização dos dispositivos de comunicação móveis, os *smartphones*, que hoje fazem parte do cotidiano de usuários em todo o mundo.

Neste capítulo, o padrão IEEE 802.11 será apresentado juntamente com exemplos práticos de sua utilização, contendo também a explicação da especificação técnica e seus principais pontos de interesse. Será discorrido sobre o mecanismo utilizado por este padrão para evitar colisões na transmissão de dados sem fio, o chamado CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), assim como seus problemas e possíveis soluções disponíveis no mercado e no meio acadêmico.

2.1. Comitê IEEE 802

Em fevereiro de 1980, foi criado o comitê IEEE 802 [8], cujo objetivo é criar e manter os padrões de comunicação e práticas recomendadas em redes sem fio locais (WLANs - *Wireless Local Area Networks*), metropolitanas (WMANs - *Wireless Metropolitan Area Networks*) e Ethernet, sendo posteriormente expandido para abranger outras áreas. A partir deste comitê foram formados subgrupos com foco em diferentes setores desta grande padronização das telecomunicações. É de interesse deste trabalho descrever brevemente as características do subgrupo 802.11 (*Wireless LAN Working Group*), responsável pela padronização das comunicações sem fio locais.

2.2. O padrão IEEE 802.11

A especificação do padrão IEEE 802.11 [2] é constituída por um extenso documento, contendo as características das camadas MAC (*Medium Access Control*) e PHY (*Physical Layer*) das WLANs. Esse documento foi utilizado como referência para as informações presentes neste tópico.

As especificações da camada PHY são, em sua maioria, características físicas do equipamento utilizado. Alguns exemplos são: a frequência de operação dos transceptores, a potência utilizada na transmissão, a sensibilidade mínima dos receptores, a taxa utilizada na comunicação, entre outros. Esses parâmetros são características do equipamento que indicam se ele está em conformidade com o padrão IEEE 802.11, garantindo assim a compatibilidade e interoperabilidade com outros equipamentos de diferentes fabricantes. A especificação da camada MAC contém os métodos a serem seguidos pelos equipamentos para acesso à camada física, como, por exemplo, os algoritmos criptográficos, formato dos quadros de dados e de gerência da rede, protocolos de comunicação, entre outros. Este trabalho foi construído com base em certas características desta camada, que possibilitam às estações compartilhar o meio físico sem concorrência direta das transmissões de RF (Rádio Frequência). Tais características e seu uso neste trabalho serão detalhados nos próximos tópicos.

As características das camadas PHY e MAC foram inicialmente criadas para estabelecer um padrão no mercado para implementações compatíveis, porém este padrão evoluiu com o passar do tempo. Sua evolução permitiu a inserção de novos recursos e modificação de recursos já existentes, sendo assim, existem várias versões do padrão IEEE 802.11, sendo comumente encontradas no mercado as versões 802.11a, b, g e n. As diferenças variam entre frequência de operação, taxa de comunicação, alcance máximo, banda ocupada, entre outros, muitas vezes objetivando a retrocompatibilidade entre versões e contando com a presença de mais de um modo no mesmo dispositivo.

2.2.1. Suporte à Multitaxa

O item 9.7 do padrão IEEE 802.11 (*Multi rate support*) [2] apresenta as diretrizes para padronização do suporte dos equipamentos à multitaxa. Esta

característica permite que o AP (*Access Point*) negocie taxas diversas para estações com características de comunicação diferentes, permitindo manter a QoS (*Quality of Service*) da comunicação mesmo que o nó esteja, por exemplo, a uma grande distância do AP. Esta característica precisa estar presente em todas as estações compatíveis com o padrão IEEE 802.11, embora este não especifique o algoritmo a ser implementado para o gerenciamento das taxas.

O padrão impõe, entretanto, uma lista de taxas mandatórias para interoperabilidade das estações, o que permite que equipamentos utilizem outras taxas além das especificadas, caso seja necessário. O suporte à variação de taxa é uma característica da camada PHY, a qual é transparente para as outras camadas após o estabelecimento da taxa pelo protocolo especificado.

A capacidade de alteração da taxa de comunicação durante a operação das estações é uma característica fundamental para o aumento da QoS de uma rede IEEE 802.11, uma vez que possibilita que estações em más condições de transmissão e recepção possam se comunicar com a base com sucesso. No dia a dia esta característica permite que equipamentos de *hardware* reduzido, como telefones celulares e *tablets*, possam se manter conectados ao ponto de acesso, mesmo possuindo antenas não tão eficientes quanto à de um *notebook*, por exemplo. Mesmo com a taxa reduzida o dispositivo é capaz de realizar as ações requisitadas pelo usuário, fornecendo a ele o serviço de conectividade esperado.

2.2.2. Métodos de acesso

A arquitetura da camada MAC do padrão IEEE 802.11 é composta por uma função de acesso ao meio usada como base para todo o protocolo, seguida de outras funções que podem ou não ser implementadas de acordo com a aplicação. O método de acesso padrão é o DCF (*Distributed Coordination Function*). Este método deve ser implementado em todos os dispositivos compatíveis com o padrão IEEE 802.11 e serve como base para o uso dos métodos opcionais, explicados na Figura 9-1 de [2].

Os métodos opcionais são utilizados majoritariamente em funções específicas de certas aplicações. O PCF (*Point Coordination Function*) é necessário em estações que requerem serviços de transmissão sem período de contenção e

não necessitam de QoS. O HCCA (*Hybrid Coordination Function Controlled Access*) é necessário em funções que precisam de QoS parametrizada, do mesmo modo que o EDCA (*Hybrid Coordination Function/Mesh Coordination Function Contention Access*) é utilizado em funções que necessitam de QoS priorizada. Por fim o MCCA (*Mesh Coordination Function Controlled Access*) é necessário em serviços de rede mesh.

2.2.2.1. Distributed Coordination Function

Para este trabalho, é importante o detalhamento deste método de acesso, usado como base para os métodos citados anteriormente. O DCF (*Distributed Coordination Function*) é fundamental para o padrão IEEE 802.11, pois é a base do método de acesso ao meio utilizado pelo sistema, devendo ser implementado por todos os equipamentos que necessitam de compatibilidade com redes deste tipo.

O algoritmo implementado no DCF do padrão IEEE 802.11 é o CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*). Este algoritmo será explicado em detalhes no item 2.2.2.2, porém vale ressaltar a sua principal característica na contenção de acesso ao meio: é mandatório que a estação aguarde certo período de tempo entre a transmissão de múltiplos pacotes consecutivos. Essa característica obriga as estações a liberarem o meio físico durante a transmissão de dados para que não haja o monopólio por uma estação, o que causaria uma queda no desempenho total da rede.

2.2.2.2. Carrier Sense Multiple Access with Collision Avoidance

O meio físico utilizado para a comunicação dos nós de uma rede IEEE 802.11 é o ar, portanto há algumas premissas implícitas ao uso deste meio de transmissão que podem ser enumeradas. É relevante para este projeto o fato de não haver a possibilidade de detecção de colisões na transmissão de dados simultânea por duas ou mais estações. Isso ocorre devido ao fato de que os módulos de comunicação sem fio podem apenas enviar ou receber dados em um dado momento, não sendo possível sentir o meio durante uma transmissão. Essa característica permite que duas estações transmitam dados simultaneamente, o que causa colisão das ondas de rádio e a consequente perda de informação por ambos os nós.

O CSMA/CA é um algoritmo que atua na camada de acesso ao meio (MAC) prevenindo colisões de dados através de um mecanismo de verificação do meio físico antes da transmissão. A sequência executada por um algoritmo CSMA/CA é a seguinte: caso uma estação deseje transmitir informações através da interface de RF esta deve, antes de iniciar o procedimento, aguardar um intervalo de *Backoff*. Passado este período, a estação deve "sentir" o meio físico em busca de transmissões de outros nós. Caso o meio esteja ocupado, ela deve aguardar por um intervalo de tempo denominado DIFS (*Distributed Inter Frame Space*) antes de reiniciar o processo, repetindo-o até que o meio esteja livre ou o número máximo de tentativas seja excedido. Em caso de repetição do processo, o tempo DIFS é adicionado de um intervalo de tempo aleatório para que a probabilidade de colisão diminua. Caso o destinatário tenha recebido o pacote corretamente, este envia uma mensagem de ACK (*Acknowledgement*) após aguardar um período de tempo denominado SIFS (*Short Inter Frame Space*). Seu funcionamento está ilustrado pelo diagrama de sequência da Figura 1, retirado de [9].

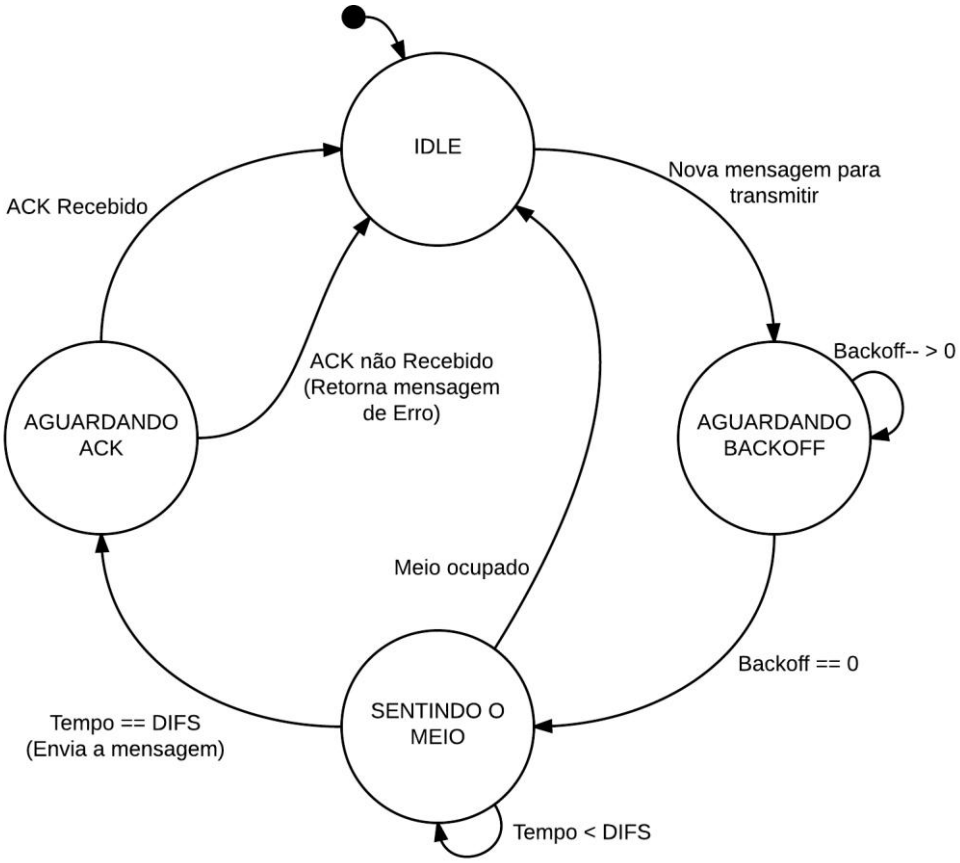

Figura 1 – Diagrama de sequência do CSMA/CA. Retirado de [9].

O principal objetivo do CSMA/CA em redes sem fio é garantir igual probabilidade de acesso das estações ao meio, quando considerado um longo intervalo de tempo. Esse método não permite que uma estação monopolize o canal com a transmissão constante de pacotes, o que degradaria a taxa de comunicação das outras estações, porém existem alguns problemas implícitos neste algoritmo cujas soluções não existem ou não foram implementadas comercialmente. Os principais são: o problema da estação escondida, o problema da estação exposta, anomalia da MAC e a ofensividade, sendo os dois últimos tratados por este trabalho.

2.3. Aplicações atuais

A principal aplicação atual das WLANs são as redes sem fio locais utilizadas em residências, empresas e no comércio para as mais diversas finalidades. Como exemplo, pode-se citar a crescente demanda por entretenimento (redes sociais, *stream* de vídeo e acesso à informação), a necessidade de integração de funcionários com sistemas (garçons realizando pedidos remotamente, funcionários auditando equipamentos em tempo real). Tais aplicações exigem uma rede de pequeno alcance, porém que permita o tráfego de taxas relativamente altas, sem prejudicar a conexão simultânea de múltiplos usuários.

Devido à sua popularidade e presença em muitos dispositivos utilizados no dia a dia, a área de cobertura proporcionada por pontos de acesso IEEE 802.11 vem aumentado substancialmente. Eles estão presentes em residências, restaurantes, empresas, *shoppings*, locais públicos, entre outros. Contudo, esta abordagem pode gerar inúmeros problemas a serem resolvidos antes de sua implementação no meio urbano, como problemas de segurança da informação e TI (Tecnologia da Informação), confidencialidade e anonimidade, gerenciamento das redes, assim como dificuldades comerciais, políticas, entre outras. Este trabalho se mantém restrito ao acesso de múltiplos usuários a um único ponto de acesso, o que será explicado em detalhes mais adiante.

2.4. Ofensividade

Como citado anteriormente, o padrão IEEE 802.11 especifica que a taxa de comunicação entre as estações deve ser manipulada de modo que permita a troca

de dados entre as estações mesmo que a potência de recepção esteja baixa. Ao diminuir a taxa de comunicação de uma estação ela, consequentemente, passará a ocupar o meio por um maior período de tempo. Isso ocorre devido às características físicas da comunicação, uma vez que os bits de dados serão sinalizados por um intervalo maior. Portanto, a transmissão de um mesmo pacote de dados levará mais tempo para ser concluída em uma estação com taxa reduzida do que em uma estação em boas condições de comunicação.

Para que a ofensividade seja caracterizada é necessário unir este fato à principal característica do CSMA/CA: a igualdade na probabilidade de acesso ao meio por todos os nós. Se observado sob um intervalo de tempo consideravelmente longo (tempo suficiente para trafegar muitos pacotes de todas as estações da rede) observa-se que cada estação trafegou uma quantidade de pacotes semelhante. Esta característica do CSMA/CA foi forjada para que uma estação não seja capaz de monopolizar o meio de comunicação, o que não permitiria que as outras estações se comunicassem.

Entretanto, ao ser utilizada em conjunto com o suporte à multitaxa, esta prática causa um efeito secundário indesejado na rede. As estações com taxa menor, apesar de trafegarem a mesma quantidade de pacotes, ocupam o meio por mais tempo do que as outras estações. Este efeito é chamado de Anomalia da MAC, e seu funcionamento e impacto na comunicação entre os componentes da rede serão explicados em detalhes a seguir.

2.5. Anomalia da MAC

O fenômeno da Anomalia da MAC foi apresentado à comunidade científica no ano de 2003 por meio da referência [1]. O trabalho pioneiro analisa o problema ao demonstrar que uma estação com taxa negociada baixa (em condições ruins de propagação do sinal) penaliza a comunicação das outras estações da rede. As estações em boas condições de comunicação e taxa negociada alta tem seu *throughput* (vazão total de dados) igual ou menor ao *throughput* da estação em condições ruins de comunicação, denominada estação ofensora [10]. A análise feita em [1] é teórica, sendo, posteriormente, comparada a medições de cenários reais,

provando e caracterizando a presença da Anomalia da MAC nas redes IEEE 802.11b.

Apesar da referência [1] tratar apenas de redes IEEE 802.11b, a anomalia da MAC também está presente nas outras versões do protocolo. De fato, este problema não está restrito somente às redes IEEE 802.11, mas pode estar presente em qualquer protocolo que utilize o CSMA/CA em conjunto com adaptação dinâmica da taxa de comunicação, como será explicado adiante. Este trabalho se restringe à continuidade dos estudos no protocolo 802.11b, abrindo espaço para trabalhos futuros com as outras versões.

A anomalia da MAC depende da presença de duas características arquiteturais do padrão IEEE 802.11. Como citado anteriormente, o CSMA/CA possui intervalos de tempo denominados DIFS e SIFS, os quais uma estação deve respeitar durante o acesso ao meio físico. Por serem múltiplos de um *slot* de tempo fixo, esses intervalos de tempo não se alteram com a mudança da taxa de comunicação. Este efeito faz com que uma estação com taxa negociada baixa ocupe o canal por tempo maior do que uma estação concorrente com taxa negociada alta, efeito ilustrado pela Figura 2.

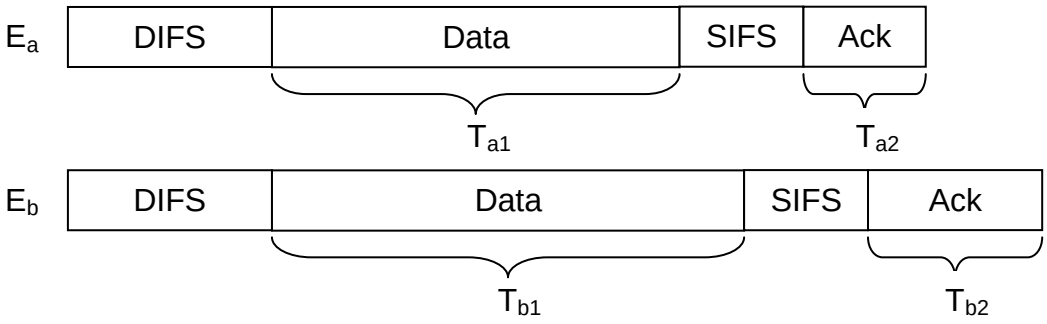

Figura 2 – Diagrama de temporização do CSMA/CA

Essa figura mostra a transmissão de um pacote pelas estações E_a e E_b, sendo que a primeira opera em uma taxa maior que a segunda. Sabendo-se que os intervalos de tempo de DIFS e SIFS são constantes em ambos os casos, observa-se que a estação E_a ocupa o meio físico por um intervalo de tempo representado por T_{Ea}, conforme a Equação (1).

$$T_{Ea} = DIFS + T_{a1} + SIFS + T_{a2} \quad (1)$$

Na qual T_{Ea} é o tempo total de transmissão de um pacote pela estação E_a, T_{a1} é o tempo necessário para que esta transmita os bits de dados, e T_{a2} é o tempo necessário para que o comando de Acknowledgement seja enviado.

Analogamente, a estação E_b ocupa o meio físico por um intervalo de tempo representado por T_{Eb}, conforme Equação (2).

$$T_{Eb} = DIFS + T_{b1} + SIFS + T_{b2} \quad (2)$$

Neste caso, como a taxa negociada da estação E_a é maior do que a taxa negociada da estação E_b, a relação entre os intervalos de tempo variáveis se torna:

$$\begin{matrix} T_{a1} < T_{b1} \\ e \\ T_{a2} < T_{b2} \end{matrix} \quad (3)$$

Portanto, o tempo de execução resultante de cada estação pode ser comparado de acordo com a Equação (4).

$$T_{Ea} < T_{Eb} \quad (4)$$

Estas equações ilustram o efeito da Anomalia da MAC, demonstrando que uma estação com taxa negociada baixa ocupa o meio por mais tempo do que uma estação com taxa negociada alta, porém sem alterar a probabilidade de acesso provida pelo uso CSMA/CA. No caso do padrão IEEE 802.11, essa diferença pode ser significativa, já que as taxas negociadas podem variar de 1MB/s a 11MB/s. Considerando-se o pior caso, a estação com taxa negociada baixa ocupa o meio por um tempo 11 vezes maior e possui a mesma probabilidade de acesso do que a estação com taxa negociada alta.

Finalmente, para que a Anomalia da MAC seja caracterizada, é necessário que o CSMA/CA atue sobre as estações, provendo a elas igual probabilidade de acesso ao meio físico. Essa característica, analisada em um longo período de tempo, mostrará que a quantidade de mensagens enviadas por cada uma das estações é similar. Observando que as duas estações enviaram uma quantidade semelhante de mensagens, é de se esperar que o tempo em que cada uma delas

ocupou o meio também seja semelhante. Entretanto, após um longo período de tempo, observa-se que a estação com taxa negociada baixa ocupa o meio por mais tempo, devido à presença da Anomalia da MAC.

2.6. Soluções disponíveis

Existem soluções para contornar o problema da Anomalia da MAC em redes IEEE 802.11, porém cada uma delas possui características próprias e diferenciadas. Entre elas, pode ser citada a solução proposta por [3], que utiliza a SNR (*Signal to Noise Relation* - Relação Sinal Ruído) para mitigar a anomalia. A abordagem propõe que a janela de contenção do CSMA/CA seja controlada através da análise da variação da SNR das estações, priorizando os nós com maior SNR e taxa de comunicação.

Como característica intrínseca a esse método, nota-se que o dispositivo deve ser modificado no baixo nível para que a solução seja aplicada, ou seja, é necessário alterar o firmware (*software* de baixo nível) responsável pelo protocolo de comunicação. Desse modo o *firmware* dos equipamentos deve ser alterado, o que leva à necessidade de troca dos equipamentos, atualização pela fábrica ou alteração do padrão de comunicação. A última opção seria a mais custosa, envolvendo o grupo de trabalho do IEEE 802 e afetando o mercado mundial, incluindo os dispositivos em operação.

Outro exemplo de solução para a Anomalia da MAC é a proposta por [4], na qual a medição da QoS através da taxa de bits da comunicação é utilizada para a parametrização da janela de contenção do CSMA/CA. Assim como o exemplo anterior, este necessita da alteração do *firmware* dos dispositivos para que a solução seja implementada e aplicada às redes reais.

Embora não seja especificado no trabalho citado, este exemplo pode ser classificado como uma aplicação de *Cross Layer*, uma vez que trabalha com parâmetros de diferentes camadas do protocolo de comunicação. O uso da janela de contenção em conjunto com a QoS pode ser classificada como *Cross Layer* por utilizar um parâmetro da camada MAC/PHY que depende de outro parâmetro da camada de aplicação da pilha OSI (*Open Systems Interconnection*) [11].

As aplicações citadas trabalham nas camadas baixas do protocolo, ou seja, nas camadas MAC e PHY, porém este trabalho utiliza uma abordagem alternativa, atuando na camada de aplicação. Esta abordagem permite que um *software* de alto nível adquira as informações da rede e atue sobre os nós ofensores, centralizando as atividades de gerência em uma aplicação no dispositivo. Com isso, é possível atualizar as redes atuais sem a necessidade de intervenção física, substituição ou compra de novos componentes, uma vez que a aplicação pode ser enviada remotamente.

2.7. Identificação de ofensores

Os trabalhos citados anteriormente visam mitigar a Anomalia da MAC através da análise da rede e ajuste de parâmetros, porém este trabalho segue por um rumo paralelo. Seu objetivo é a identificação dos ofensores, para que seja possível agir somente na estação problemática (ofensora), realizando as ações necessárias e punindo o ofensor, o que elimina ou diminui a anomalia presente na rede. Assim, o primeiro passo para a mitigação da anomalia de uma rede IEEE 802.11 é a identificação do ofensor.

A principal utilidade da identificação do ofensor é a construção de uma aplicação de gerência de redes para punição do ofensor. Este processo pode ser realizado de inúmeras maneiras como a eliminação do ofensor da rede, o bloqueio de futuras conexões de uma estação com Anomalia ou até o envio de informações para um centro de gerência para que sejam disparadas ações corretivas. Por poder estar presente em todas as redes IEEE 802.11, seria de grande valia para a qualidade de serviço das redes sem fio que a mitigação da Anomalia da MAC fosse implementada, uma vez que a expansão de dispositivos com este tipo de conectividade está cada vez maior e mais requisitada pelos usuários.

A aplicação de gerência de redes deve tomar as ações necessárias a partir das informações sobre o potencial de ofensividade de cada estação, fornecidas pelos estudos realizados neste trabalho, de acordo com aplicação e cenário no qual estão inseridas. Neste ponto, deve-se salientar a necessidade de uma ação de gerência antes que a anomalia cause maiores problemas à rede.

Em muitos casos, é possível que um especialista, com capacidade técnica adequada, analise os dados coletados de uma rede em operação e rapidamente encontre a estação ofensora. Entretanto há casos em que este método se torna inviável, como em aplicações que necessitam de cálculos em intervalos de tempo pequenos, ou em ambientes com muitas variáveis a serem comparadas e controladas.

Apesar da capacidade de dedução de informações subjetivas através dos dados analisados ser um processo trivial para o cérebro humano, o processamento de informações não analíticas é um processo complicado para um computador comum. Neste caso, devido às variáveis e comportamentos explicados adiante, a identificação de ofensores é um método matemático com elementos não exatos, ou seja, a aplicação de cálculos exatos não se aplica por não ser viável ou por não entregar os resultados esperados. Em casos como este, é feito uso da Lógica Nebulosa (ou Lógica *Fuzzy*), cujo objetivo é fornecer uma saída que represente a variável desejada, porém não sendo necessariamente um valor com significado físico.

Neste trabalho, o estudo de caso da Anomalia da MAC fará uso da Lógica Nebulosa para processar as informações de gerência, coletadas de redes IEEE 802.11, gerando como saída o potencial ofensivo de cada uma das estações. As informações de entrada serão a obsolescência do equipamento, a potência do sinal recebido, a taxa nominal negociada e a taxa efetivamente praticada.

3. SISTEMAS BASEADOS EM LÓGICA NEBULOSA

É comum na literatura a análise de problemas lógicos a partir de modelagem matemática, como em [3] e [4]. De fato, há muitos casos nas diversas áreas do conhecimento para aplicação dessa técnica, porém há casos em que as variáveis envolvidas não podem ser expressas por uma equação ou cálculo exato. Esses valores geralmente são linguísticos ou não lineares, necessitando de tratamento especial para seu processamento. Essas variáveis podem ser analisadas através da Lógica Nebulosa, cujo objetivo principal é lidar com valores numéricos ou variáveis linguísticas.

A teoria fundamental da Lógica Nebulosa, os Conjuntos Nebulosos (*Fuzzy Sets*), foi introduzida na literatura pelo matemático Lotfali Askar-Zadeh em 1965 [12]. A principal característica dos conjuntos nebulosos é a capacidade de reproduzir, de modo mensurável pela matemática, variáveis linguísticas tratadas pelo homem, possibilitando também a execução de operações de conjuntos (como união e intersecção) sobre os valores. O trabalho pioneiro citado foi seguido por muitos outros, ampliando, assim, o conceito e as aplicações da Lógica Nebulosa para as mais diversas áreas.

Na Engenharia, a referência [13] se destaca pela análise teórica das características da Lógica Nebulosa aplicada à área das Ciências Exatas. Nela, o autor descreve a Lógica Nebulosa como sendo um mapeamento de dados não lineares em um valor de saída escalar, utilizando regras para analisar simultaneamente valores de entrada numéricos e linguísticos. Este trabalho foi utilizado como base para muitas das explicações deste capítulo.

Conforme mencionado anteriormente, a Lógica Nebulosa é usada em diversas situações e nos mais diversos campos de atuação. Como exemplificado em [13], pode ser citado o uso de Lógica Nebulosa para sistemas de controle (aviões, trens e automóveis), sistemas aeroespaciais, análise do mercado de ações, estabilização de imagens e até gerenciamento de elevadores. Também pode-se citar as referências [14] e [15], que fazem uso da Lógica Nebulosa para aplicações de economia e gestão de redes de telecomunicações.

Este capítulo apresenta a teoria de Lógica Nebulosa aplicada à Engenharia. Para utilizar a Lógica Nebulosa na resolução de problemas dessa área do conhecimento há a necessidade de se trabalhar com variáveis de entrada e saída numéricas, ao invés de linguísticas. Portanto, as etapas iniciais e finais do sistema nebuloso, denominado FLS (Fuzzy Logic System), devem ser capazes de realizar as transformações necessárias nos dados de entrada e saída do sistema, respectivamente.

3.1. Avaliação qualitativa e calibração

O princípio básico dos Sistemas Nebulosos é a transformação de valores de entrada não lineares em uma saída numérica, que pode ser analisada e comparada com outras saídas geradas pelo mesmo sistema. A avaliação realizada por este processo não é analítica, e sim qualitativa, baseada em conjuntos nebulosos e regras previamente definidos. Por possuir tais características, os sistemas nebulosos devem passar por um processo de desenvolvimento e calibração específicos da aplicação em questão, pois cada cenário, ambiente ou grandezas analisadas são diferentes e possuem características próprias que precisam ser devidamente inseridas no sistema.

O procedimento de criação do Sistema Nebuloso será detalhado no tópico a seguir, porém é importante destacar que o seu desenvolvimento deve envolver um grande volume de dados experimentais, já que a criação dos conjuntos nebulosos e regras depende da análise não analítica do comportamento do sistema em estudo e de suas saídas esperadas. Os dados experimentais devem ser divididos em, pelo menos, dois conjuntos, os quais devem representar acontecimentos semelhantes no sistema em estudo. O primeiro conjunto de dados deverá ser utilizado exclusivamente para a criação e calibração do Sistema Nebuloso, enquanto o segundo conjunto deve ser utilizado para a verificação do seu funcionamento. Essa segregação é importante para a correta calibração do Sistema Nebuloso, uma vez que ele pode apresentar resultados concisos, porém não representar o comportamento real do sistema, caso os dados de calibração sejam utilizados para a verificação.

Os dados experimentais necessários para a criação de um sistema nebuloso são medições das grandezas de entrada e saída do sistema que se deseja estudar. Essas grandezas serão relacionadas pelo Sistema Nebuloso, o qual irá passar a gerar uma saída correspondente ao sistema real quando excitado pelas variáveis de entrada reais. Uma vez que há dados suficientes para a criação e calibração de um Sistema Nebuloso, há a ação de um especialista, definindo quais os conjuntos nebulosos relacionados às variáveis em estudo e qual a relação entre elas.

O especialista pode ser definido como uma pessoa que possui experiência com o sistema real que se deseja reproduzir com o Sistema Nebuloso. Ele é o responsável pela criação dos conjuntos nebulosos, regras e pela calibração do sistema, uma vez que possui conhecimento empírico e a experiência necessária para transferir os acontecimentos do sistema em estudo para o Sistema Nebuloso em construção. Por tratar os dados a partir de uma visão não analítica, a atuação de um especialista é de extrema importância para o sucesso do desenvolvimento de um Sistema Nebuloso funcional.

3.2. Sistema nebuloso

O professor Ebrahim Mamdani publicou, em 1975, uma aplicação para a teoria dos conjuntos nebulosos [16], inicialmente proposta por Zadeh [17], descrevendo um mecanismo nebuloso composto por quatro etapas capaz de realizar cálculos sobre variáveis de entrada, obtendo um valor de saída correspondente. A referência [16] contém a base para os estudos sobre Lógica Nebulosa e foi utilizada como referência em muitos dos trabalhos aqui citados, sendo a base dos cálculos nebulosos explicados neste capítulo.

Um Sistema Nebuloso pode ser dividido em subcomponentes com tarefas específicas na formação do sistema completo. Esses componentes possuem responsabilidades distintas, processando os dados de entrada, os conjuntos nebulosos, as regras e emitindo os dados de saída do algoritmo. Nesse item, cada bloco do Sistema Nebuloso será explicado em detalhes com base na Figura 3, a qual destaca e identifica cada etapa do processo.

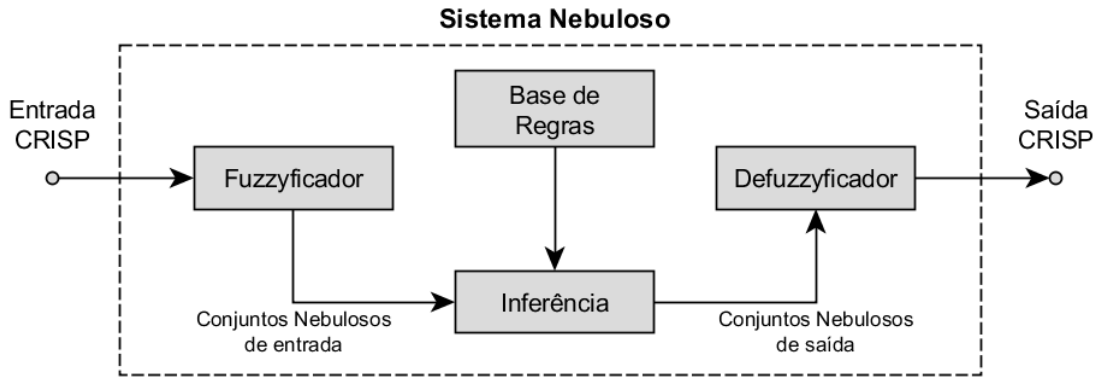

Figura 3 – Sistema Nebuloso (Figura baseada em [13]).

3.2.1. Valores de entrada

Os dados inseridos em um Sistema Nebuloso são chamados valores crisp. Essa nomenclatura se refere ao enquadramento dos valores em um dado conjunto, ou seja, eles podem pertencer ou não pertencer a um conjunto. Tal característica difere dos conjuntos nebulosos pelo fato de que estes podem adquirir pertinências em mais de um conjunto.

Na Engenharia, os valores de entrada dos Sistemas Nebulosos geralmente são números ou grandezas físicas adquiridas experimentalmente. Os valores *crisp* podem ser utilizados em cálculos matemáticos, já que são números reais e representam uma grandeza física. Esses são os dados que estão presentes no dia-a-dia de um engenheiro, podendo ser utilizados em equações matemáticas para a modelagem de um sistema. Como exemplo, pode ser citada a taxa de ocupação de um ambiente, que pode variar de 0 a 30 pessoas em uma sala de aula comum, ou de 0 a 200 pessoas em um auditório pequeno.

Entretanto, algumas aplicações lidam diretamente com valores de entrada do tipo *Fuzzy*. Esses dados são representados por variáveis linguísticas, associados a qualificativos linguísticos como "muito" ou "pouco", e são populares em aplicações nas quais os valores físicos isolados não possuem correlação direta com o fato que está sendo analisado. Como exemplo pode-se citar a sensação térmica da mesma sala de aula mencionada anteriormente, que pode estar quente, fria ou agradável, variando com a opinião da pessoa questionada.

3.2.2. Fuzzyficação

Para que um valor *crisp* possa ser utilizado em um Sistema Nebuloso é necessário que ele seja pré-processado. Esta etapa, chamada de Fuzzyficação, é constituída por duas etapas.

Primeiramente, é necessário representar a variável em questão utilizando conjuntos nebulosos. Cada conjunto é representado por uma função de pertinência descrita por valores modais. Esses conjuntos indicam a pertinência da variável *crisp* em todo o intervalo em estudo, podendo, assim, transformar o valor de entrada em uma variável linguística para que seja processada pelo Sistema Nebuloso.

As funções de pertinência podem assumir vários formatos, sendo mais comuns os formatos triangulares, trapezoidal, Gaussiano e linear por partes [13]. Neste estudo, serão utilizados os dois primeiros casos, uma vez que são aproximações válidas para muitas funções de maior complexidade. Além disso, este tipo de função necessita de menor processamento para geração dos resultados, como será demonstrado posteriormente neste trabalho. Estão exemplificados, na Figura 4, as funções de pertinência de formatos Triangular e Trapezoidal, juntamente com os valores modais (A, B, C e D) que as definem.

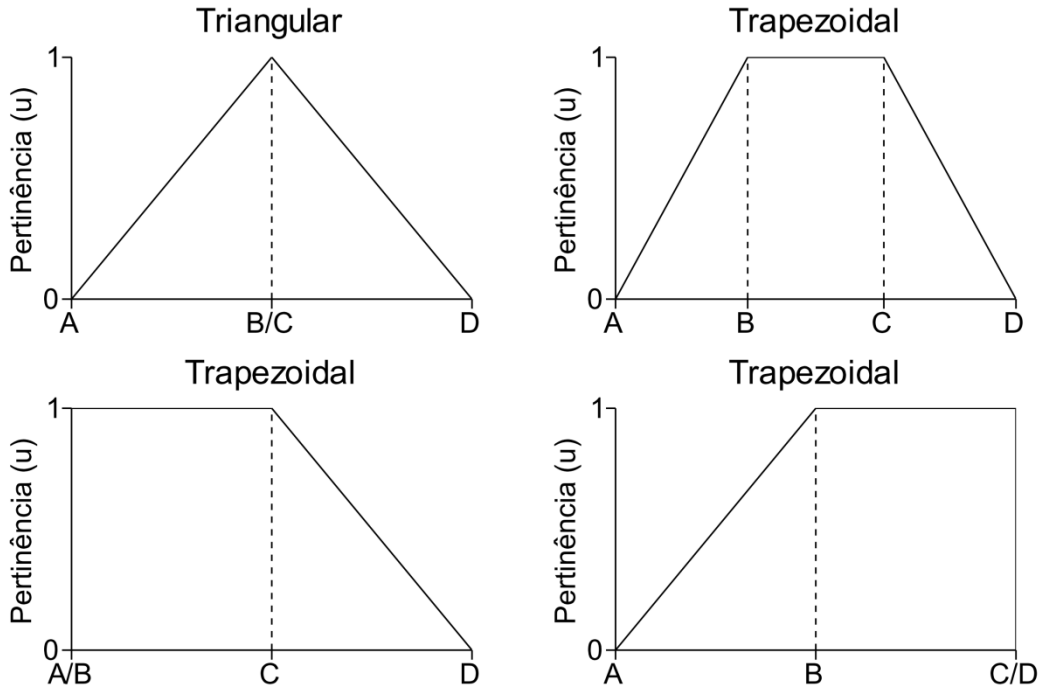

Figura 4 – Funções de pertinência Triangulares e Trapezoidais.

Na Figura 5, está ilustrado um exemplo de aplicação das funções de pertinência citadas, utilizadas para representar os conjuntos nebulosos ("Alta", "Média" e "Baixa"), associados a duas variáveis de entrada (originalmente valores *crisp*): temperatura e umidade de um dado ambiente. Esse exemplo foi criado de forma meramente ilustrativa, e será utilizado para auxiliar na explicação dessa e das próximas etapas do Sistema Nebuloso. Assim, os valores explícitos em seus conjuntos nebulosos não refletem, necessariamente, o comportamento de nenhum ambiente em específico.

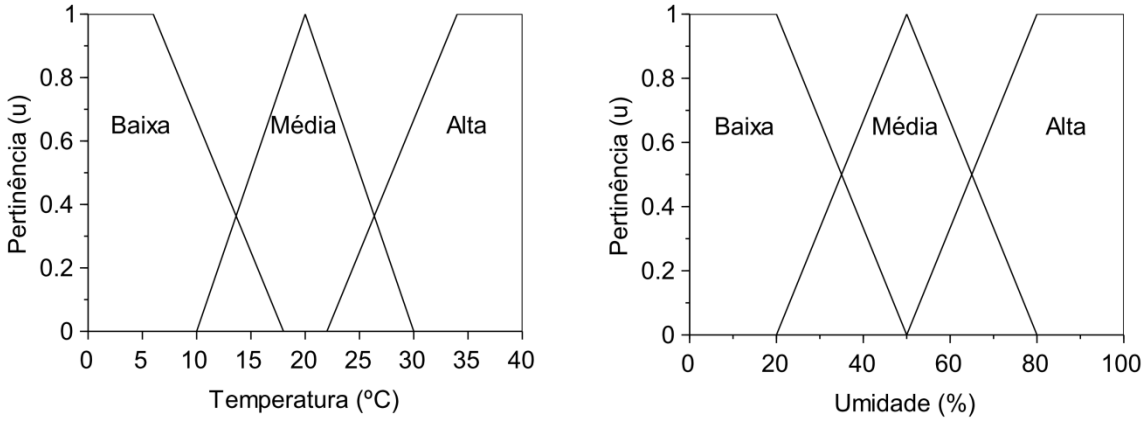

Figura 5 – Conjuntos nebulosos de exemplo

Uma vez definidos os conjuntos nebulosos associados a cada variável de entrada do Sistema Nebuloso, a segunda etapa do processo de Fuzzyficação refere-se ao cálculo da pertinência que o valor *crisp* de entrada representa em cada um desses conjuntos nebulosos. Se analisada graficamente, a pertinência pode ser adquirida simplesmente observando-se a correspondência entre os valores no gráfico. A Figura 6 mostra a pertinência dos valores de entrada aos conjuntos nebulosos do exemplo anterior, calculada de modo gráfico.

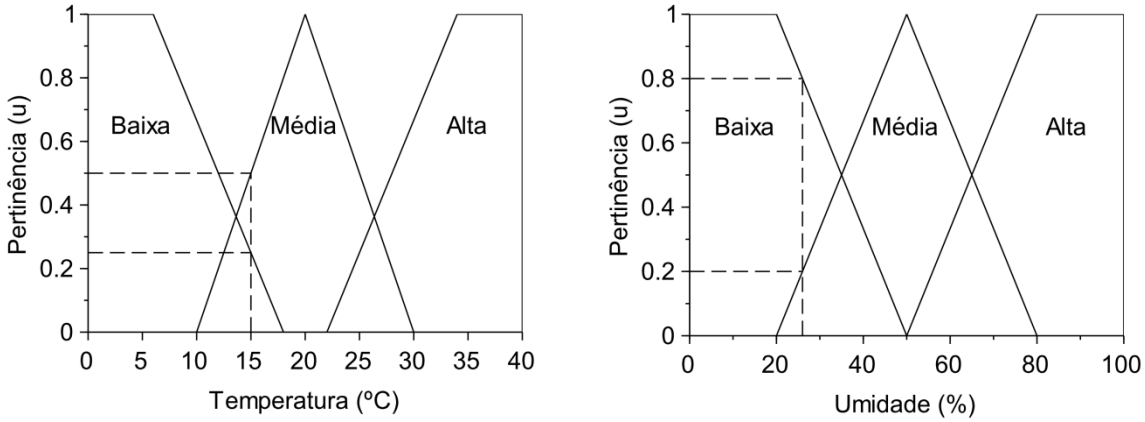

Figura 6 – Conjuntos nebulosos de exemplo com pertinências associadas aos valores de entrada.

Neste caso, a pertinência relativa à temperatura de 15ºC é de 0,25 no conjunto "Baixa", de 0,5 no conjunto "Média" e de zero no conjunto "Alta". A pertinência relativa à umidade de 26% é de 0,8 no conjunto "Baixa", de 0,2 no conjunto "Média" e de zero no conjunto "Alta". A pertinência relativa ao conjunto "Alta" é igual a zero tanto para a temperatura quanto para a umidade (para os valores de entrada analisados, ou seja, temperatura de 15ºC e umidade de 26%).

O cálculo matemático das pertinências, no caso das funções de pertinência triangulares e trapezoidais, é composto somente por valores constantes ou pertencentes a uma reta, portanto ele pode ser realizado com o auxilio da função definida por partes, conforme Equação (5).

$$u_{(crisp)} = \begin{cases} 0, & se\ crisp < a \\ crisp - a/(b-a)\,, & se\ a < crisp < b \\ 1, & se\ b < crisp < c \\ d - crisp/(d-c)\,, & se\ c < crisp < d \\ 0, & se\ crisp > d \end{cases} \quad (5)$$

Sendo '$u_{(crisp)}$' o valor da pertinência correspondente ao conjunto nebuloso em função da variável de entrada; sendo 'a', 'b', 'c' e 'd' os valores modais da função de pertinência (relativos à Figura 4), e sendo 'crisp' o valor da variável de entrada do sistema. Assim, para um exemplo numérico, pode-se aplicar a essas equações os mesmos valores de entrada utilizados para o cálculo gráfico da Figura 6, resultando nas equações a seguir.

crisp = 15ºC
Conjunto = Baixa

$$u_{(15^{o}C)} = \frac{(d - crisp)}{(d - c)} = \frac{(18 - 15)}{(18 - 6)} = 0{,}25 \quad (6)$$

crisp = 15ºC
Conjunto = Média

$$u_{(15^{o}C)} = \frac{(crisp - a)}{(b - a)} = \frac{(15 - 10)}{(20 - 10)} = 0{,}5 \quad (7)$$

crisp = 15ºC
Conjunto = Alta

$$u_{(15^{o}C)} = 0 \quad (8)$$

crisp = 26%
Conjunto = Baixa

$$u_{(26\%)} = \frac{(d - crisp)}{(d - c)} = \frac{(50 - 26)}{(50 - 20)} = 0{,}8 \quad (9)$$

crisp = 26%
Conjunto = Média

$$u_{(26\%)} = \frac{(crisp - a)}{(b - a)} = \frac{(26 - 20)}{(50 - 20)} = 0{,}2 \quad (10)$$

crisp = 26%
Conjunto = Alta

$$u_{(26\%)} = 0 \quad (11)$$

Por se tratarem de equações de primeiro grau, elas podem ser utilizadas em dispositivos de baixo processamento e com restrições de consumo de energia sem que as características do sistema sejam prejudicadas. Esta vantagem é um dos principais pontos abordados pelos estudos deste trabalho.

Com a transformação das variáveis de entrada *crisp* em variáveis linguísticas fuzzyficadas, é possível seguir para o próximo passo da execução de um Sistema Nebuloso: a inferência. Entretanto, antes que esta etapa possa ser demonstrada é necessária a exposição do conceito de Base de Regras em Lógica Nebulosa.

3.2.3. Base de regras

Nos Sistemas Nebulosos, a Base de Regras é composta por regras que, neste trabalho, possuem formato fixo:

$$\begin{gathered}\boldsymbol{SE}\\(V_{E1}\text{ é }X)\\\boldsymbol{E}\\(V_{E2}\text{ é }Y)\\\boldsymbol{ENTÃO}\\(V_S\text{ é }Z)\end{gathered} \qquad (12)$$

Na qual, V_{E1} e V_{E2} são variáveis linguísticas de entrada do sistema, chamadas "antecedentes"; V_S é uma das variáveis de saída do sistema, chamada "consequente"; X e Y são qualificativos linguísticos relativos às variáveis de entrada do sistema; Z é o qualificativo linguístico relativo a uma das variáveis de saída do sistema.

A seguir, são exemplificadas duas possíveis regras pertencentes a um sistema no qual os valores de entrada são a temperatura e umidade do ar, e a saída é o conforto térmico:

$$\begin{gathered}\boldsymbol{SE}\\(Temperatura\text{ é }Alta)\\\boldsymbol{E}\\(Umidade\text{ é }Baixa)\\\boldsymbol{ENTÃO}\\(Conforto\ Térmico\text{ é }Baixo)\end{gathered} \qquad (13)$$

$$\begin{gathered}\boldsymbol{SE}\\(Temperatura\text{ é }Média)\\\boldsymbol{E}\\(Umidade\text{ é }Média)\\\boldsymbol{ENTÃO}\\(Conforto\ Térmico\text{ é }Bom)\end{gathered} \qquad (14)$$

Durante a execução do Sistema Nebuloso todas as regras serão aplicadas, ou seja, os dados de entrada serão inseridos em todas as regras, porém nem todas serão implicadas. Observa-se que a regra presente na equação (13) seria aplicada, porém não seria implicada. Já a regra presente na equação (14) seria aplicada e implicada. Vide equações (6) a (11).

Uma particularidade do sistema de regras utilizado neste trabalho é que, mesmo utilizando o operador "E" para descrição da base de regras há o suporte intrínseco para o uso do operador "OU" nos antecedentes. A equação (15) ilustra uma regra com o uso do operador "OU".

$$\begin{gathered}\boldsymbol{SE} \\ (Temperatura \text{ é } Média) \\ \boldsymbol{E} \\ (Umidade \text{ é } Média)\ OU\ (Umidade \text{ é } Alta) \\ \boldsymbol{ENTÃO} \\ (Conforto\ Térmico \text{ é } Bom)\end{gathered} \tag{15}$$

Este fato ocorre pois todas as regras são aplicadas em conjunto na etapa de inferência, fazendo com que duas regras separadas causem o efeito do operador "OU". A equação (16) mostra as duas regras que, em conjunto, resultam na operação lógica da equação (15).

$$\begin{gathered}\boldsymbol{SE} \\ (Temperatura \text{ é } Média) \\ \boldsymbol{E} \\ (Umidade \text{ é } Média) \\ \boldsymbol{ENTÃO} \\ (Conforto\ Térmico \text{ é } Bom)\end{gathered} \qquad \begin{gathered}\boldsymbol{SE} \\ (Temperatura \text{ é } Média) \\ \boldsymbol{E} \\ (Umidade \text{ é } Alta) \\ \boldsymbol{ENTÃO} \\ (Conforto\ Térmico \text{ é } Bom)\end{gathered} \tag{16}$$

O conjunto de regras (Base de Regras) de um Sistema Nebuloso é construído a partir de análises experimentais, usualmente com base na experiência de um especialista, da variação da saída em função das entradas e deve cobrir todos os casos relevantes. É importante frisar que, como mencionado em 3.1, as regras devem ser calibradas por um especialista, garantindo que o sistema irá representar com precisão o comportamento em estudo.

3.2.4. Inferência

Esta etapa consiste na aplicação das regras sobre as pertinências calculadas na etapa de Fuzzyficação. Existem vários métodos para se realizar esta tarefa, sendo importante para as aplicações de Engenharia o método da Média [18] e o

método do Mínimo-Máximo [16]. Neste trabalho foi utilizado o segundo método, uma vez que é robusto, intuitivo e amplamente aceito pela academia.

O método do Mínimo-Máximo será aplicado a todas as regras do sistema através de dois passos. Primeiramente as pertinências dos conjuntos relativos aos antecedentes são comparadas, adquirindo-se o mínimo valor entre elas. O segundo passo é a aquisição do valor máximo entre a pertinência resultante do passo anterior e a pertinência atual do conjunto indicado pelo consequente. Caso o valor atual da pertinência do consequente seja substituído, a regra é considerada implicada. Como exemplo, consideram-se os valores de entrada "Temperatura Alta" e "Umidade Baixa", aplicados à Equação (13). Neste caso, a regra seria implicada, portanto seria calculada a pertinência mínima entre as pertinências relativas aos valores de entrada, seguido pelo cálculo da pertinência máxima entre o valor calculado e a pertinência atual do conjunto de saída.

Após a execução da inferência em um Sistema Nebuloso, as pertinências das variáveis de saída do sistema estarão calculadas, uma vez que o procedimento é realizado para todos os valores modais de saída do sistema.

3.2.5. Defuzzyficação

Esta é a última etapa do Sistema Nebuloso, antes de se obter o valor final esperado. O seu objetivo é transformar os valores linguísticos calculados através do processo nebuloso de volta em um valor *crisp*, para que possa gerar resultados tangíveis ao mundo da Engenharia. Como as outras etapas do Sistema Nebuloso, esta pode ser realizada de inúmeras maneiras dependendo da aplicação à qual os cálculos são destinados. Alguns exemplos são o método do Centróide, o método do Máximo, o método da Média dos Máximos, entre outros [13].

Neste trabalho, foi utilizado o método do Centróide, também chamado de Centro de Gravidade ou Centro de Área. Esta opção foi feita devido, principalmente, ao seu significado físico para a Engenharia, uma vez que resulta de uma variação da média ponderada das pertinências de saída do sistema. Além disso, o seu resultado pode ser obtido através da integração (ou seja, uma sequência de pequenas somas) das pertinências de saída, um tipo de cálculo benéfico para sistemas de baixo processamento. A Equação (17) mostra o cálculo do Centróide.

$$crisp_{final} = \frac{\sum_{i=0}^{n}\left(Xn(i) * un(i)\right)}{\sum_{i=0}^{n}\left(un(i)\right)} \quad (17)$$

Onde n é o número de pontos definido na Defuzzyficação, Xn é o valor do eixo X no ponto atual, e "un" é a pertinência de saída relativa ao mesmo ponto.

A Figura 7 ilustra a aplicação do método do Centróide no conjunto de saída do exemplo deste capítulo.

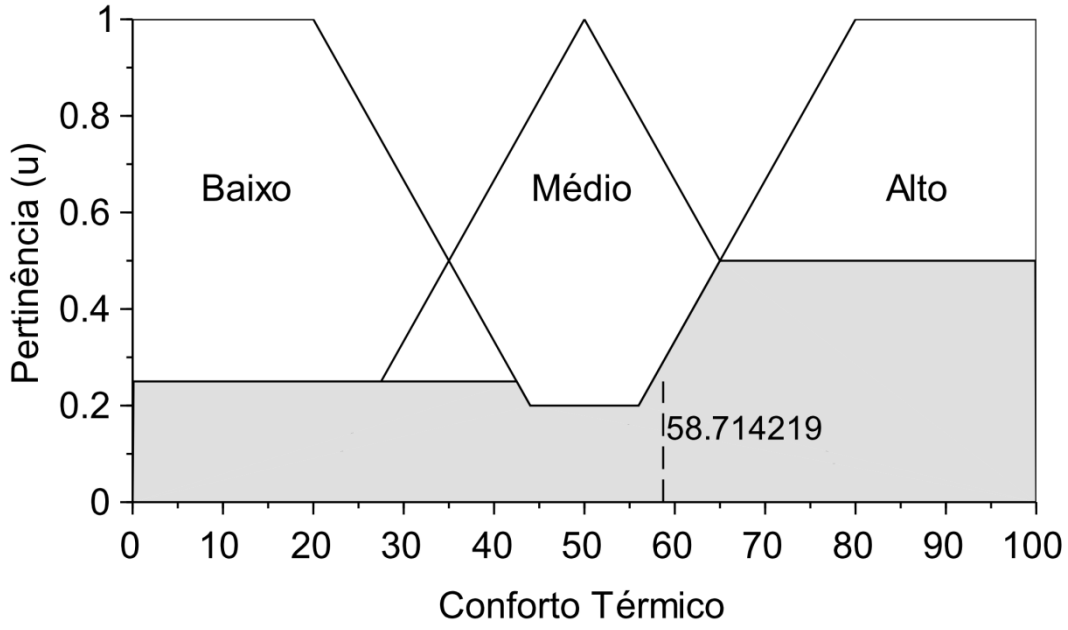

Figura 7 – Conjunto nebuloso de saída com o centróide calculado.

Pode-se observar na figura que as pertinências associadas a cada conjunto de saída do sistema foram associadas para formar a área hachurada. O centróide calculado sobre esta área está representado pela linha tracejada, resultando no valor explicitado. Observa-se que o valor final do centróide está deslocado para a direita em relação ao centro da escala. Isso é explicado pelo fato de que a pertinência relativa ao conjunto "Quente" é maior do que a pertinência relativa ao conjunto "Fria", deslocando o centro de massa da área em questão para o lado direito da figura.

3.2.6. Valores de Saída

Ao final da aplicação das etapas mencionadas neste capítulo, é obtido um valor de saída *crisp*. Este valor geralmente não possui significado físico, uma vez que é obtido através da análise de variáveis linguísticas, que também não podem ser explícitas pela matemática pura. Entretanto, seu significado é relativo aos dados da aplicação à qual ele está inserido.

Como explicado anteriormente, os valores de saída do sistema deverão ser previamente divididos em conjuntos nebulosos, aos quais serão atribuídas pertinências de saída. Ao realizar a divisão nestes conjuntos, automaticamente são estabelecidos os limites mínimo e máximo para a variação do valor *crisp* de saída, ou seja, os possíveis valores que ele pode assumir no final da aplicação do Sistema Nebuloso.

Caso o valor de saída não possua significado físico, ele pode ser utilizado para comparação entre outros resultados calculados a partir do mesmo Sistema Nebuloso, uma vez que eles são relacionados através do comportamento das variáveis linguísticas e regras que possuem em comum. Do mesmo modo, caso o valor de saída possua significado físico este poderá representar uma grandeza física mensurável do ambiente em estudo, lembrando-se sempre que a variável de saída possui limites mínimo e máximo, o que pode não ser verdade no ambiente real.

3.3. Identificação de ofensores com sistemas nebulosos

Conforme citado anteriormente, a identificação de ofensores em redes IEEE 802.11 não é uma tarefa simples. Para que seja possível analisar as estações de uma determinada rede e indicar os possíveis ofensores, é necessário um estudo aprofundado das características dos nós, de seu comportamento, e até mesmo das propriedades intrínsecas ao *hardware* presente nas estações.

Neste item serão analisados os pontos principais na identificação de ofensores em redes IEEE 802.11 utilizando Lógica Nebulosa. Serão explicados os conceitos que permeiam esta metodologia e o trabalho que foi utilizado como base para o desenvolvimento desta dissertação.

3.3.1. Ponto de partida

Este trabalho surgiu como continuação do tema tratado por [10], no qual é proposta a configuração de um Sistema Nebuloso para a atribuição de um Potencial Ofensivo a cada estação da rede. Os experimentos realizados em [10] mostram a criação de regras e conjuntos nebulosos que permitem a distinção entre estações comuns e ofensoras em redes IEEE 802.11b. Neste caso, as saídas são válidas para redes com duas estações e um AP.

Os testes foram validados a partir do pós-processamento de dados coletados de uma rede real, analisando-se as medições realizadas através de *software* de alto nível sendo executado em computadores cujo poder de processamento e consumo de energia não são itens limitantes. O resultado gerado foi um conjunto de valores que indicam o potencial ofensivo de cada estação, podendo esses ser comparados entre si para indicar qual das estações possui maior chance de ser o ofensor.

3.3.2. Sequência de execução da análise de ofensores

Uma estação ofensora pode ser definida como aquela que prejudica as outras estações da rede, seja por seu comportamento físico ou lógico. Neste trabalho, a definição de ofensividade se refere à capacidade de uma estação em prejudicar a comunicação das outras estações da rede através da ocupação demasiada do meio físico, levando-se em consideração que todas as estações possuem a mesma quantidade de dados para trafegar.

Quando a Anomalia da MAC está instalada em uma rede IEEE 802.11b, o comportamento das estações se torna similar. Durante este processo a taxa de comunicação de todas as estações é limitada e a taxa de comunicação da rede é degradada, o que pode ser causado por vários motivos, entre eles a qualidade da potência de sinal, a quantidade de dados a serem trafegados, a qualidade e robustez do *hardware* específico de cada equipamento e a carga do processador da estação. Por esse motivo, não basta que o sistema de gerência analise as características de cada estação separadamente para identificar o ofensor. Para que o método de identificação funcione corretamente, é necessário analisar e comparar as características de todas as estações da rede.

Após a identificação do ofensor, é necessário tomar alguma atitude do ponto de vista de gerência. Essa ação pode variar desde a simples desconexão do dispositivo até atitudes mais enérgicas, como a inclusão do identificador específico da estação (como o endereço MAC) em uma lista para que seja bloqueada qualquer tentativa futura de acesso pelo dispositivo.

3.3.3. Configuração do Sistema Nebuloso

Como explicado anteriormente, para que os Sistemas Nebulosos funcionem corretamente, esses devem ser previamente configurados e calibrados por um especialista. Para este trabalho, foram utilizados os Conjuntos Nebulosos e regras fornecidas pelo artigo [19]. Esse artigo apresenta a aplicação da Lógica Nebulosa em uma rede IEEE 802.11 para identificação de ofensores, demonstrando os testes realizados, os dados de entrada do sistema, as saídas esperadas, os Conjuntos Nebulosos e as regras utilizadas para redes com a presença de um ofensor. Tais dados foram adquiridos experimentalmente pelos autores através de uma bancada de testes funcional contendo duas estações e um AP. Esta rede permitiu que fossem coletados dados suficientes para a configuração do Sistema Nebuloso, além da realização de testes sobre o sistema finalizado, provando o conceito estudado.

Apesar das matrizes utilizadas terem sido criadas para a presença de apenas um ofensor na rede, o produto deste trabalho pode ser utilizado com quaisquer outras matrizes, sendo necessário realizar um estudo futuro sobre redes com mais de um ofensor para que o método possa ser aplicado.

A implementação embarcada criada neste trabalho, descrita no Capítulo 4, utilizou as regras e conjuntos nebulosos apresentados no artigo citado, que estão reproduzidos no Apêndice 1, para exemplificar a aplicação do sistema. Os dados foram utilizados para testar a implementação do Sistema Nebuloso com dados reais, em um nó de rede real, como será explicado adiante.

4. APLICAÇÃO EMBARCADA

O desenvolvimento de *software* para dispositivos embarcados possui certas exigências extras, quando comparado a *softwares* desenvolvidos para computadores comuns. As diferenças entre eles variam entre a portabilidade dos equipamentos, segurança (de *hardware* e *software*), dissipação térmica, tamanho físico, interface com o usuário, confiabilidade, disponibilidade de serviços, segregação e proteção de memória, recuperação de falhas, maior possibilidade de ataques físicos, entre outros [20, 21, 22]. Dentre as características citadas, o principal ponto abordado nesta categoria é a necessidade de reduzir ao máximo o consumo energético geral do produto. Isso ocorre devido à natureza dos dispositivos embarcados em possuir *hardware* reduzido e, em muitos casos, ser alimentado com fontes de energia alternativas como baterias e células solares.

Além da escassez de energia, os dispositivos embarcados são cada vez mais requisitados em termos de processamento de dados. Alguns exemplos são o grande crescimento da automação de dispositivos para serviços ao usuário, como telefones celulares, casas inteligentes, centrais automotivas de entretenimento e informação (infotainment); há também a crescente demanda do mercado pela disseminação da Internet das Coisas (*Internet of Things* - IoT), através da qual os mais variados dispositivos do dia-a-dia serão interconectados [23]; também vale citar a aplicação de sistemas embarcados na gerência autônoma de redes, como em nós roteadores de rede e em concentradores *Smart Grid* [24].

Diante dessa necessidade tão variada exigida pelas aplicações, os sistemas embarcados devem ser capazes de realizar novas tarefas cada vez mais complexas. Essa evolução adicionou novos desafios para os desenvolvedores, que precisam criar novos meios de processar mais informações, porém se mantendo alinhados com as exigências explicadas anteriormente.

Foi dessa necessidade que surgiu o tema tratado por este trabalho, tendo por objetivo a implementação embarcada do método de processamento com Lógica Nebulosa, que é explicado neste capítulo.

4.1. Estado da Arte

O uso de processamento embarcado está cada vez maior devido ao constante avanço da tecnologia e o aumento do poder de processamento de dispositivos, que são cada vez menores. São exemplos cotidianos desse avanço os aparelhos telefônicos móveis, os *notebooks* e câmeras digitais. Mais especificamente, este trabalho se restringe à análise de dois tipos de dispositivos do mercado: os sensores de baixo consumo energético, através do Arduino [6], e os roteadores sem fio de pequeno porte, através do OpenWRT [25]. Cada um desses dispositivos possui características próprias que merecem destaque neste tópico. Primeiramente, serão tratados os roteadores sem fio, cujas características vêm sendo adaptadas ao longo do tempo para suprir as necessidades das redes, cada vez mais disseminadas.

Os rotadores sem fio são dispositivos cujo papel principal é realizar a conexão final de uma rede, ou seja, é através dele que a interface do usuário irá interagir com os outros componentes da rede. Este é o papel básico dos APs comerciais e residenciais, porém as funcionalidades aplicadas a estes dispositivos cresceram ao longo do tempo. Foram acrescentadas a eles principalmente funções de gerência de redes, como aplicações de log, firewall, encaminhamento de portas, gerência remota, controle dos pais, entre outros, gerando a necessidade de atualização do *hardware* e *software* empregados [11].

Atualmente, os roteadores sem fio fazem uso de um sistema operacional simplificado (geralmente uma distribuição proprietária do Linux) e são compostos por aplicações para cada um dos recursos citados anteriormente. Por esse motivo, e pelo fato de o dispositivo possuir alimentação externa constante, os roteadores sem fio não priorizam, necessariamente, o baixo consumo de energia. Desse modo, seu poder de processamento é maior que o dos sensores, podendo realizar tarefas de maior complexidade.

Já os sensores de baixo consumo são uma modalidade de dispositivos que vêm demarcando seu espaço no mercado nos últimos anos. Com a possibilidade real de implantação de redes *Smart Grid* e as pesquisas em IoT, o desenvolvimento

de dispositivos robustos, menores e com baixo consumo energético tornou-se um objetivo comum para muitas empresas em todo o mundo.

Os nós sensores deste tipo devem realizar as mais variadas tarefas, porém grande parte delas é realizada em um curto intervalo de tempo, seguido por um longo período de inatividade do dispositivo, que aguarda em um estado de baixo consumo de energia. Devido a essa característica os sensores devem coletar os dados, processa-los e gerar um resultado em um curto período de tempo, e como processador mais econômico possível. Os processadores de baixo consumo energético não são novidade na indústria, temos como exemplo a família MSP430 da Texas Instruments [26] e a tecnologia picoPower da Atmel [27], porém este tipo de processador será empregado em todo o mundo com a ascensão das novas tecnologias citadas anteriormente.

Juntamente com os processadores de baixo consumo, há a necessidade do desenvolvimento de aplicações que realizem as tarefas necessárias, porém mantendo o tempo e necessidade de processamento baixos. É neste ponto que está situado este trabalho, cujo principal fruto é uma aplicação de análise nebulosa para dispositivos embarcados, criada a partir desses conceitos.

4.2. Scilab

O Scilab [28] é uma linguagem de programação de alto nível voltada para análise matemática e prototipagem de algoritmos computacionais, sendo também utilizado na criação de modelos de referência para outros *softwares*, como neste trabalho. Além disso, o Scilab foi utilizado como ferramenta para a criação de todos os gráficos apresentados nas figuras deste documento.

Seu uso é frequente em Universidades e instituições de pesquisa por ser similar à linguagem Matlab [29]. A distribuição do Scilab é feita gratuitamente pelo fabricante, permitindo seu uso em ambientes operacionais Windows, Linux e Mac OS sem a necessidade de compra de licença ou direitos autorais.

4.3. Plataforma Arduino

O Arduino [6] é uma plataforma de código aberto para prototipagem e desenvolvimento de *hardware* e *software* de dispositivos embarcados. A

comunidade do Arduino é ativa, sendo composta por empresas, engenheiros e entusiastas dispostos a compartilhar conhecimento e construir as mais diversas aplicações.

As aplicações para Arduino são desenvolvidas em linguagem Wiring, que é baseada em C e C++. Por esse motivo, elas são muito semelhantes quanto às funções e construção do código fonte. A construção de *software* no Arduino pode ser baseada no uso de bibliotecas, para que a aplicação do usuário se concentre em utilizar as funcionalidades providas pelo código incluído. A construção dessas bibliotecas é independente e pode ser feita em linguagem C sem que a aplicação de usuário precise ser alterada.

Essa característica torna possível o reaproveitamento de código das bibliotecas desenvolvidas para Arduino para uso em aplicações comerciais. Por esse motivo, um dos *softwares* desenvolvidos por este trabalho será uma biblioteca Arduino para análise nebulosa, tópico que será explicado em detalhes posteriormente.

4.4. OpenWRT

Dentre as distribuições de Linux Embarcado há uma que se destaca no uso em roteadores de pequeno porte: o OpenWRT [25], que foi criado para que usuários pudessem personalizar ou modificar as funcionalidades dos seus roteadores, incluindo algoritmos, aplicativos ou até alterando parâmetros da camada física do protocolo de comunicação.

Por ser um *software* de código aberto, os usuários de sua ativa comunidade contribuem constantemente na manutenção da distribuição, além de auxiliar os desenvolvedores na inclusão de suporte a novos dispositivos de mercado. Este fato garante que ele seja suportado em novos produtos, permitindo que este trabalho tenha continuidade após a publicação deste documento.

4.5. Método Matricial

Em [5] é descrito um método para implementação de um Sistema Nebuloso através de cálculos com matrizes numéricas. De acordo com a autora, o método apresentado é generalista, já que as operações numéricas entre matrizes permitem

sua implementação tanto em *hardware* quanto em *software*. O método auxilia nos processos de tomada de decisão que envolvem a subjetividade, podendo ser aplicados a uma grande variedade de cenários.

Este método é composto por 3 matrizes, cada uma com uma funcionalidade específica. A Matriz C contém os valores dos conjuntos nebulosos utilizados, a Matriz E contém os valores de entrada do sistema, e a Matriz R é composta pelas regras nebulosas. Cada uma delas desempenha um papel específico no cálculo nebuloso, já que possuem, além das informações citadas, outros dados importantes.

A Matriz E é, geralmente, a menor e mais simples das três matrizes envolvidas, sendo constantemente manipulada pelo *software* para inserção de novos valores de entrada no sistema. Ela possui duas colunas, sendo que a primeira coluna contém os valores de entrada, e a segunda coluna indica quantos conjuntos nebulosos estão relacionados a cada um dos valores de entrada. Através da análise das informações contidas nesta matriz, o sistema é capaz de inferir a quais conjuntos nebulosos cada valor de entrada se refere, ou seja, quais linhas da Matriz C devem processar cada valor de entrada. A Matriz E possui tantas linhas quantos forem os valores de entrada no Sistema Nebuloso.

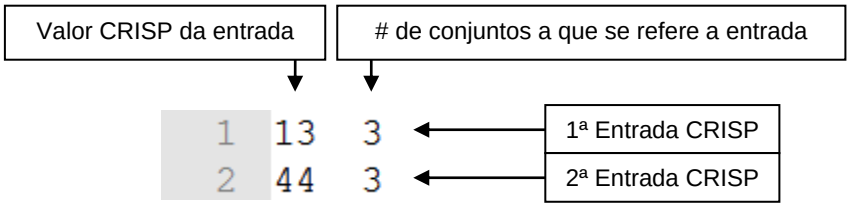

Figura 8 – Matriz E de exemplo.

Cada linha da Matriz C apresenta duas informações. Como explicado anteriormente, este trabalho se limitou ao estudo de conjuntos representados por funções de pertinência de formatos triangular ou trapezoidal, portanto cada conjunto nebuloso sempre será composto por quatro valores modais. Desse modo, a Matriz C é constituída por cinco colunas, nas quais os quatro primeiros valores representam os valores modais do conjunto nebuloso, e o quinto valor representa a pertinência calculada na etapa de Fuzzyficação. A Matriz C possui uma quantidade de linhas igual à soma dos valores presentes na segunda coluna da Matriz E adicionada da quantidade de conjuntos nebulosos de saída do sistema.

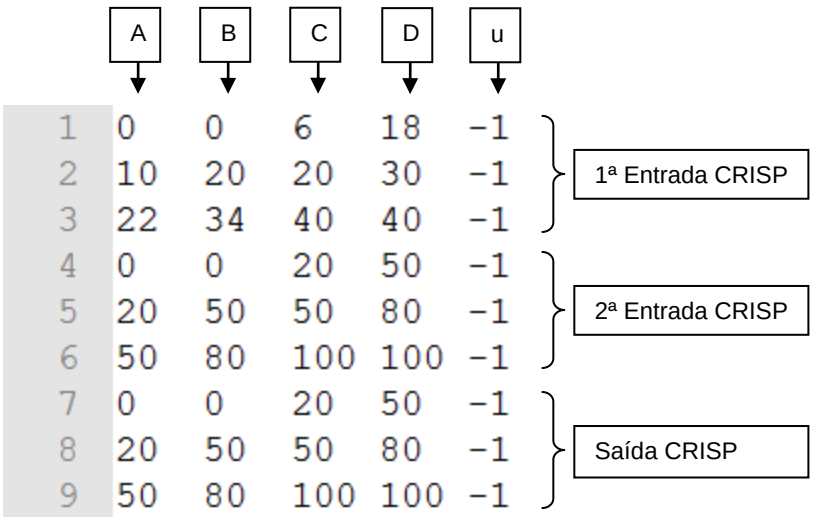

Figura 9 – Matriz C de exemplo.

Finalmente, a Matriz R contém as regras do Sistema Nebuloso. Como explicado anteriormente, elas possuem o formato "SE X E Y, ENTÃO Z", portanto, a Matriz R é formada por três colunas, indicando o número das linhas da Matriz C que devem ser analisadas, neste caso, correspondendo aos conjuntos nebulosos indicados por X, Y e Z. A Matriz R possui tantas linhas quanto forem necessárias para o correto funcionamento do Sistema Nebuloso em questão, sendo definidas na etapa de Calibração por Especialista.

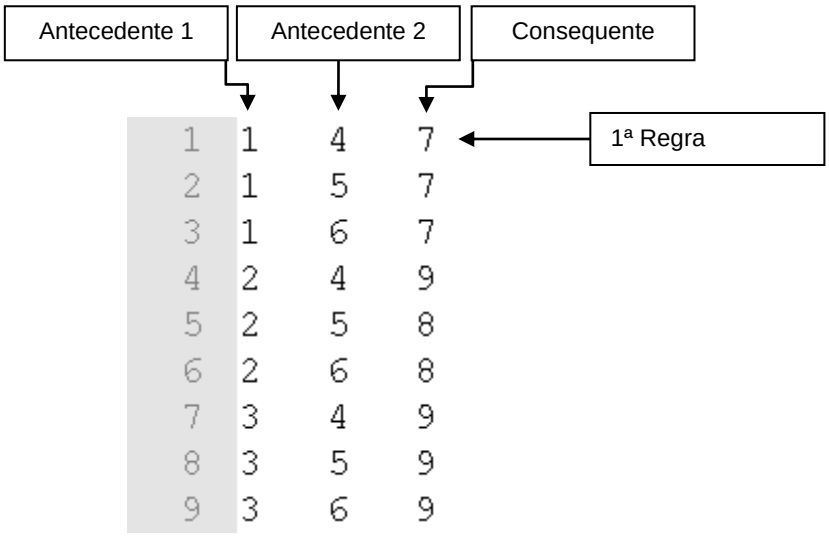

Figura 10 – Matriz R de exemplo.

Os cálculos do Método Matricial se baseiam no procedimento proposto por Mamdani [16], sendo constituído por uma arquitetura de cinco passos:

1. Construir as matrizes do mecanismo de inferência;
2. Realizar os cálculos sobre a Matriz C (Fuzzyficação);
3. Realizar a aplicação das regras sobre a Matriz C;
4. Definir as pertinências dos conjuntos de saída a partir da agregação de consequentes (Defuzzyficação);
5. Calcular os valores de saída do sistema (Defuzzyficação).

A arquitetura foi representada aqui de forma meramente informativa, visando o melhor entendimento da metodologia aplicada, devendo a referência original ser consultada caso sejam necessárias informações detalhadas sobre cada etapa do processo. Por isso, este item se limita a explicar o funcionamento e características do modelo matricial, ou seja, como se dá a construção e processamento das informações nebulosas através do Método Matricial.

A criação das matrizes do mecanismo de inferência nada mais é do que a definição das variáveis explicadas no item 3.2.1, no qual o sistema é analisado por um especialista, obtendo-se os valores dos conjuntos nebulosos, regras e limites de validade das entradas e saídas. A aplicação matricial da primeira etapa do processo envolve apenas a inserção dos dados nas matrizes C, E e R.

Em seguida, é realizada a Fuzzyficação das entradas. Para este passo a aplicação matricial traz o benefício de tratar os dados de entrada separadamente, ou seja, através da análise de cada linha da matriz C individualmente. Neste passo, deve-se analisar o valor de entrada sobre os quatro primeiros valores da linha correspondente, inserindo o valor resultante na última coluna da mesma linha.

Uma vez que todas as linhas da Matriz C referentes aos conjuntos nebulosos de entrada passaram pelo processo de Fuzzyficação, há aplicação das regras sobre eles. Mais uma vez, a aplicação das matrizes permite que cada linha da Matriz R seja tratada individualmente, portanto é realizada a comparação dos valores Fuzzyficados indicados pelos dois primeiros valores, armazenando o resultado na linha da Matriz C indicada pelo último valor da linha atual.

Finalmente, a agregação dos consequentes e Defuzzyficação dos valores de saída do sistema se tornam simples, uma vez que as linhas da Matriz C correspondentes aos conjuntos nebulosos de saída foram preenchidas pelo procedimento anterior. Desse modo, basta realizar o procedimento inverso, ou seja, a partir da agregação dos valores calculados na aplicação das regras é gerado um conjunto com a soma lógica dos mesmos, sendo aplicado sobre este conjunto o método do centróide para Defuzzyficação. O resultado deste procedimento é o valor de saída do sistema, indicando a grandeza ou relação de saída relativa aos dados de entrada.

O objetivo da sequência de cálculos explicada é o uso em sistemas computacionais embarcados. Este trabalho buscou implementar tal algoritmo em dispositivos reais, com base em aplicações da indústria.

5. METODOLOGIA

O desenvolvimento prático deste trabalho envolveu a implementação de três *softwares* para a resolução de problemas utilizando Lógica Nebulosa. Cada um dos três *softwares* foi desenvolvido em linguagens diferentes e foram idealizados para servir propósitos distintos.

O primeiro *software* (item 5.1) teve como objetivo provar o conceito do Método Matricial explicado no item 3.3 e detalhado na referência [5], sendo utilizado também como modelo de referência para a validação dos resultados dos outros *softwares*. O segundo algoritmo (item 5.2) foi implementado visando o uso em plataformas embarcadas de baixo consumo de energia e baixo poder de processamento, objetivando o uso em sensores e dispositivos miniaturizados. Já o terceiro *software* (item 5.3) é uma aplicação para a distribuição de Linux embarcado OpenWRT, podendo ser utilizada para gerência de redes em roteadores de mercado.

É importante destacar que, apesar das implementações dos *softwares* para Arduino e OpenWRT realizadas neste trabalho serem funcionais, seu uso em aplicações reais não é trivial. Para que os *softwares* criados sejam utilizados em trabalhos futuros é necessário o estudo e calibração em cada aplicação. Em especial, a utilização dos *softwares* no estudo de caso da ofensividade em redes IEEE 802.11 tratado neste trabalho não é trivial. Para isso, é necessária a criação de um *software* de gerência de redes capaz de invocar o *software* criado neste trabalho, processar as informações coletadas da rede e tomar uma ação de gerência sobre as estações ofensoras. Além disso, o procedimento deve ser testado e validado através de testes em redes reais.

Como os softwares precisaram do auxilio de um conjunto de matrizes para serem desenvolvidos, essas estão ilustradas neste capítulo e serão utilizadas nas explicações. Estes dados, chamados de Matrizes de Desenvolvimento, foram utilizados para a construção e validação inicial do funcionamento dos softwares, assim como na comparação dos resultados gerados por eles.

Na Figura 11, está ilustrado o fluxograma que representa a sequência de execução dos três *softwares* criados. Os passos indicados na Figura 11 correspondem aos tópicos presentes no Item 3.2. As duas exceções são as caixas de verificação da validade dos valores de entrada e da verificação de erro durante aplicação do método. A primeira verificação realiza uma análise simples dos dados de entrada, retornando erro caso o número de linhas e colunas das matrizes sejam diferentes do esperado. A segunda verificação retorna erro caso os valores contidos nas matrizes estejam incoerentes, ou seja, caso haja algum problema durante o processamento dos dados.

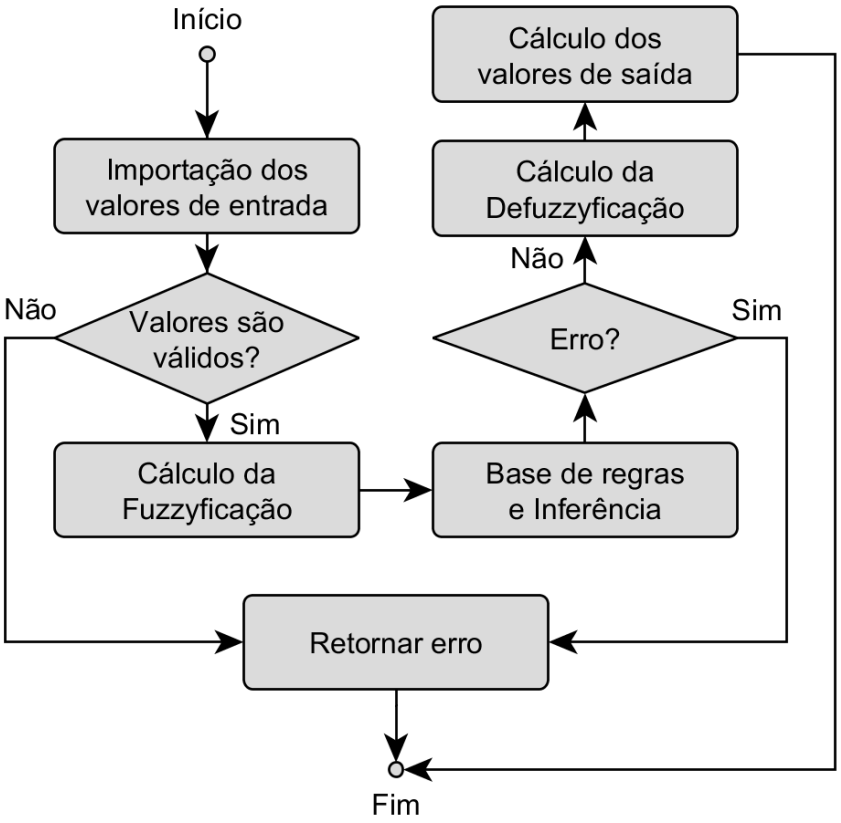

Figura 11 – Fluxograma de execução dos *softwares* implementados.

Todos os softwares criados neste trabalho, contendo os códigos fonte e bibliotecas, são apresentados no Apêndice 4, no Apêndice 5 e no Apêndice 6. As versões mais atuais dos códigos podem ser encontradas no repositório *web*, disponível em [30].

5.1. Desenvolvimento de *software* para teste de conceito do Método Matricial utilizando Scilab

O *software* implementado nesta etapa utiliza a linguagem de programação Scilab, sendo empregada a versão mais atual do programa disponível no momento do início das atividades, cuja distribuição é a 5.5.0. A implementação foi criada a partir das especificações do Método Matricial explicadas anteriormente e detalhadas na referência [5], e busca retratar o algoritmo de modo fiel e didático. O objetivo foi criar um *software* capaz de servir como prova de conceito para validação dos outros *softwares*, ao mesmo tempo em que provê um melhor entendimento do processo nebuloso através da geração de tabelas e gráficos informativos. Este *software* não prevê as características de implementação embarcada apresentados, uma vez que não será utilizado nestes dispositivos.

Antes de iniciar o desenvolvimento, foi necessário adquirir conhecimento sobre a ferramenta computacional e suas funções. Para isso, a melhor fonte de informações encontrada foi o próprio *website* do Scilab [28], já citado anteriormente, cuja página de ajuda fornece todos os detalhes necessários para o entendimento e desenvolvimento das aplicações.

5.1.1. Importação dos dados

Assim como os sistemas nebulosos, o desenvolvimento do *software* pode ser dividido em etapas. Primeiramente, foi necessário inserir os dados de entrada no Scilab. Para isso, o método escolhido foi a leitura e escrita de arquivos de texto, cujo conteúdo reflete as matrizes do sistema nebuloso. Desse modo, o programa necessita que três arquivos estejam presentes no diretório no qual ele será executado:

- matriz_E.txt
- matriz_C.txt
- matriz_R.txt

Esses arquivos são do tipo texto, e devem estar preenchidos apenas com valores numéricos, separados por um ou mais espaços. O caractere 'tab' também é válido. O caractere '.' deve ser utilizado para indicar as casas decimais.

O arquivo "matriz_E.txt" é o menor deles. Ele é composto por duas colunas, sendo que a primeira contém os valores de entrada do sistema, e a segunda indica quantos conjuntos nebulosos estão associados a cada um dos valores de entrada. A soma dos valores da segunda coluna deve ser igual ao número total de conjuntos nebulosos de entrada do sistema. A Figura 12 contém um exemplo para o conteúdo deste arquivo (a coluna mais escura indica o número da linha).

```
1  13  3
2  44  3
```

Figura 12 – Matriz E de desenvolvimento.

Pode-se observar que, neste exemplo, há uma grandeza de entrada com o valor 13 que se refere às 3 primeiras linhas da Matriz C, seguida de outra grandeza de valor 44 que se refere às próximas 3 linhas da Matriz C.

O arquivo "matriz_C.txt" contém exatamente cinco colunas, sendo o número de linhas dependente da aplicação. As quatro primeiras colunas contêm os valores modais A, B, C e D de cada conjunto nebuloso do sistema, sendo relacionadas primeiro às linhas referentes aos valores de entrada, e, por último, às linhas relacionadas aos valores de saída. A última coluna deverá ser inicializada com o valor -1 em todas as linhas, pois será preenchida pelo *software* durante sua execução com o valor da pertinência dos valores de entrada calculadas sobre os valores modais. A Figura 13 ilustra o conteúdo deste arquivo, correspondendo ao exemplo da figura anterior.

```
1  0   0   6   18  -1
2  10  20  20  30  -1
3  22  34  40  40  -1
4  0   0   20  50  -1
5  20  50  50  80  -1
6  50  80  100 100 -1
7  0   0   20  50  -1
8  20  50  50  80  -1
9  50  80  100 100 -1
```

Figura 13 – Matriz C de desenvolvimento.

A figura apresenta, em cada linha da matriz, um conjunto de valores modais. A primeira linha da figura indica os valores de: A=0; B=0; C=6; D=18 e u=-1, indicando que este será calculado posteriormente. Este padrão se repete para os valores modais das outras linhas da matriz.

O arquivo "matriz_R.txt" é composto por três colunas, relacionando cada uma das regras do Sistema Nebuloso. O número de linhas não depende dos outros conjuntos, correspondendo apenas ao número de regras selecionadas na etapa de calibração. A Figura 14 ilustra um conteúdo hipotético para este arquivo, relativo às figuras anteriores.

1	1	4	7
2	1	5	7
3	1	6	7
4	2	4	9
5	2	5	8
6	2	6	8
7	3	4	9
8	3	5	9
9	3	6	9

Figura 14 – Matriz R de desenvolvimento.

O entendimento da matriz de regras depende do significado de cada linha da Matriz C, sendo que cada linha representa uma regra no formato da Equação (12). No caso da primeira linha da Matriz R representada, se os valores de entrada corresponderem aos conjuntos nebulosos das linhas 1 e 4 da Matriz C, então a saída será armazenada na linha 7 da Matriz C.

Com a criação dos arquivos, é possível importar os dados para o Scilab. Isso é feito através da função `fscanfMat()`, que preenche uma variável com a matriz lida do arquivo. Após realizar o procedimento para os três arquivos, todos os dados estão inseridos no Scilab e prontos para serem processados através das variáveis:

```
E_matriz
C_matriz
R_matriz
```

5.1.2. Verificação de integridade

Após a importação dos dados para o Scilab, o *software* realiza a verificação de integridade das matrizes. Esta etapa é simples, porém de grande importância já que evita que erros na inserção de dados nos arquivos resultem na geração de valores de saída errados. A verificação de integridade analisa os dados, verificando se as matrizes inseridas possuem um número compatível de colunas.

5.1.3. Execução do Método Matricial

O primeiro passo para a execução do Método Matricial é a Fuzzyficação. Esta etapa calcula a pertinência referente aos valores de `E_matriz`, de acordo com os cálculos apresentados na Equação (5), e preenche a quinta coluna de `C_matriz` com os valores correspondentes. Para realizar os cálculos indicados pelas equações, foi criada a função `fuzzy()`, cujos parâmetros de entrada são os valores do conjunto nebuloso e o valor *crisp* de entrada, e o parâmetro de saída é a pertinência relacionada. Após esta etapa as linhas de `C_matriz` referentes aos conjuntos nebulosos de entrada estarão preenchidas, conforme exemplificada pela Figura 15.

```
 0.    0.    6.    18.    0.417
10.   20.   20.    30.    0.3
22.   34.   40.    40.    0.
 0.    0.   20.    50.    0.2
20.   50.   50.    80.    0.8
50.   80.  100.   100.    0.
 0.    0.   20.    50.  - 1.
20.   50.   50.    80.  - 1.
50.   80.  100.   100.  - 1.
```

Figura 15 – Matriz C Fuzzyficada.
Os valores de saída ainda não foram preenchidos.

As linhas desta matriz correspondem às linhas da matriz da Figura 13, porém com o valor da pertinência de saída (última coluna) preenchida com o valor fuzzyficado. No caso da primeira linha da Matriz C ilustrada acima, a pertinência do

valor de entrada 13 (da Figura 12), associado aos valores modais A=0; B=0; C=6 e D=18, resultam em uma pertinência de u=0,417.

Após a Fuzzyficação ter sido concluída com sucesso, o *software* desenvolvido inicia a aplicação das regras sobre as pertinências calculadas. As regras são aplicadas uma a uma até que todas as linhas de `R_matriz` tenham sido percorridas, resultando no preenchimento dos valores da quinta coluna de `C_matriz` correspondentes aos conjuntos de saída com as pertinências calculadas.

Após esta etapa, as regras presentes em `R_matriz` são aplicadas a todas as linhas de entrada, resultando no completo preenchimento de `C_matriz`, ilustrada na Figura 16.

```
 0.    0.     6.    18.    0.417
10.   20.    20.    30.    0.3
22.   34.    40.    40.    0.
 0.    0.    20.    50.    0.2
20.   50.    50.    80.    0.8
50.   80.   100.   100.    0.
 0.    0.    20.    50.    0.417
20.   50.    50.    80.    0.3
50.   80.   100.   100.    0.2
```

Figura 16 – Matriz C Fuzzyficada.
Os valores de saída já estão preenchidos.

As colunas desta matriz correspondem às da Figura 15, porém com os valores de saída (três últimas linhas da matriz) devidamente preenchidos.

Finalmente, o último passo é a Defuzzyficação, realizado pelo Método Matricial através da aplicação do Método do Centróide. Este é o passo no qual a maioria dos cálculos realizados está concentrada, uma vez que realiza a integração da curva de saída a partir das pertinências correspondentes.

Esta etapa se inicia com a criação do vetor "un" de N posições, sendo N configurável pelo usuário. Este vetor é preenchido com o menor valor entre a pertinência de saída do conjunto sendo avaliado e o valor fuzzyficado do mesmo conjunto no ponto de interesse. Nos pontos nos quais há a presença de mais de um

conjunto nebuloso, o valor escolhido é o maior entre os dois candidatos. As equações a seguir ilustram o cálculo realizado para cada ponto do vetor "un", e a Figura 17 mostra o gráfico de saída do programa com o vetor "un" em destaque. Os cálculos da Equação (18) são realizados para cada linha de entrada do sistema, e o conjunto dos cálculos é feito para todos os pontos da curva.

$$\forall\ linha\ de\ entrada:$$
$$C'_{Max} = \max(C, C_{Max}) \qquad (18)$$

Onde C_{Max} é valor da pertinência atual do conjunto de saída em questão, C'_{Max} é o valor da nova pertinência que irá substituir o valor antigo, e C é igual a:

$$C = \min(\mu_L, Fuzzy_L) \qquad (19)$$

Onde μ_L é a pertinência de saída da linha atual, e $Fuzzy_L$ é a pertinência referente à Fuzzyficação do valor do ponto em questão na linha atual.

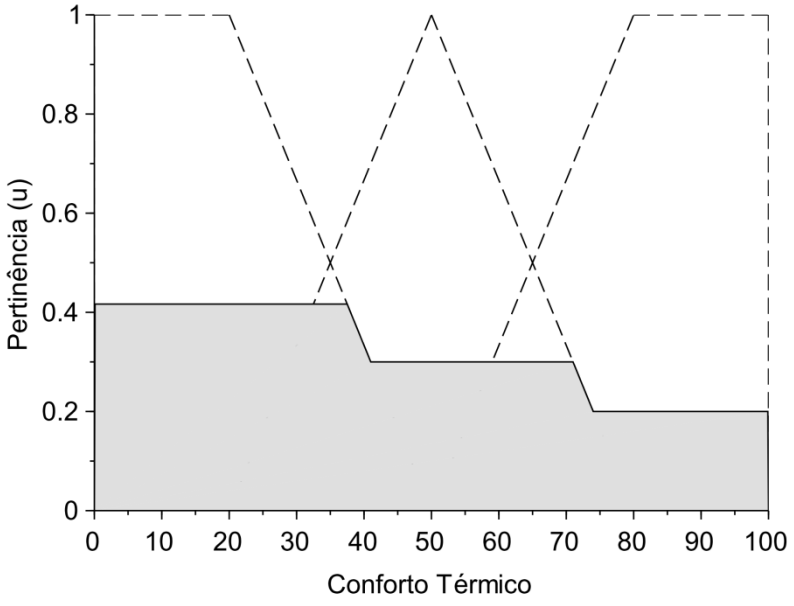

Figura 17 – Gráfico de saída do sistema com vetor "un" e sua área hachurada.

Por fim, o centróide é calculado como uma média ponderada dos pontos do vetor "un", de acordo com a Equação (17), reapresentada a seguir por conveniência.

$$crisp_{final} = \frac{\sum_{i=0}^{n}\left(Xn(i) * un(i)\right)}{\sum_{i=0}^{n}\left(un(i)\right)} \quad (17)$$

O resultado está ilustrado na Figura 18.

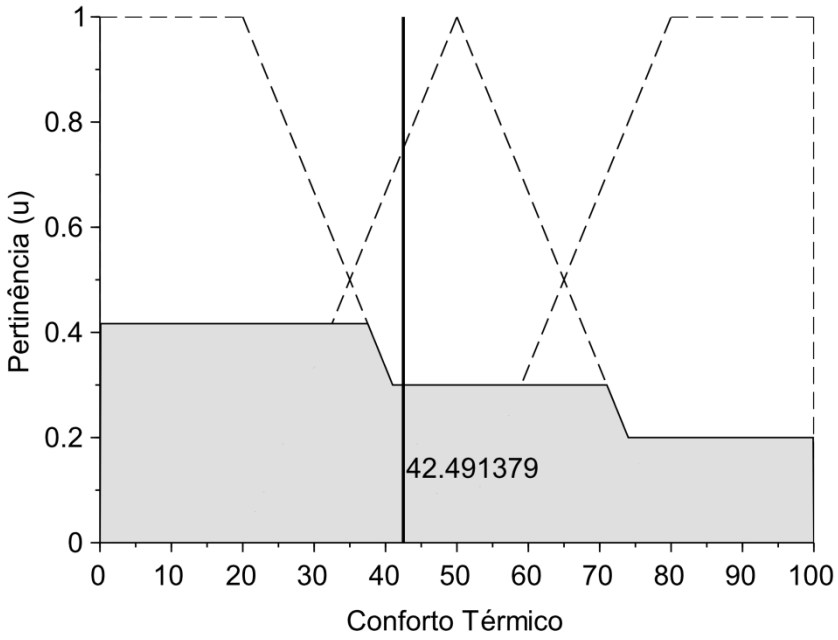

Figura 18 – Gráfico de saída do sistema com centróide calculado.

O valor final mostra a variável crisp de saída do sistema, que pode ser avaliada, de acordo com a aplicação para auxiliar na tomada de decisão junto ao sistema. Pode-se observar, neste caso, que a sensação térmica é Boa, tendendo para Fria.

5.2. Desenvolvimento de *software* para embarcar o Método Matricial no Arduino

A versão do *software* para Arduino consiste em uma biblioteca que pode ser utilizada pelas aplicações de usuário. A biblioteca, denominada `embeddedFuzzy`, deve ser adicionada ao diretório de bibliotecas do Arduino e incluída na aplicação para que possa ser utilizada. Por este motivo foi implementado também, um código de desenvolvimento que realiza esta função.

O *hardware* utilizado para a implementação e testes da biblioteca é o Arduino Nano V3.1, ilustrado na Figura 19. Este módulo conta com um processador Atmel ATmega328P [31], executando a uma frequência de 16MHz e alimentado através da

porta USB com tensão de 5V. Com memória Flash de 32KBytes e RAM de 2KBytes, este dispositivo é o maior de sua família, cuja principal característica, para este trabalho, é a ausência de *hardware* dedicado a realizar operações com números de ponto flutuante, chamado FPU (*Floating Point Unit*).

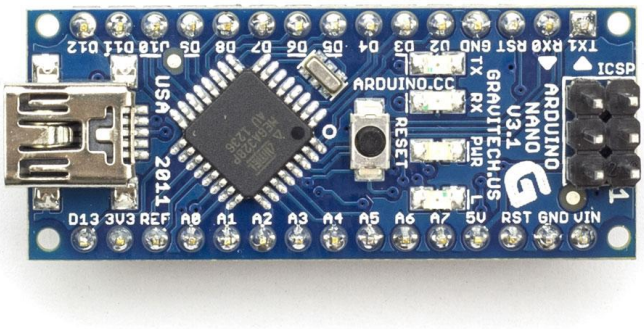

Figura 19 – Placa do Arduino Nano V3.1. Figura retirada de [32].

A biblioteca foi criada em linguagem C++, porém a maioria dos recursos utilizados são da linguagem C, o que permite que ela seja facilmente reorganizada para um arquivo em C puro. Basicamente, foram utilizadas as estruturas de classes do C++ para organizar as funções e variáveis, criando-se o famoso "C estruturado".

O algoritmo em C se assemelha em muitos pontos à implementação do Scilab. As diferenças se concentram principalmente nas características da linguagem utilizada, mantendo-se o algoritmo praticamente inalterado. Uma vez que o módulo Arduino utilizado não possui interface com o usuário, os gráficos não fazem parte do programa, sendo utilizada a interface serial para transmissão de texto em ASCII para o computador de desenvolvimento.

Por esse motivo, a principal diferença entre a implementação feita no Scilab e a implementação feita no Arduino, é que a leitura dos dados de entrada não é feita a partir de um arquivo, e sim de variáveis. Desse modo, durante a execução da aplicação, esta pode alterar a matriz que contém os dados de entrada e executar a função que realiza os cálculos.

O *software* foi criado com base em variáveis do tipo `float`, ou seja, números de 32 bits que podem variar de $3.4028235 \times 10^{-38}$ até 3.4028235×10^{38}, de acordo com [33]. O motivo desta escolha foi testar a capacidade de processamento das diversas plataformas embarcadas com números que necessitam de alto processamento. Caso a aplicação seja tolerante quanto à inserção de erro pela perda de informação, os cálculos poderiam ser realizados com números inteiros resultantes da multiplicação do valor original por uma constante de precisão para manter as casas decimais desejadas, bastando o código da biblioteca ser alterado para realizar a modificação.

Para o desenvolvimento do *software*, foi criada uma estrutura de classes C++, cuja instanciação ocorre automaticamente com a inclusão do cabeçalho com o comando "`#include <embeddedFuzzy.h>`", portanto o usuário deverá utilizar a classe `eFuzzy` diretamente em sua aplicação. A biblioteca deverá ser invocada em duas etapas, sendo necessário configurá-la com as matrizes de entrada e depois executar a função que realiza os cálculos.

A configuração das matrizes deve ser feita através da função `eFuzzy.configure()`, que recebe como parâmetros os ponteiros para as matrizes de dados `mC`, `mR` e `mE` criadas pelo usuário. Esta função também é responsável pelo cálculo de algumas grandezas importantes para os cálculos, como o número total de linhas de entrada e saída do sistema.

Uma vez configuradas as matrizes, o sistema está pronto para ser executado. Isto é feito na aplicação do usuário através da função `eFuzzy.run()`, que não recebe nenhum parâmetro e executa o Método Matricial sobre as matrizes previamente inseridas. Caso a aplicação de usuário altere as matrizes, como no caso da substituição dos dados de entrada, basta executar a função `eFuzzy.run()` novamente, não sendo necessária nova configuração.

A execução do código da biblioteca segue os mesmos princípios explicados para o Scilab. Primeiramente, são calculadas as pertinências, e a última coluna da matriz `mC` referente aos conjuntos nebulosos de entrada é preenchida. Em seguida, todas as regras da matriz `mR` são aplicadas, completando o preenchimento de `mC`.

Por fim, é realizada a Defuzzyficação dos conjuntos nebulosos de saída e o consequente cálculo da variável de saída através do Método do Centróide.

As informações apresentadas ao usuário através da interface serial do Arduino, quando alimentado com as matrizes de desenvolvimento, estão ilustradas na Figura 20.

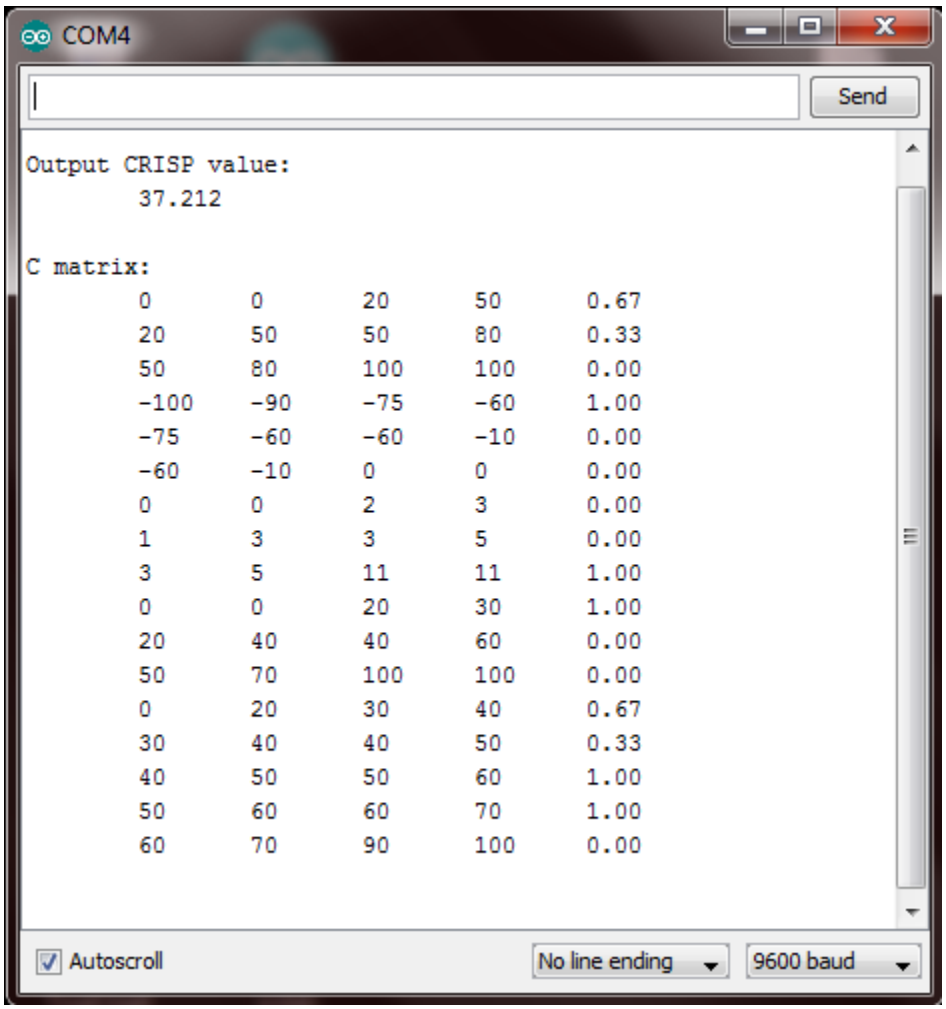

Figura 20 – Dados de saída através da interface Serial do Arduino.

5.3. Desenvolvimento de *software* para embarcar o Método Matricial em dispositivos com Linux OpenWRT

O módulo utilizado para a implementação do Método Matricial em dispositivos com Linux Embarcado foi o roteador portátil TP-Link MR3020 [34], ilustrado na Figura 21. Este módulo possui um SoC (*System on Chip*) Atheros AR9331, que integra um módulo de comunicação sem fio (Wi-Fi) a um processador AR7240 de

32-bits com arquitetura RISC, sendo executado a uma frequência de 400MHz. O processador disponível no SoC não possui FPU em *hardware* para aceleração dos cálculos com números decimais.

Figura 21 – Foto do roteador portátil TP-Link MR3020 (Retirada de [35])
e sua placa de circuito impresso interna (Retirada de [7]).

O roteador portátil MR-3020 é alimentado através de um conector 5V USB, porém a conexão com o computador de desenvolvimento é feita através de um cabo Ethernet ou pela interface Wi-Fi. Esta conexão possibilita acessar o dispositivo através de um terminal, permitindo que o *software* seja transferido para o dispositivo, executado, e que as informações de saída sejam visualizadas.

O roteador MR3020 foi escolhido por ser um dos dispositivos suportados pela distribuição Linux de código aberto OpenWRT, por isso deve ser realizada a substituição do *firmware* de fábrica. A substituição do *firmware* em um equipamento ainda não modificado é simples, sendo realizada através da interface *web* disponibilizada pelo equipamento no IP 192.168.0.254, selecionando-se o arquivo binário desejado na tela de atualização de *firmware* e clicando no botão "*Upgrade*". Após o processamento da atualização, o módulo irá iniciar com a distribuição OpenWRT instalada. Outros métodos de substituição do *firmware* somente são necessários caso ocorram erros durante o processo, ou o *firmware* precise ser reinstalado no módulo. No momento da redação deste documento, a versão estável mais recente da distribuição é a "Barrier Breaker 14.07", utilizada neste trabalho.

A diferença básica entre este *software* e o do Arduino está nas funções de impressão de dados, que, neste caso, possuem suporte ao `printf()`. Também foi alterada a aplicação de teste que realiza o acesso à biblioteca Fuzzy, pois neste caso a inclusão, acesso e bibliotecas padrão são diferentes. Portanto, se comparados, os códigos desta biblioteca e do Arduino se mostrarão muito parecidos, com sua estrutura de cálculos inalterada.

Para que o *software* criado possa ser executado no processador em questão foi necessário o uso de um *Cross Compiler*, nome dado ao programa que compila *softwares* para outra arquitetura, diferente da arquitetura na qual ele está sendo executado. Neste caso o *Cross Compiler* utilizado é uma versão modificada do BuildRoot [36], que está disponível na página de *downloads* do OpenWRT [37].

O *Cross Compiler* foi instalado na máquina de desenvolvimento, realizando-se o *download* do pacote de arquivos, seguido da extração deste pacote. Para que o *toolchain* (conjunto de ferramentas para *Cross* Compilação) fosse construído, foi executado o comando `make` no diretório raiz dos códigos fonte:

```
$ make
```

Com o *Cross Compiler* compilado, as variáveis de ambiente foram exportadas para utilização no sistema através da adição dos seguintes comandos ao final do arquivo `/etc/bash.bashrc`:

```
export PATH=$PATH:<path>/openwrt/staging_dir/toolchain-
mips_34kc_gcc-4.8-linaro_uClibc-0.9.33.2/bin

export STAGING_DIR=<path>/openwrt/staging_dir

export CROSS_COMPILER_PREFIX=mips-openwrt-linux-
```

Nos quais `<path>` deve ser substituído pelo diretório no qual os arquivos do *Cross Compiler* foram construídos.

Nota-se que o prefixo do *Cross Compiler* é `mips-openwrt-linux-`, que deve ser adicionado do sufixo do programa desejado (geralmente gcc ou g++) quando invocado através do terminal. Um exemplo de invocação do *Cross Compiler* para a compilação de um programa é:

```
$ mips-openwrt-linux-gcc teste.c -o teste.o
```

Além disso, a variável de ambiente `PATH` informa que a versão do *Cross Compiler* utilizada é a 4.8, e a versão da biblioteca uClibc é a 0.9.33.2. Foram utilizadas as versões mais atuais disponíveis durante a redação deste trabalho.

Para verificar se o *Cross Compiler* foi executado com sucesso, basta executar o comando:

```
$ file <arquivo>.o
```

Sua saída indica as características do arquivo binário compilado, como no exemplo a seguir, que ilustra a saída do comando para um arquivo compilado para a arquitetura x86, e outro *Cross* Compilado para a arquitetura MIPS (*Microprocessor without Interlocked Pipeline Stages*):

```
$ file teste_x86.o

teste_x86.o: ELF 64-bit LSB executable, x86-64, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux
2.6.24, not stripped

$ file teste_mips.o

teste_mips.o: ELF 32-bit MSB executable, MIPS, MIPS32 rel2
version 1, dynamically linked (uses shared libs), not stripped
```

O projeto de *software* final foi criado com o auxílio de um arquivo do tipo *Makefile*, cujo conteúdo está presente no Apêndice 2. Para *Cross* Compilar o projeto basta executar o comando:

```
$ make
```

O diretório corrente deve ser o local no qual o projeto está presente.

6. RESULTADOS

Conforme citado anteriormente, este trabalho tem por objetivo embarcar o Método Matricial para análise nebulosa descrito em [5], enfocando as características de baixo consumo de processamento e energia. Nesse contexto, este trabalho apresenta três implementações de *software* que contemplam o citado Método Matricial: (a) implementação de *software* para prova de conceito do Método Matricial, desenvolvido no Scilab; (b) implementação do Método Matricial embarcado no Arduino e (c) implementação do Método Matricial em Linux embarcado.

O código implementado para o Scilab, explicado em detalhes no capítulo anterior, foi implementado com base nas especificações teóricas do Método Matricial, portanto fez-se necessária a comparação dos resultados gerados pelo algoritmo com outro *software*, com o objetivo de validação da implementação. Esta comparação foi realizada com o auxílio do *software* original, também escrito para o Scilab e gentilmente cedido pelos autores do Método Matricial [5]. O código original do Método Matricial para o Scilab está presente no Apêndice 3.

Como os *softwares* criados neste trabalho são voltados para a implementação genérica do Método Matricial, e para efeito de comparação e estudos de caso, as matrizes e dados utilizados nos testes são os relacionados à detecção de ofensores em redes IEEE 802.11, uma vez que apresentam um cenário real para a aplicação e testes das implementações. As matrizes utilizadas estão presentes no Apêndice 1.

6.1. Testes de validação

Para validar o correto funcionamento da implementação desenvolvida neste trabalho, foi necessário comparar o seu resultado com outra implementação do Método Matricial, eliminando, assim, possíveis divergências de interpretação ou específicas do código criado. Para este caso, a implementação do algoritmo no Scilab criada neste trabalho foi comparada com o *software* original criado pelos autores do Método Matricial [5].

A primeira comparação a ser realizada é o correto preenchimento da Matriz C, que deve ser equivalente para todas as implementações do algoritmo, já que as etapas envolvidas no processo (cálculo das pertinências e aplicação de regras) independem de variáveis configuráveis pelo usuário, diferentemente da Defuzzyficação, com será mostrado adiante.

Para realizar a comparação do preenchimento da Matriz C foram utilizadas as matrizes do estudo de caso dos Ofensores IEEE 802.11, sendo os dados de entrada equivalentes à coluna A da Tabela II, presente na referência [19]. Os dados foram carregados nos dois *softwares* e estes foram executados, sendo ilustradas na Figura 22 e na Figura 23 as matrizes geradas pelo *software* original e pela implementação criada neste trabalho, respectivamente.

```
    0.5     0.8     1.      1.      0.
    0.2     0.5     0.5     0.8     0.3333333
    0.      0.      0.2     0.5     0.6666667
    0.      0.      10.     60.     0.3654
    10.     60.     60.     75.     0.6346
    60.     75.     90.     100.    0.
    3.      5.      11.     11.     1.
    1.      3.      3.      5.      0.
    0.      0.      2.      3.      0.
    0.5     0.7     1.      1.      0.
    0.2     0.4     0.4     0.6     0.
    0.      0.      0.2     0.3     0.
    0.6     0.7     0.9     1.      0.
    0.5     0.6     0.6     0.7     0.3333333
    0.4     0.5     0.5     0.6     0.3333333
    0.3     0.4     0.4     0.5     0.6346
    0.      0.2     0.3     0.4     0.3654
-----> centroide somatoria 0.350264
```

Figura 22 – Dados de saída do *software* para Scilab originalmente descrito em [5].

```
    0.5     0.8     1.      1.      0.
    0.2     0.5     0.5     0.8     0.3333333
    0.      0.      0.2     0.5     0.6666667
    0.      0.      10.     60.     0.3654
    10.     60.     60.     75.     0.6346
    60.     75.     90.     100.    0.
    3.      5.      11.     11.     1.
    1.      3.      3.      5.      0.
    0.      0.      2.      3.      0.
    0.5     0.7     1.      1.      0.
    0.2     0.4     0.4     0.6     0.
    0.      0.      0.2     0.3     0.
    0.6     0.7     0.9     1.      0.
    0.5     0.6     0.6     0.7     0.3333333
    0.4     0.5     0.5     0.6     0.3333333
    0.3     0.4     0.4     0.5     0.6346
    0.      0.2     0.3     0.4     0.3654

**************************
Valor CRISP final: 0.350264
**************************
```

Figura 23 – Dados de saída do *software* para Scilab criado neste trabalho.

Pode-se observar que as pertinências associadas aos conjuntos nebulosos de entrada do sistema (a última coluna das primeiras doze linhas da matriz) são compatíveis, o que indica que a etapa de Fuzzyficação está correta. Analogamente, com a confirmação de que as pertinências associadas aos conjuntos de saída do sistema (a última coluna das últimas cinco linhas da matriz) são idênticas, pode-se afirmar que a aplicação das regras presentes na Matriz R também apresentou resultados corretos.

A comparação final dos resultados obtidos por ambos os *softwares* se dá pelo resultado final da aplicação do Método Matricial. O valor de saída do sistema, diferentemente das etapas anteriores, pode ser influenciado por constantes configuráveis nos algoritmos, ou seja, a etapa de Defuzzyficação e cálculo do Centróide são dependentes de valores escolhidos pelo usuário. A variável em questão indica o número de pontos que serão utilizados para a construção do gráfico de envoltória relativo às pertinências de saída do sistema, ou seja, quantos pontos irão constituir o vetor da envoltória a ser preenchido pelo algoritmo.

Na implementação do Método Matricial criada neste trabalho, a variável que indica a granularidade do gráfico de envoltória é chamada `n_pontos`. Na Figura 22

e na Figura 23, os valores utilizados para esta variável foram iguais ao utilizado pelos autores em [19], ou seja: `n_pontos = 10`. Portanto, pode-se observar nas figuras que os valores de saída de ambos os sistemas são idênticos quando utilizam a mesma quantidade de pontos, o que complementa a prova anterior de que a implementação para o Scilab do Método Matricial é compatível com o modelo. Este teste foi repetido para as implementações para Arduino e OpenWRT, provando que os resultados obtidos nas três plataformas são equivalentes.

6.2. Influência da Defuzzyficação e Centróide

Um ponto importante observado durante a execução do experimento anterior é que a Defuzzyficação e o cálculo do Centróide possuem papéis que merecem atenção especial nos resultados deste trabalho. Uma vez que a quantidade de pontos destas etapas é configurável, elas influenciam no resultado obtido.

É correto afirmar que a precisão obtida no valor de saída será maior com o aumento do valor de `n_pontos`, porém este valor não deve ser elevado sem uma análise prévia da sua real necessidade. Ao aumentar o valor desta variável, o processamento necessário para realizar o cálculo sobre o vetor também aumenta, como será demonstrado adiante. Além disso, o vetor da envoltória contém, necessariamente, o mesmo número de posições indicado pela variável `n_pontos`, portanto pode haver uma limitação de memória RAM. No caso da implementação para o Arduino pôde ser observado que o sistema não suporta a configuração de `n_pontos = 1000`, que faz com que o sistema trave, porém a parametrização deste valor depende da aplicação na qual o Método Matricial está sendo utilizado. Sendo assim, não é possível fornecer um valor limite pré-estabelecido para a configuração desta variável.

Apesar do uso de uma alta granularidade do gráfico de envoltória tornar o resultado mais preciso, talvez tal precisão não seja realmente necessária uma vez que o resultado esperado para o sistema é uma tomada de decisão (geralmente com valores linguísticos). Mesmo com o uso de um valor de `n_pontos` baixo, prejudicando a precisão do resultado final, é possível obter uma tomada de decisão satisfatória para a aplicação em questão. Na etapa de calibração das matrizes por um especialista é necessário identificar o valor de `n_pontos` que permite reduzir o

processamento sem que a tomada de decisão seja influenciada, reduzindo o consumo energético e de processamento sem alterar a tomada de decisão.

6.3. Testes de desempenho

Para mensurar o gasto energético das aplicações embarcadas foram utilizadas medições de tempo de execução, o que permite prever como será o comportamento do Método Matricial quando utilizado por uma aplicação. Além disso, pode-se inferir, caso necessário, como ele irá se comportar em outras plataformas de *hardware* como, por exemplo, diferentes frequências do processador.

Os testes foram realizados com as matrizes de [19], para que representem um caso de uso real do Método Matricial. Os resultados serão diferentes para outras matrizes e aplicações.

6.3.1. Desempenho no Arduino

Todos os testes de temporização no Arduino foram realizados em um *loop* de 1 segundo, permitindo que fosse observada visualmente a estabilidade dos intervalos de tempo durante os testes. Desse modo, as medidas aqui apresentadas são reais, não sendo necessário calcular a média de um dado intervalo.

Para o primeiro teste no Arduino, um pino de uso geral foi configurado para permanecer em nível alto durante a execução completa do Método Matricial, sendo medido em um osciloscópio Link Instruments, modelo MSO-19.2, o tempo decorrido entre a borda de subida e descida do sinal. O resultado está ilustrado na Figura 24. Nessa figura o eixo X representa o tempo, no qual cada divisória quadrada representa um intervalo de tempo de 5ms, e o eixo Y representa a tensão aplicada ao pino de uso geral. O cursor está posicionado no fim da forma de onda, indicando o período do pulso.

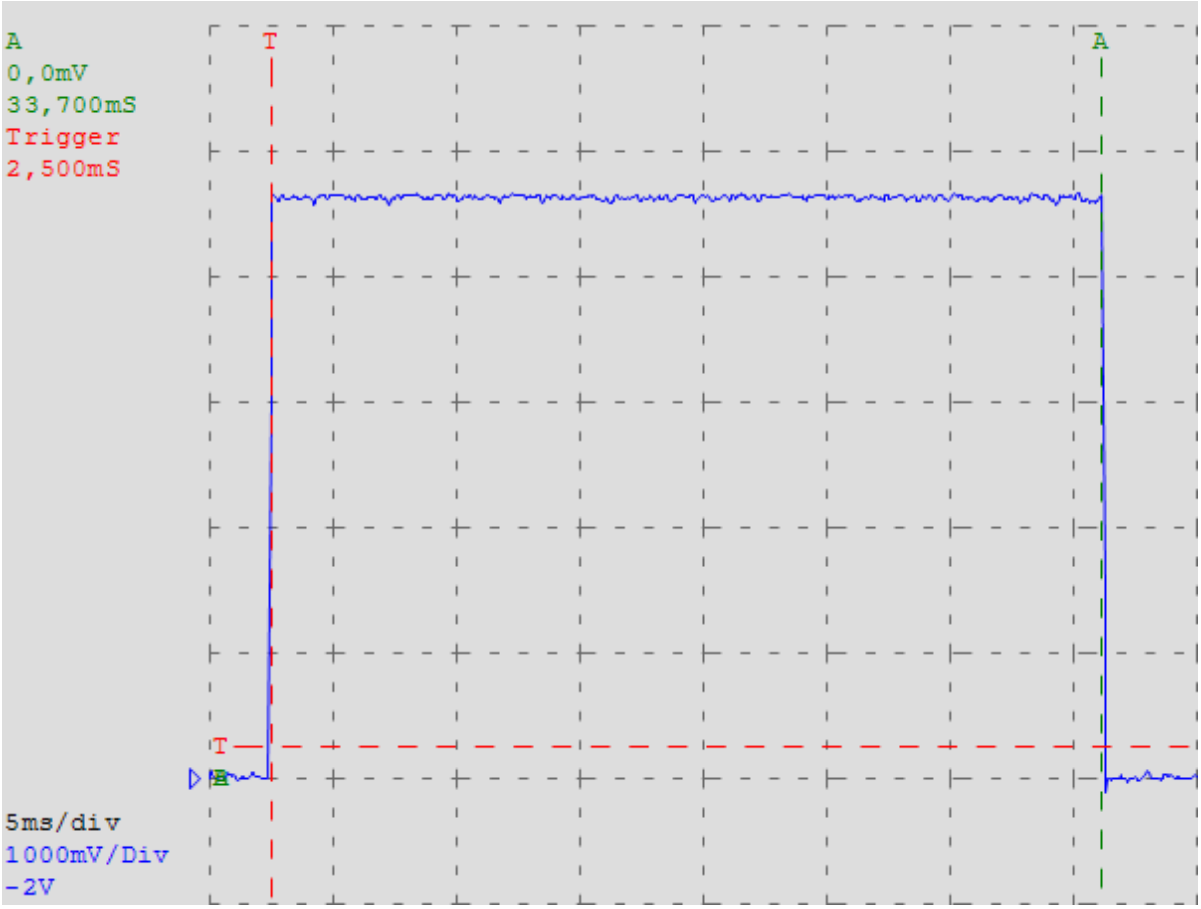

Figura 24 – Tempo total de execução no Arduino.
n_pontos = 100.

Como o cursor está posicionado na borda de descida do pulso, pode-se observar pelas informações presentes no canto superior esquerdo da figura que o tempo total de execução das matrizes de ofensividade no Método Matricial do Arduino, com `n_pontos=100`, é de 33,7ms.

Com este resultado em mãos, o pino do Arduino foi reconfigurado para que permanecesse em nível alto somente durante a execução de cada etapa do Método Matricial. A Figura 25 mostra a forma de onda adquirida com a indicação das etapas correspondentes a cada ciclo.

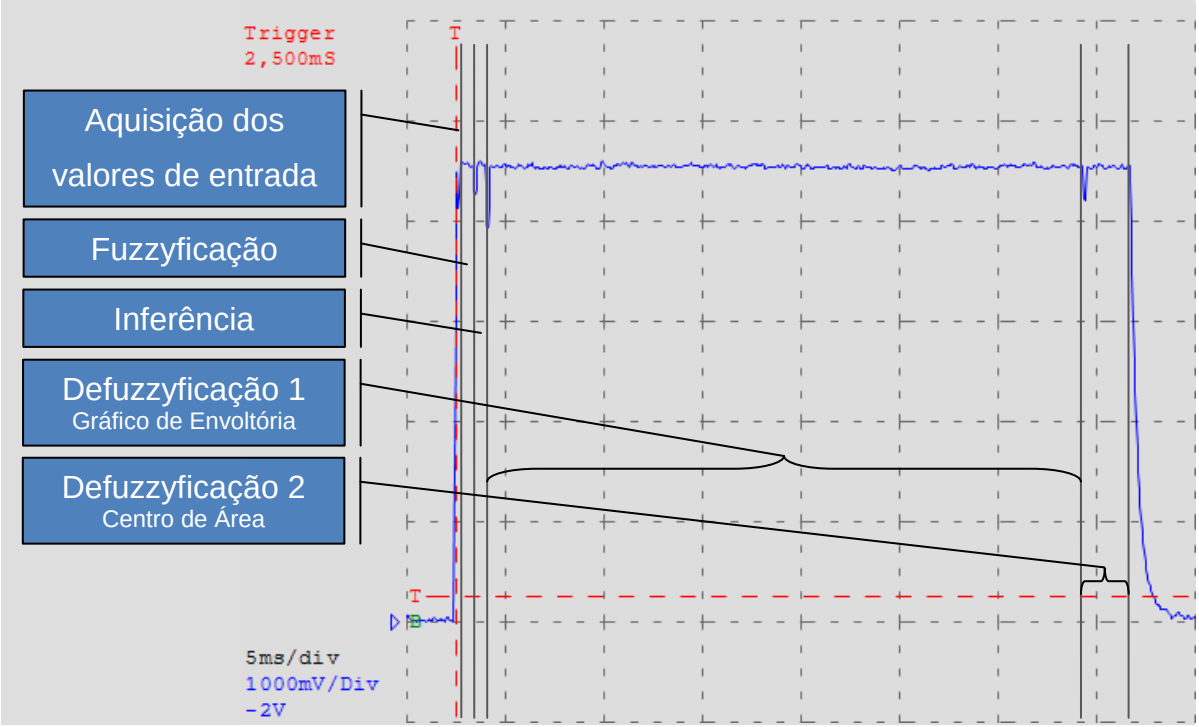

Figura 25 – Tempo de execução de cada etapa no Arduino.
n_pontos = 100.

As etapas mostradas na figura são referentes, da esquerda para a direita, às funções `doExtractUtilInfo()`, `doCalculatePertinence()`, `doApplyRules()`, `doDefuzzy()` e `doFindCentroid()`, que representam, respectivamente, às etapas de: aquisição dos valores de entrada (item 3.2.1), Fuzzyficação (item 3.2.2), Base de Regras e Inferência (itens 3.2.3 e 3.2.4), Defuzzyficação (item 3.2.5) e cálculo dos valores de saída (item 3.2.6). Em comparação com a Figura 24 pode-se notar visualmente que o tempo total de execução aumentou, o que é explicado pelo tempo entre as transições intermediárias do sinal adicionadas manualmente ao código neste teste para que o osciloscópio fosse capaz de identificar as rápidas transições entre funções. Este aumento pode ser descartado, já que esta figura é apenas uma comparação entre a duração de cada etapa do processo.

Em seguida, foram realizadas medições focadas em cada etapa do método, visando uma medição precisa do tempo de execução de cada uma delas. Para este teste, o pino do Arduino foi configurado para permanecer em nível alto durante a

execução de cada etapa individualmente, sendo os tempos medidos através dos cursores do osciloscópio. Os resultados estão representados na Tabela 1.

Tabela 1 – Tempo de execução de cada etapa no Arduino.
n_pontos = 100.

Função:	**Etapa (item 3.2)**	**Tempo de execução:**
doExtractUtilInfo()	Valores de Entrada	0,074 ms
doCalculatePertinence()	Fuzzyficação	0,773 ms
doApplyRules()	Base de Regras e Inferência	0,502 ms
doDefuzzy()	Defuzzyficação (Gráfico de envoltória)	30,100 ms
doFindCentroid()	Defuzzyficação (Centro de Área)	2,250 ms

A análise desta tabela, em conjunto com a Figura 25, mostra que o maior consumo está nas duas últimas etapas do processo. Estas medições complementam as afirmações de 6.2 sobre a importância das etapas de Defuzzyficação e cálculo do Centróide, uma vez que ficam claros os maiores tempos de processamento consumidos durante sua execução. A pequena diferença observada entre o tempo total medido e a soma do tempo de execução de cada função se dá devido ao acúmulo de erros de medição e pela natureza não sequencial dos testes realizados.

Para comparação do efeito causado pela granularidade do gráfico de envoltória as medições foram repetidas para `n_pontos=10`. Os resultados são apresentados apenas para o tempo total de execução e para as funções `doDefuzzy()` e `doFindCentroid()`, uma vez que as outras funções não dependem desta variável para realizar os cálculos e possuem o mesmo tempo de execução indicado na Tabela 1. A Figura 26 mostra o tempo total da execução do Método Matricial no Arduino, para este caso.

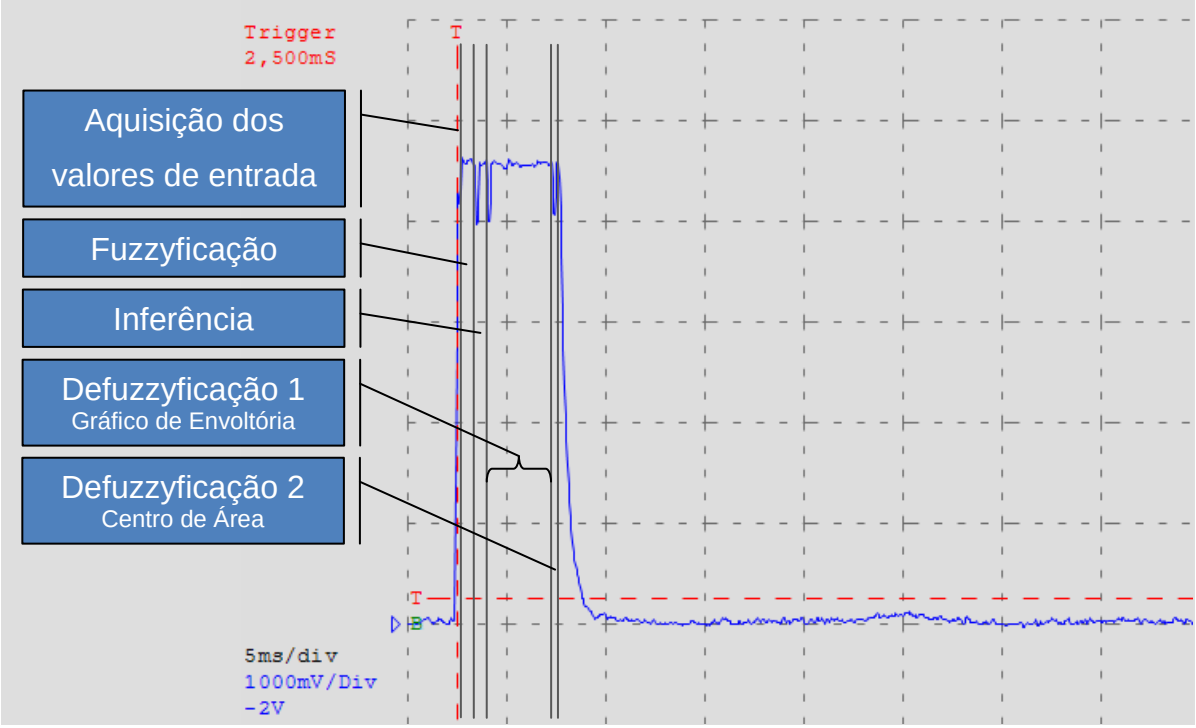

Figura 26 – Tempo de execução de cada etapa no Arduino.
n_pontos = 10.

Com um tempo total de execução de 4,79ms e escala idêntica à Figura 25, este gráfico mostra claramente que o tempo de execução das três primeiras etapas se manteve o mesmo. A diminuição do tempo nas duas últimas etapas mostra que o processamento pode, realmente, ser minimizado através de um valor de `n_pontos` que entregue o melhor custo benefício para com a tomada de decisão da aplicação.

A Tabela 2 compara o tempo de execução das duas últimas etapas do Método Matricial com a variação da granularidade do gráfico de envoltória.

Tabela 2 – Tempo de execução das duas últimas etapas no Arduino.
n_pontos = 10 e n_pontos = 100.

Função:	**Tempo de execução (n_pontos = 100)**	**Tempo de execução (n_pontos = 10)**
doDefuzzy()	30,100 ms	3,200 ms
doFindCentroid ()	2,250 ms	0,256 ms

6.3.2. Desempenho no OpenWRT

A execução dos testes de desempenho no OpenWRT foram diferentes das realizadas no Arduino, principalmente devido à natureza da aplicação. Na distribuição Linux as medições foram realizadas pelo ponto de vista de uma aplicação que chama o *software*, como o uso esperado do programa criado. Deste modo, o tempo de execução medido será o tempo total de execução da aplicação.

Como a execução de programas em um sistema operacional envolve maior complexidade do que em um sistema *bare metal*, como o Arduino, pode haver a ocorrência de eventos assíncronos, como, por exemplo, o escalonamento de processos. Estas ocorrências fazem com que a aplicação deixe de ser executada pelo processador por um determinado período de tempo, o que exclui a possibilidade da utilização do procedimento executado no Arduino.

Para este cenário foi utilizado um programa padrão nas distribuições Linux que permite medir o tempo que o processo permaneceu em atividade no processador. Este programa, chamado t*ime*, é executado através do comando de mesmo nome e fornece várias informações sobre a execução de um dado programa pelo processador.

Para a execução dos testes de desempenho no OpenWRT o programa foi executado mil vezes e o tempo resultante foi dividido por mil, sendo esta média computada com o objetivo de homogeneizar as ações e preempções do sistema operacional, além de minimizar as possíveis ocorrências de *Time Slices* de duração muito curta ou muito elevada.

A função `time` deve ser invocada com um parâmetro indicando o programa que deve ser medido. Para que fosse possível realizar a média de mil medições foi necessário construir um *script* auxiliar. Esse *script*, escrito na linguagem Shell Script, é chamado `efuzzy_time_profile.sh` e possui o seguinte conteúdo:

```
#!/bin/sh
for i in $(seq 1 1000); do './efuzzy.o'; done
```

Portanto, a linha de comando utilizada para a execução dos testes mostrados a seguir é:

```
$ time -v ./efuzzy_time_profile.sh
```

A Figura 27 mostra a saída resultante da execução do comando.

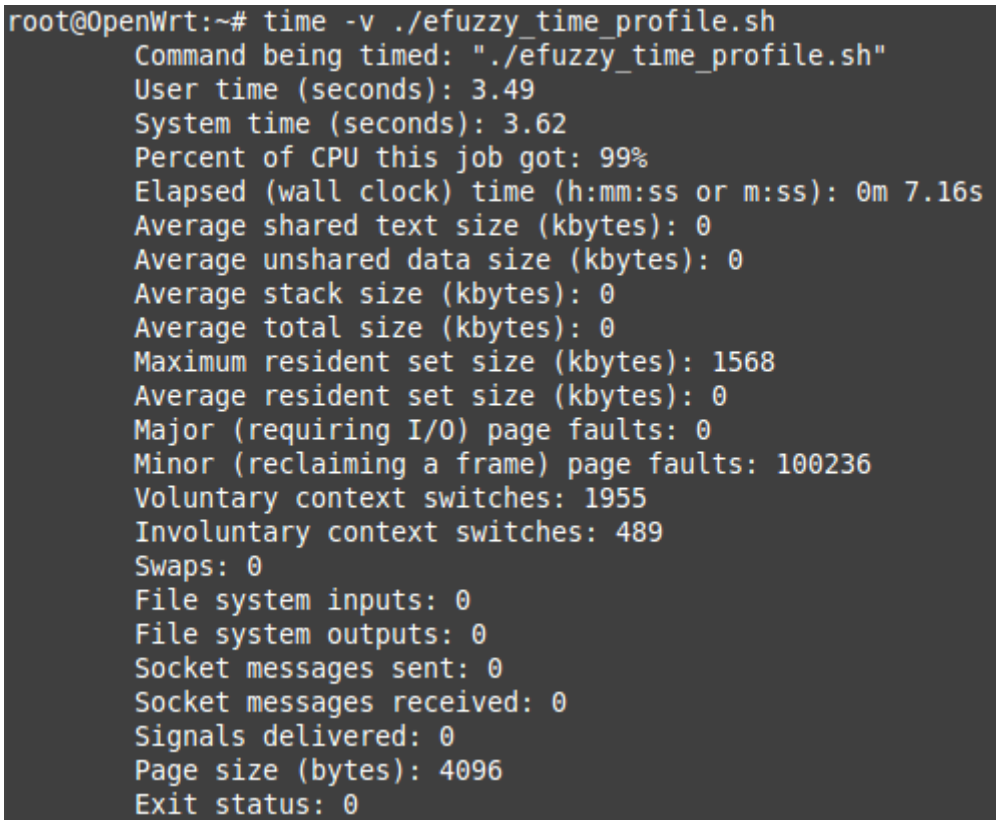

Figura 27 – Tempo total de execução no OpenWRT sem carga na CPU.
n_pontos = 100.

Dentre as informações entregues pelo comando `time` podem ser observados os intervalos de tempo dedicados pelo processador para atender às requisições do programa. A primeira informação importante, denominada *User time*, indica o tempo que o processador utilizou para atender às requisições do programa em espaço de usuário. A segunda informação de interesse, denominada *System time*, indica o tempo total que o processador utilizou para processar as chamadas de sistema em espaço de *kernel*.

Sendo assim, o tempo total dedicado pelo processador para o programa em questão é a soma dos valores de *User time* e *System time*. Neste caso, mil execuções levaram 7,11s para serem processadas, portanto cada execução foi processada, em média, a cada 7,11ms.

Uma característica importante dessa medição é o tempo total que o programa teve o processador disponível para realizar as suas tarefas. No campo "*Percent of CPU this job got*" é possível verificar que não existia outro processo concorrendo com a execução do programa, portanto o tempo total de execução das mil medidas (*Elapsed time*) está próximo do tempo que o processador se dedicou a ele. Na Figura 28 está ilustrada a execução do mesmo *script* do caso anterior, porém desta vez o processador possui uma carga maior de processamento de outros programas.

```
root@OpenWrt:~# time -v ./efuzzy_time_profile.sh
        Command being timed: "./efuzzy_time_profile.sh"
        User time (seconds): 3.63
        System time (seconds): 3.66
        Percent of CPU this job got: 38%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0m 19.10s
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 1568
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 100231
        Voluntary context switches: 1982
        Involuntary context switches: 752
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

Figura 28 – Tempo total de execução no OpenWRT com carga na CPU.
n_pontos = 100.

Observa-se que, neste caso, o tempo que o processador se dedicou ao programa (*User time* + *System time*) se encontra próximo do caso anterior, porém o tempo total de execução (*Elapsed time*) é muito maior. Este fato se deve à baixa porcentagem de uso do processador, que precisava se dedicar à execução de outras tarefas em paralelo. O tempo de cada execução do Método Matricial, como observado pelo usuário, aumentou de 7,16ms para 19,10ms devido a este fenômeno.

7. CONCLUSÕES

Com base na análise dos resultados obtidos, é possível concluir que o Método Matricial para análises nebulosas, originalmente descrito em [5], é aplicável a dispositivos embarcados. A implementação dos *softwares* para Arduino e OpenWRT são exemplos de que o algoritmo é funcional e pode ser utilizado em plataformas de baixo poder de processamento e baixo consumo de energia.

Dependendo das características específicas exigidas pela aplicação, tais *softwares* podem fazer uso dos cálculos com ponto flutuante, como está implementado, ou podem ser realizadas alterações para transformar os tipos utilizados em valores inteiros. Neste caso, a precisão deve ser avaliada e as alterações necessárias devem ser aplicadas aos códigos.

Observando-se os resultados dos testes de tempo de execução, pode-se dizer que eles são compatíveis com o estudo de caso da Ofensividade em redes IEEE 802.11, no qual os cálculos podem ser realizados pelo AP em tempo real. Apesar deste trabalho não abordar a implementação dos algoritmos em uma rede real para a mitigação da Ofensividade em redes IEEE 802.11, é possível que uma aplicação utilize os softwares criados para atuar automaticamente na tomada de decisão desta aplicação específica. Desse modo, um AP inteligente poderia mitigar a ofensividade através, por exemplo, da punição da estação ofensora, resultando em um melhor aproveitamento da banda disponível pelas outras estações.

Além deste estudo de caso, conclui-se que o tempo de execução do Método Matricial em dispositivos embarcados pode ser compatível com aplicações diversas. No caso da presença de um SO deve-se levar em consideração a capacidade de processamento da CPU e sua carga durante a execução do algoritmo, tornando possível analisar se o seu uso se aplica no caso em questão.

Por fim, mostrou-se através dos testes que é grande a influência da granularidade do gráfico de envoltória no desempenho total do algoritmo. O balanceamento entre a precisão dos cálculos e o seu impacto na tomada de decisão é vital para atingir o máximo de desempenho do sistema, sendo responsável diretamente pela diminuição do consumo energético do *software*.

Por depender da aplicação à qual se destina, este balanceamento deve ser realizado pelo especialista individualmente para cada cenário de uso, podendo em algum momento até mesmo se tornar inviável. Sendo assim, este tema permanece aberto para pesquisas e aprimoramentos futuros na indústria e na academia.

Com a conclusão deste trabalho, ficam inseridas no meio acadêmico as implementações de referência para o uso do Método Matricial em sistemas embarcados. Forma-se assim uma enorme gama de possibilidades para uso do *software*, tanto para pesquisa, quanto para aplicações de mercado.

8. REFERÊNCIAS

[1] M. Heusse, F. Rousseau, G. Berger-Sabbatel and A. Duda, "Performance anomaly of 802.11b," *IEEE Societies,* vol. 2, pp. 836-843, 2003.

[2] IEEE 802.11 Standard, [Online]. Available: http://standards.ieee.org/about/get/802/802.11.html. [Acesso em 31 10 2014].

[3] O. C. Branquinho, N. Reggiani and D. M. Ferreira, "Mitigating 802.11 MAC Anomaly Using SNR to Control Backoff Contention Window," *IEEE Computer Society,* pp. 55-61, 2006.

[4] H. Kim, S. Yun, I. Kang and S. Bank, "Resolving 802.11 Performance Anomalies through QoS Differentiation," *IEEE Communications Letters,* vol. 9, pp. 655-657, 07 July 2005.

[5] A. R. C. Siqueira, “Análise Matricial Nebulosa de Indicadores Para Apoio á Tomada de Decisão na Governança de TIC,” Dissertação de Mestrado, Pontifícia Universidade Católica de Campinas, Campinas, 2013.

[6] Arduino, [Online]. Available: www.arduino.cc. [Acesso em 31 10 2014].

[7] OpenWRT: TP-Link MR3020 Guide, [Online]. Available: http://wiki.openwrt.org/toh/tp-link/tl-mr3020. [Acesso em 31 10 2014].

[8] IEEE 802 Comitee, [Online]. Available: http://www.ieee802.org. [Acesso em 31 10 2014].

[9] T. T. Ganselli, “Implantação do CSMA/CA em Redes de Sensores Radiuino,” Trabalho de Conclusão do Curso de Engenharia Elétrica com Habilitação em Telecomunicações, Pontifícia Universidade Católica de Campinas, Campinas, 2012.

[10] C. P. C. Marques, “Identificação de Ofensores via Análise da Sensibilidade de Estações na Vazão de Redes IEEE 802.11,” Dissertação de Mestrado, Pontifícia Universidade Católica de Campinas, Campinas, 2013.

[11] A. S. Tanenbaum, Computer Networks, 4th ed., Prentice Hall, 2007.

[12] L. A. Zadeh, "Fuzzy Sets," *Information and Control,* vol. 8, no. 3, pp. 338-353, 1965.

[13] J. M. Mendel, "Fuzzy Logic Systems for Engineering: A Tutorial," *Proceedings of the IEEE,* vol. 83, no. 3, pp. 345-377, 1995.

[14] C. M. F. Carlson, J. R. F. Formigoni and H. M. F. Tavares, "Planning of Optical Fiber Telecommunication Networks: An Approach Based on Fuzzy Sets," *International Fuzzy*

Systems Association World Congress, 25-29 June 1997.

[15] C. M. F. Carlson, H. M. F. Tavares and J. R. F. Formigoni, "Parametric Mixed Linear Programming With Fuzzy Numbers and an Application to Telecommunications Networks Planning," *IEEE Telecommunications Symposium,* pp. 323-328, 9-13 August 1998.

[16] E. H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *International Journal of Man-Machine Studies,* vol. 7, no. 1, pp. 1-13, 1975.

[17] L. A. Zadeh, "Outline of a New Approach to the Analysis of a Complex Systems Decision Processes," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 3, no. 1, pp. 28-44, 1973.

[18] M. Sugeno e T. Takagi, “Multi-dimensional Fuzzy Reasoning,” *Fuzzy Sets and Systems,* vol. 9, n. 2, 1983.

[19] C. P. C. Marques, A. R. C. Siqueira, D. F. Pirani, T. Avansi, M. L. F. Abbade, A. A. Mota e L. T. M. Mota, “Identificação de Ofensores em Redes IEEE 802.11 Utilizando Lógica Nebulosa,” *XXXI Simpósio Brasileiro de Telecomunicações - SBrT,* Setembro 2013.

[20] K. Yaghmour, J. Masters, G. Ben-Yossef and P. Gerum, Building Embedded Linux Systems, O'Reilly, 2008.

[21] J. Corbet, A. Rubini and G. Kroah-Hartman, Linux Device Drivers, 3rd ed., O'Reilly, 2005.

[22] D. Kleidermacher and M. Kleidermacher, Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development, Newnes, 2012.

[23] L. Coetzee and J. Eksteen, "The Internet of Things - Promise for the Future? An Introduction," in *IST - Africa 2011*, 2011.

[24] P. P. Parikh, M. G. Kanabar e T. S. Sidhu, “Opportunities and Challenges of Wireless Communication Technologies for Smart Grid Applications,” *Power and Energy Society General Meeting,* pp. 1-7, 25-29 July 2010.

[25] OpenWRT, [Online]. Available: openwrt.org. [Acesso em 31 10 2014].

[26] Texas Instruments, [Online]. Available: www.ti.com. [Accessed 31 10 2014].

[27] Atmel, [Online]. Available: www.atmel.com. [Acesso em 31 10 2014].

[28] Scilab, [Online]. Available: www.scilab.org. [Acesso em 31 10 2014].

[29] Matlab, [Online]. Available: www.mathworks.com/products/matlab. [Acesso em 31 10 2014].

[30] Repositório online: embeddedFuzzy, [Online]. Available: https://sourceforge.net/projects/embeddedfuzzy/. [Acesso em 31 10 2014].

[31] Atmel: ATMEGA328P, [Online]. Available: http://www.atmel.com/devices/ATMEGA328P.aspx. [Accessed 31 10 2014].

[32] Arduino Nano, [Online]. Available: http://arduino.cc/en/Main/arduinoBoardNano. [Acesso em 31 10 2014].

[33] Arduino Reference - Float, [Online]. Available: http://arduino.cc/en/Reference/Float. [Acesso em 31 10 2014].

[34] TP-Link MR3020, [Online]. Available: http://www.tp-link.com.br/products/details/?model=TL-MR3020. [Acesso em 31 10 2014].

[35] Foto do TP-Link MR3020, [Online]. Available: http://www.tp-link.com.br/resources/images/products/large/TL-MR3020-V1-03.jpg. [Acesso em 31 10 2014].

[36] BuildRoot, [Online]. Available: buildroot.uclibc.org. [Acesso em 31 10 2014].

[37] OpenWRT: Cross Compile Guide, [Online]. Available: http://wiki.openwrt.org/doc/devel/crosscompile. [Acesso em 31 10 2014].

9. APÊNDICE

Apêndice 1

Matrizes de ofensividade
(Baseadas em [19])

Matriz R		
1	4	15
1	5	14
1	6	13
2	4	14
2	5	15
2	6	16
3	4	17
3	5	16
3	6	15
1	10	13
1	11	14
1	12	15
2	10	14
2	11	14
2	12	15
3	10	15
3	11	16
3	12	17
4	7	16
4	8	16
4	9	15
5	7	16
5	8	15
5	9	14
6	7	15
6	8	14
6	9	13
4	10	15
4	11	16
4	12	17
5	10	14
5	11	15
5	12	16
6	10	13
6	11	14
6	12	15
6	0	14

Matriz E	
0.3	3
41.73	3
11	3
38.36	3

Matriz C				
0.5	0.8	1	1	0
0.2	0.5	0.5	0.8	0
0	0	0.2	0.5	0
0	0	10	60	0
10	60	60	75	0
60	75	90	100	0
3	5	11	11	0
1	3	3	5	0
0	0	2	3	0
0.5	0.7	1	1	0
0.2	0.4	0.4	0.6	0
0	0	0.2	0.3	0
0.6	0.7	0.9	1	0
0.5	0.6	0.6	0.7	0
0.4	0.5	0.5	0.6	0
0.3	0.4	0.4	0.5	0
0	0.2	0.3	0.4	0

Apêndice 2

Makefile para o OpenWRT

```
#
# Cross Compile for OpenWRT Makefile
#
# Author: Tiago T. Ganselli
# November 2014

all: openwrt

openwrt:
	$(CROSS_COMPILER_PREFIX)gcc fuzzy.cpp main.cpp -o efuzzy.o #-Wall

x86:
	gcc fuzzy.cpp main.cpp -o efuzzy_x86.o #-Wall

clean:
	rm *.o
```

Apêndice 3

Implementação do
Método Matricial original para Scilab.
(Baseado em [5])

```
// ************************************************************************
//Processamento da logica Fuzzy -
//Entrada: Matriz e do arquivo Matriz.txt
// Variveis utilizadas: Matriz de conjuntos nebulosos C, origem:
// Arquivo Matriz C.txt
// Variveis utilizadas: Matriz de Regras nebulosas C, origem:
// Arquivo Matriz R.txt

clc
clear

//************************************************************************
//Declaração de funções:
//***********************************************
//        FUZZY                                  *
//***********************************************
// Onde A,B,C,D são dados das colunas da matriz C e o retorno "u" deve
// ser gravado na mesma linha de ABCD na 5.a coluna
// que é a pertinência

function [u] = fuzzy(A, B, C, D, xe)

    if(xe < A) | (xe >= D)
        u=0;  //fora da faixa
      end
      if(xe > A) & (xe < B)
          u=((xe-A)/(B-A));  // inclinação positiva
      end
      if(xe >= B) & (xe <= C)
           u=1;
      end
        if(xe > C) & (xe < D)  // inclinação negativa
           u=((D-xe)/(D-C));
      end
endfunction

//************************************************************************
// 1.a etapa: leitura das matrizes:
//************************************************************************
//DIR='.\m_tiago';
DIR='.\m_ana';

//Ler a Matriz e, entradas
e= fscanfMat(DIR+'\Matriz E.txt');
//Ler Matriz c, conjuntos nebulosos
c= fscanfMat(DIR+'\Matriz C.txt');
//Ler matriz r, regras nebulosas
r= fscanfMat(DIR+'\Matriz R.txt');


//************************************************************************
//2.a Etapa: Leitura e cálculo das perntinências na matriz c de acordo
// com dados da matriz de entradas
// A matriz de entrada tem duas colunas a primeira coluna corresponde
// aos valores das variáveis e a segunda a correspondencia das variáveis
// às alinhas da matriz C ex.:
//************************************************************************

//Rotina para preenchimento da Matriz 'C', valores de pertinências, com
// base nas informações da Matriz de entrada e
```

```
Tamanho = size(e);   // tamanho da matriz e, é um vetor com duas posições:
                //  Tamanho(1) = n.o de linhas e Tamanho(2)= n.o de colunas


CountLines = 1 ;   // contador de linhas na matriz C, para uma variável
ProxVariavel = 0;  // armazena a posição que parou na matriz C ao mudar
                   // de variável
LineC = 1 ;        // conta a linha corrente da Matriz C
//-----------------------------------------------------------------------
for i = 1 : Tamanho(1)       // Vai percorrer todas as linhas da matriz e
    NlinesemMC = e(i,2)  ;   // n.o de linhas que vai percorrer na Matriz C

    // incrementa a posição das variaveis na matriz C
    ProxVariavel= ProxVariavel + NlinesemMC ; // Se a primeira variavel
                             // for 5 linhas e a segunda 6,
                             // aqui já tenho a posição para ler a matriz
                             // C até a 11.a linha

    CountLines = LineC ;     // inicia na posição que é a validação da
                             // variável - Memoriza a posição que parou
                             // na matriz C

    for x=CountLines:ProxVariavel // da linha que parou ou iniciou até a
                                  // ultima linha da variável na Matriz C
        u=0;
        c(LineC, 5) = fuzzy(c(LineC,1),
                            c(LineC,2),
                            c(LineC,3),
                            c(LineC,4),
                            e(i,1)) ;  // Valores: A,B,C,D,xe
        //printf('Valor da pertinencia    %.1f\n',c(LineC,5));
        LineC = LineC + 1 ;

    end

end

//***********************************************************************
// 3. Implicação das regras e agregação dos consequentes
//Iniciar processo de tratamento da matriz R: ela contém três colunas
// antecedente e consequente e uma terceira coluna
// que indica a linha da Matriz C a ser atualizada. Então dada uma
// pertinencia em C
//***********************************************************************
// Vai rodar a matriz R
for i= 1: size(r,1)
    if r(i,2) ~= 0  then  // diferente de zero tem dois argumentos
      c(r(i,3),5) = max( c(r(i,3),5) , min(c(r(i,1),5)  ,c(r(i,2),5)  )  );
   else
      c(r(i,3),5) = max( c(r(i,3),5) , c(r(i,1),5)  );
   end
end

disp(c);

//***********************************************************************
// 4. Deffuzificação
// Dada a matriz C, com os pontos de corte calculados , montar o cálculo
// da área com passos de 0,1
// usando algoritimo do centroide.
```

```
// trabalhar com a últimas 5 linhas da matriz C, analisando os resultados
// de alfa cut calculado
//*********************************************************************

// Dimensionar os passos do eixo x, criar um vetor

xi = 0:(1-0)/10:1;

// Criei um vetor com 11 posições em passos de 0.1
// variáveis úteis no processo

ui_somat=0;   // somatoria das pertinências
ui_xi_somat = 0 ;// somat ui * xi
centroide = 0.0;

Vet_ui = zeros(11);
Vet_xi = zeros(11);
ui =0.0;

// *********************************************************************
// Início do processamento - Cálculo do centróide baseado na área da
// envoltória gerada pelo alfa cut (corte)
//*********************************************************************
// cuidado o vetor começa com 1, diferente da linguagem C

for i=1:size(xi,2)    // varre todos os xi´s em passos de 0.1
     //printf("-----> i %f\n", i)
       ui=0.0;
     ui_xi=0.0;

     for j=size(c,1):-1:size(c,1)-5
         //printf("-----> J %f\n", j)
         //printf("-----> xi %f\n", xi(1,i))
         u=fuzzy(c(j,1),c(j,2),c(j,3),c(j,4),xi(1,i)) ;

        ui =max( (  min(  u, c(j,5)   )  ) , ui   ) ;

        Vet_ui (1,i)=ui;
        Vet_xi(i) = ui*xi(1,i);

     end

    // *********************************************************************
    // somatória do numerador e denominador do cálculo do centroide
    //*********************************************************************
     ui_somat = ui_somat + ui;
     ui_xi_somat = ui_xi_somat + (ui * xi(1,i));
     //printf("-----> ui somatoria %f\n", ui_somat)
     //printf("-----> ui_xi_somat %f\n", ui_xi_somat)

end
centroide= ui_xi_somat / ui_somat;
printf("-----> centroide somatoria %f\n", centroide);
```

Apêndice 4

Implementação do Método Matricial para Scilab

```
////////////////////////////////////////////////////////////////////
// Principios Conceitos e Metodologia de Gestao                   //
// Pos-graduacao PUC-Campinas - 2° Semestre de 2013               //
//                                                                //
// Tiago T. Ganselli                                              //
// Daniel B. Barros                                               //
//                                                                //
// @Descrição:                                                    //
//  Este é o arquivo principal, que deve ser executado pelo       //
//  usuário para a fuzzyficação das matrizes.                     //
//                                                                //
// @Instruções de uso:                                            //
//  1 - Para utilizar este programa as variáveis de configuração  //
//  devem ser alteradas. Essas variáveis estão presentes abaixo   //
//  com suas devidas explicações.                                 //
//  2 -
//
// @ToDo:
//  - Verificar se a matriz R possui algum valor 0 ou tratar estas
//  ocorrências ao aplicar as regras.
//  - Tipos e tempo de execução no Arduino:
//  https://learn.sparkfun.com/tutorials/data-types-in-arduino
////////////////////////////////////////////////////////////////////

//Diretório no qual o arquivo está localizado:
//local_do_programa = 'C:\Users\Tiago\Google
Drive\PUC\Mestrado\dissertacao\analise_nebulosa\fuzzy';

//Nome da pasta na qual estão as matrizes:
//pasta_das_matrizes = 'matrizes_por_aulas_calibra';
//pasta_das_matrizes = 'matrizes_por_aulas_A';

//Numero de pontos para discretização dos valores do eixo X.
n_pontos = 10;

//Habilita ou desabilita a impressão de informações no console do SciLab
imprimir_info  = 1;
imprimir_debug = 1;

//Propriedades do gráfico:
plotar_grafico  = 0;    //Habilita a geração de gráficos
plotar_crisp    = 0;    //Habilita os valores crisp mostrados nos gráficos
plotar_separado = 0;    //Plota os gráficos em janelas separadas (ignora as
configurações de subplot abaixo)
subplot_lines   = 3;    //Indica quantos gráficos serão desenhados na
vertical
subplot_columns = 2;    //Indica quantos gráficos serão desenhados na
horizontal

//Os labels precisam estar na mesma quantidade de linhas de entrada da
matriz E + o nome do gráfico de saída.
xlabel_graf = [ "Obsolescência do equipamento";
                "Potência do Sinal Recebido [dBm]";
                "Taxa nominal negociada [MB/s]";
                "Taxa efetivamente praticada [%]";
                "Potencial Ofensivo"
              ];

////////////////////////////////////////////////////////////////////
//
```

```
// O programa começa aqui.
//
////////////////////////////////////////////////////////////////////------
clc();
printf("Fuzzyficando...\n");

//Altera o diretório atual do SciLab para o local do programa no PC.
//chdir(local_do_programa+'\'+pasta_das_matrizes);

//Importa as funções presentes no arquivo separado
exec('../funcoes.sce');

//Importa as matrizes e calcula a quantidade de linhas e colunas.
[C_matriz] = fscanfMat("matriz_C.txt");
[C_linhas_n, C_colunas_n] = size(C_matriz);
[E_matriz] = fscanfMat("matriz_E.txt");
[E_linhas_n, E_colunas_n] = size(E_matriz);
[R_matriz] = fscanfMat("matriz_R.txt");
[R_linhas_n, R_colunas_n] = size(R_matriz);

//Verificação de integridade das matrizes:
if C_colunas_n ~= 5 then
  disp('A matriz C deve possuir 5 colunas.');
  disp('Preencher a 5ª coluna com o valor -1.');
  return
end
if E_colunas_n ~= 2 then
  disp('A matriz E deve possuir 2 colunas.');
  return
end
if R_colunas_n ~= 3 then
  disp('A matriz R deve possuir 3 colunas.');
  return
end
if (size(xlabel_graf, 1)) ~= (E_linhas_n+1) then
  disp('A nomenclatura do gráfico está errada na variável: xlabel_graf.');
  disp('Ela deve possuir entradas correspondentes a todas as linhas da
matriz E mais uma para o nome da saída do sistema.');
  disp('Ou seja, deve possuir um numero de frases igual a: E_linhas_n+1');
  return
end

//Calculo do numero de linhas de entrada do sistema
linhas_de_entrada = 0;
for i = 1:E_linhas_n
    linhas_de_entrada = linhas_de_entrada + E_matriz(i,2);
end

//Calculo do numero de linhas de saida do sistema
linhas_de_saida = C_linhas_n-linhas_de_entrada;

if imprimir_debug then
    printf("\nValores de entrada:\n");
    for i = 1:E_linhas_n
        printf("\t%f\n",E_matriz(i,1));
    end
end

////////////////////////
// Pertinencia        //
```

```
//////////////////////

//Vamos calcular a pertinência e adicioná-la à matriz C.
//Para isso é feito um loop que rodará tantas vezes quanto indicado na
// segunda coluna da matriz E.
//Para cada linha da matriz C será executada a função fuzzy() com os seus
// valores e o resultado será salvo na posição de 'u'.

//printf("Fuzzyficando a matriz C...\n");
//Inicializa o offset de linhas para preenchimento da matriz C
offset_linha_C = 0;
//Percorre todas as linhas da matriz E
for curr_linha_E=1:E_linhas_n
    //Preenche as linhas da matriz C que estão indicadas na linha atual da
matriz E
    for i = 1:E_matriz(curr_linha_E,2)
        crisp = E_matriz(curr_linha_E,1);
        a = C_matriz(i+offset_linha_C,1);
        b = C_matriz(i+offset_linha_C,2);
        c = C_matriz(i+offset_linha_C,3);
        d = C_matriz(i+offset_linha_C,4);
        C_matriz(i+offset_linha_C,5) = fuzzy(crisp, a, b, c, d);
    end
    //Adiciona o offset de linhas da matriz C.
    //É a soma dos valores da coluna 2 da matriz E até a linha atual.
    offset_linha_C = offset_linha_C + E_matriz(curr_linha_E,2);
end

//////////////////////
// Regras           //
//////////////////////

//Uso da estrutura de regras para preenchimento do valor u:
//printf("Aplicando estrutura de regras para %d linhas\n", R_linhas_n);
for i = 1:R_linhas_n //Executa tantas vezes quanto o numero de
                     // linhas da matriz R.

//    printf("Linha: %d\n", i);

    if (R_matriz(i,2) > 0) then
        //Aplicação normal da regra
        A1_valor = C_matriz(R_matriz(i,1),5);
        A2_valor = C_matriz(R_matriz(i,2),5);
        Cn_valor = C_matriz(R_matriz(i,3),5);

        //Coloca na matriz C o valor resultante da regra aplicada.
        resultado_regra = aplicaRegra( A1_valor , A2_valor , Cn_valor );
    elseif (R_matriz(i,2) == 0)
        //Aplicação da regra quando a segunda regra é ZERO
        A1_valor = C_matriz(R_matriz(i,1),5);
        Cn_valor = C_matriz(R_matriz(i,3),5);

        //Coloca na matriz C o valor resultante da regra aplicada.
        resultado_regra = max(A1_valor, Cn_valor);
    else
        disp("=====================================");
        disp("ERRO: Matriz de regras com N° negativo");
        disp("=====================================");
        return;
    end
```

```
    if imprimir_debug then
        if resultado_regra > C_matriz(R_matriz(i,3),5) then
            printf("Regra %d Implicada!\n", i);
        else
            //printf("\n");
        end
    end

    C_matriz(R_matriz(i,3),5) = resultado_regra;

end

if imprimir_info then
    printf("\nMatriz C completa:\n");
    disp(C_matriz);
end

//Imprime os cortes alfa para todas as linhas de saída do sistema
//if imprimir_info then
//    printf("\n");
//    for i = (linhas_de_entrada+1) : C_linhas_n
//        printf("Corte Alfa %d = %f\n",(i-linhas_de_entrada),
C_matriz(i,5));
//    end
//end

//////////////////////////
// Defuzzyficação     //
//////////////////////////

//Calculo do maior valor possivel de X
//Esse valor e o maior entre os valores da coluna 'd' da matriz C para
//  as linhas referentes à saída do sistema.
maior_valor_de_X = 0;
for i = linhas_de_entrada : C_linhas_n
    maior_valor_de_X = max(maior_valor_de_X, C_matriz(i,4));
end

//Calculo do passo
passo = maior_valor_de_X/n_pontos;

//Cria um vetor com os valores de Xn
Xn = [0:passo:maior_valor_de_X];
//Cria um vetor para ser preenchido com os valores de un calculados
un = [0:passo:maior_valor_de_X];

//Fazer o calculo em todos os pontos do eixo X
for i = 1:n_pontos+1
    //Inicializa a variável para o corte máximo
    corte_max = 0;
    //Iteração ocorre por todas as linhas de saída do sistema
    //  para que seja encontrado o valor mínimo entre o CorteAlfa
    //  e o valor fuzzyficado no dado ponto.
    //  É escolhido o maior valor entre os encontrados
    //  para cada linha de saída do sistema.
    for j = (linhas_de_entrada+1) : C_linhas_n
        //Calcula o mínimo entre o CorteAlfa e o valor da
        //  pertinência no ponto em questão.
        a = C_matriz(j,1);
```

```
        b = C_matriz(j,2);
        c = C_matriz(j,3);
        d = C_matriz(j,4);
        corte_j = min(C_matriz(j,5), fuzzy(Xn(i), a, b, c, d) );
        //Calcular o máximo entre o valor calculado e o anterior.
        //O resultado é o máximo entre os valores calculados para
        //  as linhas de saída do sistema.
        corte_max = max(corte_j, corte_max);
    end

    //Grava em un(i) o valor final
    un(i) = corte_max;
end

//////////////////////////
// Centróide            //
//////////////////////////

//O centróide é calculado como uma média ponderada dos pontos:
// result = Somatória( Xn(i)*un(i) ) / Somatória( un(i) )
// Com i variando de 0 até o valor máximo estabelecido por n_pontos
num = 0;
den = 0;
for i = 1:n_pontos+1
    //O numerador é a somatória de Xn(i) * un(i)
    num = num+Xn(i)*un(i);
    //O denominador é a somatória de un(i)
    den = den+un(i);
end
result = num/den;

//Imprime o valor final calculado
if imprimir_info then
printf("\n**************************\n");
printf("Valor CRISP final: %f", result);
printf("\n**************************\n");
end

//Exporta o valor final para arquivo
if imprimir_info then
    printf("Exportando resultados para arquivo...\n");
end
fprintfMat("result.txt", result);

//////////////////////////
// Gráficos             //
//////////////////////////
if plotar_grafico then
    printf("Imprimindo os gráficos...\n");
    //Limpa a tela de gráficos para não ocorrer a sobreposição.
    clf();
    //Cria a janela de gráficos:
    scf(0);

    //Imprime as matrizes de entrada do sistema
    offset = 0;
    offset_maior_valor_x = 0;

    for curr_plot=1 : (E_linhas_n)
      //Indica o subplot atual
```

```
  if plotar_separado then
    //plot(subplot_lines,subplot_columns,curr_plot)
  else
    subplot(subplot_lines,subplot_columns,curr_plot)
  end

  //Cálculo do maior e menor valores do eixo X para entradas do sistema
  menor_valor_x = C_matriz( (offset+1), 1);
  maior_valor_x = C_matriz( (offset+E_matriz(curr_plot,2)), 4);

  //Pontos dummy para que as escalas fiquem bonitinhas.
  //plot( (menor_valor_x-1) , 1.2);
  //plot( (maior_valor_x+1) , 1.2);

  //Imprime os trapézios do grupo atual
  for i=(1+offset) : (E_matriz(curr_plot,2)+offset)
    plotMatrixLine( C_matriz(i,1), ..
                    C_matriz(i,2), ..
                    C_matriz(i,3), ..
                    C_matriz(i,4) );
    if plotar_crisp then
        plotDotLine(0,
                    C_matriz(i,5),
                    E_matriz(curr_plot,1),
                    C_matriz(i,5));
        //Imprime o texto indicando o valor final
        xstring(0, C_matriz(i,5), string(C_matriz(i,5)));
    end
  end

  //Atualiza o offset para a próxima iteração
  offset = offset+E_matriz(curr_plot,2);
  //Imprime reta vertical indicando o valor CRISP
  if plotar_crisp then
      plotLine(E_matriz(curr_plot,1), 0, E_matriz(curr_plot,1), 1.2);
      //Imprime o texto indicando o valor final
      xstring(E_matriz(curr_plot,1), 1, string(E_matriz(curr_plot,1)));
  end

  //Nomeia o gráfico e os eixos. Hardcoded no início do programa.
  xlabel(xlabel_graf(curr_plot,1),"fontsize",4);
  ylabel("Pertinência","fontsize",4);

  if plotar_separado then
    scf(curr_plot);
  end
end

//Imprime as variáveis de saida do sistema
//Atualiza o plot atual
curr_plot = curr_plot+1;

//Indica o subplot atual
if plotar_separado then
    //plot(subplot_lines,subplot_columns,curr_plot)
else
    subplot(subplot_lines,subplot_columns,curr_plot)
end

//Cálculo do maior e menor valores do eixo X para a saída do sistema
```

```
    menor_valor_x = C_matriz( (offset+1), 1);
    maior_valor_x = C_matriz( (offset+linhas_de_saida), 4);

    //Pontos dummy para que as escalas fiquem bonitinhas.
    plot( (menor_valor_x-1) , 1.2);
    plot( (maior_valor_x+1) , 1.2);

    //Imprime os trapézios do grupo atual
    for i=(1+offset) : (linhas_de_saida+offset)
      plotMatrixLine( C_matriz(i,1), ..
                      C_matriz(i,2), ..
                      C_matriz(i,3), ..
                      C_matriz(i,4) );
    end

    if plotar_crisp then
        //Imprime reta vertical indicando o valor CRISP
        plotLine(result, 0, result, 1.2);
        //Imprime o texto indicando o valor final
        xstring(result, 1, string(result));
    end

    //Coloca o label no gráfico
    //xtitle("Ocupação Nebulosa");
    xlabel(xlabel_graf(curr_plot,1),"fontsize",4);
    ylabel("Pertinência","fontsize",4);

    //Plota o gráfico da pertinência sobre o de saída.
    plot(Xn,un,'b');
end
```

```
////////////////////////////////////////////////////////////////////
// Principios Conceitos e Metodologia de Gestao                   //
// Pos-graduacao PUC-Campinas - 2° Semestre de 2013               //
//                                                                //
// Tiago T. Ganselli                                              //
// Daniel B. Barros                                               //
//                                                                //
// Descrição: Este arquivo contém as funções utilizadas pelo      //
//            programa de fuzzyficação.                           //
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
//Função fuzzy()
//    Entradas:
//    - crisp. Valor real da variável.
//    - a, b, c e d. Valores modais da função. São os pontos do
//                   eixo X que formam o gráfico.
//    Retorna:
//    - u. Valor de entrada fuzzyficado no intervalo de 0 a 1.
////////////////////////////////////////////////////////////////////
//funcprot(0);
function u=fuzzy( crisp, a, b, c, d)
    // Teste de borda. Por definição, deve-se utilizar a
    //maior pertinência quando dois valores crisp são iguais.
    // Os 'if' são independentes (separados) para que o maior
    //valor (1) seja escolhido caso o valor CRISP seja de borda.
    // Exemplo:
    // Se o valor CRISP for igual a 'a' e 'b' simultaneamente, 'u'
    //receberá o valor 0 e, em seguida, o valor 1.
    if crisp==a | crisp==d  then
        u = 0;
    end
    if crisp==b | crisp==c  then
        u = 1; //max
    end

    // Fuzzyficação simples.
    // Trata os valores pertencentes aos grupos que não são de borda.
    if crisp<a then
        //disp("Crisp menor que A.\n");
        u = 0;
    elseif crisp<b then
        //disp("Crisp entre A e B.\n");
        u = (crisp-a)/(b-a)
    elseif crisp<c then
        //disp("Crisp entre B e C.\n");
        u = 1;
    elseif crisp<d then
        //disp("Crisp entre C e D.\n");
        u = (d-crisp)/(d-c);
    elseif crisp>d then
        //disp("Crisp maior que D.\n");
        u = 0;
    end
endfunction

////////////////////////////////////////////////////////////////////
//Função aplicaRegra()
//    Entradas:
//    - A1, A2, Cn. Valores de u da matriz C, obtidos através dos
```

```
//                  número da respectiva linha na matriz R.
//    Retorna:
//    - u. Valor a ser colocado na posição de Cn, que está indicada
//         na matriz R.
////////////////////////////////////////////////////////////////////////
//funcprot(0);
function uu=aplicaRegra( A1, A2, Cu)
    uu = min(A1, A2);
    uu = max(uu, Cu);
endfunction

////////////////////////////////////////////////////////////////////////
//Função plotMatrixLine()
//    Entradas:
////////////////////////////////////////////////////////////////////////
//funcprot(0);
function plotMatrixLine(a, b, c, d)
    xpts = [a b];
    ypts = [0 1];
    plot(xpts,ypts, 'k--');

    xpts = [b c];
    ypts = [1 1];
    plot(xpts,ypts, 'k--');

    xpts = [c d];
    ypts = [1 0];
    plot(xpts,ypts, 'k--');
endfunction

////////////////////////////////////////////////////////////////////////
//Função plotLine()
//    Entradas:
////////////////////////////////////////////////////////////////////////
//funcprot(0);
function plotLine(x1, y1, x2, y2)
    xpts = [x1 x2];
    ypts = [y1 y2];
    plot(xpts,ypts,'r');

    //Não consegui passar o tipo por parâmetro =(
    //type = "r--";
    //plot(xpts,ypts,type);
endfunction

////////////////////////////////////////////////////////////////////////
//Função plotLine()
//    Entradas:
////////////////////////////////////////////////////////////////////////
//funcprot(0);
function plotDotLine(x1, y1, x2, y2)
    xpts = [x1 x2];
    ypts = [y1 y2];
    plot(xpts,ypts,'g--');

    //Não consegui passar o tipo por parâmetro =(
    //type = "r--";
    //plot(xpts,ypts,type);
endfunction
```

Apêndice 5

Biblioteca Arduino embeddedFuzzy

```
/*
 * Embedded Fuzzy library for arduino.
 * Tiago Ganselli
 *
 * Work done as prerequisite to the Masters Degree
 * in management of telecommunications networks.
 *
 * PUC-Campinas 2014
 */

#include <Arduino.h>
#include "embeddedfuzzy.h"

//=============================================================
// Global Variables
//=============================================================

eFuzzyClass eFuzzy;

//=============================================================
// Public Functions
//=============================================================

eFuzzyClass::eFuzzyClass() {

    //
    // Itilialize system internal variables
    //
    total_input_lines = 0;
    total_output_lines = 0;
    memset(Xn, 0, sizeof(Xn));
}

eFuzzyClass::~eFuzzyClass() {
}

/**
 * @brief Initialize the system
 * @param       None
 * @return      0 if success.
 *              -1 if error.
 */
int eFuzzyClass::configure( void* mC_, int mC_ln,
                            void* mR_, int mR_ln,
                            void* mE_, int mE_ln){
    PINFO("Configuring...");

    int ret = 0;

    //
    // check matrix integrity
    //
//    ret = doCheckIntegrity();
//    if ( ret ) {
//        PERROR("in doCheckIntegrity()");
//        PERROR(ret);
//        return ret;
//    }

    //
```

```
    //Save variables in the system
    //
    mC[0] = (mC_entry_t*)mC_;
    mR[0] = (mR_entry_t*)mR_;
    mE[0] = (mE_entry_t*)mE_;

    mC_lines_n = mC_ln;
    mR_lines_n = mR_ln;
    mE_lines_n = mE_ln;

    //
    //Calculate the number of input and output lines in the system
    //
    for(int i=0; i<mE_lines_n; i++) {
        total_input_lines = total_input_lines + mE[i]->index;
    }
    total_output_lines = mC_lines_n - total_input_lines;

    PINFO("Configured");

    return 0;
}

/**
 * @brief Run the fuzzy system over the data
 * @param       None
 * @return      0 if success. -1 if error.
 */
int eFuzzyClass::run(fuzzySet_t* result) {
    PINFO("Running...");

    //
    // fuzzy - calculate mC pertinence
    //
    doCalculatePertinence();

    //
    // apply rules
    //
    doApplyRules();

    //
    // defuzzy
    //
    doDefuzzy();

    //
    // find centroid
    //
    *result = doFindCentroid();

    PINFO("Success");
    return 0;
}

/**
 * @brief
 * @param
 * @return
 */
```

```
void eFuzzyClass::printMatrixC() {

    for (int i=0; i<mC_lines_n; i++) {
        Serial.printf("\t%d\t%d\t%d\t%d\t", (int)mC[i]->a,\
                                            (int)mC[i]->b,\
                                            (int)mC[i]->c,\
                                            (int)mC[i]->d );
        Serial.println( mC[i]->pertinence );
    }

}

//=========================================================
// Private Functions
//=========================================================

/**
 * @brief       Calculate pertinence of the crisp input value.
 * @param[in]   a, b, c, d. Modal values.
 * @param[in]   crisp. Input CRISP value.
 * @param[out]  u. The calculated pertinence.
 * @return      0 if success. -1 if error.
 */
int eFuzzyClass::fuzzy( fuzzySet_t a,
                        fuzzySet_t b,
                        fuzzySet_t c,
                        fuzzySet_t d,
                        fuzzySet_t crisp,
                        fuzzySet_t* u ) {
    //
    // Edge test
    //  By definition, when two crisp values are the same
    //  the highest pertinence must be used.
    //  Ifs are independet to result in 1 in the case of
    //  an edge crisp.
    //
    if ( crisp == a || crisp == d ) {
        *u = 0;
    }
    if ( crisp == b || crisp == c ) {
        *u = 1;
    }

    //
    // Simple Fuzzyfication
    //  Deal with the non-edge values
    //
    if ( crisp < a ) {          //crisp is smaller than a
        *u = 0;
    } else if ( crisp < b ) {   //crisp between a and b
        *u = (crisp-a)/(b-a);
    } else if ( crisp < c ) {   //crisp between b and c
        *u = 1;
    } else if ( crisp < d ) {   //crisp between c and d
        *u = (d-crisp)/(d-c);
    } else if ( crisp > d ) {   //crisp greater than d
        *u = 0;
    } else {
        //Should never get here
        return -1;
```

```
    }

    return 0;
}

/**
 * @brief       Apply rule over input data.
 * @param[in]   A1, A2. Input pertinence values from mC.
 * @param[inout]Cu. Target output pertinence value from mC.
 * @return      1 if rule was implyed. 0 if not.
 */
int eFuzzyClass::applyRule( fuzzySet_t A1,
                            fuzzySet_t A2,
                            fuzzySet_t* Cu ) {
    fuzzySet_t u;

    u = min(A1, A2);

    // max(u,*Cu)
    if ( u > *Cu ) {
        //Rule was implyed
        *Cu = u;
        return 1;
    }

    //Rule was not implyed
    return 0;
}

/**
 * @brief
 * @param
 * @return
 */
int eFuzzyClass::doCheckIntegrity() {
    return 0;
}

/**
 * @brief   Extract some useful info from the input matrix set.
 * @param   None.
 * @return  0 if success. -1 if error.
 */
int eFuzzyClass::doExtractUtilInfo() {

    //
    //Calculate the number of lines for each input matrix
    //
    mC_lines_n = ( sizeof(mC)/sizeof(mC_entry_t) );
    mR_lines_n = ( sizeof(mR)/sizeof(mR_entry_t) );
    mE_lines_n = ( sizeof(mE)/sizeof(mE_entry_t) );

    //
    //Calculate the number of input and output lines in the system
    //
    for(int i=0; i<mE_lines_n; i++) {
        total_input_lines = total_input_lines + mE[i]->index;
    }

    total_output_lines = mC_lines_n - total_input_lines;
```

```
    return 0;
}

/**
 * @brief   Calculate pertinence for each line of matrix C.
 * @param   None.
 * @return  0 if success. -1 if error.
 */
int eFuzzyClass::doCalculatePertinence() {
    PINFO("Running Fuzzy on matrix C...");

    int mC_offset = 0;

    //Run for each line of matrix E
    for(int mE_curr_line=0; mE_curr_line<mE_lines_n; mE_curr_line++) {
        //Fill matrix C lines that are indicated in current matrix E line
        for(int i=0; i<mE[mE_curr_line]->index; i++) {
            fuzzy(mC[i+mC_offset]->a,\
                  mC[i+mC_offset]->b,\
                  mC[i+mC_offset]->c,\
                  mC[i+mC_offset]->d,\
                  mE[mE_curr_line]->value,\
                  &mC[i+mC_offset]->pertinence );
        }
        //Update matrix C offset:
        // It is the sum of mE index to the current line.
        mC_offset = mC_offset + mE[mE_curr_line]->index;
    }

    return 0;
}

/**
 * @brief   Apply rules and populate output pertinences on mC.
 * @param   None.
 * @return  0 if success. -1 if error.
 */
int eFuzzyClass::doApplyRules() {
    PINFO("Applying rules...");

    // -1 (and +1) are there because rules
    // begin in 1, but here they begin in 0.

    for(int i=0; i<mR_lines_n; i++) {
        if ( mR[i]->precedenceB > 0 ) {      //Normal rule application
            if ( applyRule( mC[(mR[i]->precedenceA-1)]->pertinence,\
                            mC[(mR[i]->precedenceB-1)]->pertinence,\
                            &mC[(mR[i]->consequence-1)]->pertinence ) ) {
                PPRINTF("Rule %d implyed\n", i+1);
            }
        }
        else if (mR[i]->precedenceB == 0 ) { //Second rule is zero
            PINFO("Second rule is zero.");
            //@todo: Test this
            mC[(mR[i]->consequence-1)]->pertinence = \
                    max(mC[(mR[i]->precedenceA-1)]->pertinence,\
                        mC[(mR[i]->consequence-1)]->pertinence);
        }
        else {                                //Error. Should not happen.
```

```
            PERROR("precedenceB is negative.");
            return -1;
        }
    }

    return 0;
}

/**
 * @brief
 * @param
 * @return
 */
int eFuzzyClass::doDefuzzy() {
    PINFO("Running DeFuzzy on matrix C...");

    fuzzySet_t greater_x_value = 0;
    fuzzySet_t step = 0;

    //
    // Find the greatest value from input matrix
    //
    for ( int i=total_input_lines; i<mC_lines_n; i++) {
        greater_x_value = max( greater_x_value, mC[i]->a );
        greater_x_value = max( greater_x_value, mC[i]->b );
        greater_x_value = max( greater_x_value, mC[i]->c );
        greater_x_value = max( greater_x_value, mC[i]->d );
    }

    //
    // Calculate the step necessary to fill all points and fill buffer
    //
    step = greater_x_value/N_POINTS;

    Xn[0].x = 0;                                //Init first element

    for ( int i=1 ; i<(N_POINTS+1); i++) {  //Fill other elements
        Xn[i].x = Xn[i-1].x+step;
    }

    //
    // Calculate each point in Xn vector
    //
    for ( int i=0; i<(N_POINTS+1); i++) {
        fuzzySet_t max_cut;
        max_cut = 0;

        //Iteration runs for every output line of the system
        // in order to find the lowest value between the
        // Alfa Cut and the fuzzy value on a given point.
        // The greatest value is choosen for each output line
        for ( int j=total_input_lines; j<mC_lines_n; j++) {
            //Calculate the minimum between Alfa Cut and the
            // point's pertinence
            fuzzySet_t jCut;
            fuzzySet_t ju;

            fuzzy(mC[j]->a, mC[j]->b, mC[j]->c, mC[j]->d, Xn[i].x, &ju);

            jCut = min( mC[j]->pertinence, ju);
```

```
            //Find max between the calculated value and the
            // previous value. The result is the max calculated
            // value for the output lines
            max_cut = max(jCut, max_cut);
        }
        //Store final value for this point
        Xn[i].u = max_cut;
    }

    return 0;
}

/**
 * @brief
 * @param
 * @return
 */
fuzzySet_t eFuzzyClass::doFindCentroid() {
    //The centroid method is the weighted average of the points:
    // result = sum( Xn[i].x*Xn[i].u )/( sum(Xn[i].u) )
    // with i in the range from 0 to the max value of N_POINTS
    fuzzySet_t numerator = 0;
    fuzzySet_t denominator = 0;
    fuzzySet_t result = 0;

    for ( int i=0; i<N_POINTS+1; i++) {
        numerator = numerator + ( Xn[i].x * Xn[i].u );
        denominator = denominator + Xn[i].u;
    }

    //Carefull with numerator or denominator overflow...
    result = numerator/denominator;

    return result;
}
```

```
/*
 * Embedded Fuzzy library for arduino.
 * Tiago Ganselli
 *
 * Work done as prerequisite to the Masters Degree
 * in management of telecommunications networks.
 *
 * PUC-Campinas 2014
 */

#ifndef _EFUZZY_H
#define _EFUZZY_H

#include <stdio.h>

//============================================================
// Configuration
//============================================================
//#define PRINT_INFO
//#define PRINT_DEBUG
//#define PRINT_ERROR

#define N_POINTS        100

//============================================================
// Types
//============================================================
typedef float fuzzySet_t;

typedef struct {
    fuzzySet_t x;
    fuzzySet_t u;
}uVector_t;

struct mC_entry_t {
    fuzzySet_t a;
    fuzzySet_t b;
    fuzzySet_t c;
    fuzzySet_t d;
    fuzzySet_t pertinence;
};

struct mR_entry_t {
    int precedenceA;
    int precedenceB;
    int consequence;
};

struct mE_entry_t {
    fuzzySet_t value;
    fuzzySet_t index;
};

//============================================================
// Macros
//============================================================
#ifdef PRINT_INFO
    #define PINFO(txt)          \
    do{                         \
        Serial.write(__func__); \
```

```
        Serial.write(": ");     \
        Serial.write(txt);      \
        Serial.write('\n');     \
    }while(0);

    //@todo: use vargs
    #define PPRINTF(txt, val)   \
    do{                         \
        Serial.printf(txt, val);\
    }while(0);

    #define PINFOVAL(txt, val)  \
    do{                         \
        Serial.write(__func__); \
        Serial.write(": ");     \
        Serial.write(txt);      \
        Serial.write(" ");      \
        Serial.print(val);      \
        Serial.write('\n');     \
    }while(0);

    #define PINFOLN(txt, val)   \
    do{                         \
        Serial.write(__func__); \
        Serial.write(": ");     \
        Serial.write(txt);      \
        Serial.println(val, 3); \
    }while(0);
#else
    #define PINFO(txt)
    #define PPRINTF(txt, val)
    #define PINFOVAL(txt, val)
    #define PINFOLN(txt, val)
#endif

#ifdef PRINT ERROR
    #define PERROR(txt)         \
    do{                         \
        Serial.write(__func__); \
        Serial.write(": ");     \
        Serial.write("!ERROR ");\
        Serial.write(txt);      \
        Serial.write('\n');     \
    }while(0);
#else
    #define PERROR(txt)
#endif

#define max(a,b)                \
({                              \
    __typeof__ (a) _a = (a);    \
    __typeof__ (b) _b = (b);    \
    _a > _b ? _a : _b;          \
})

#define min(a,b)                \
({                              \
    __typeof__ (a) _a = (a);    \
    __typeof__ (b) _b = (b);    \
    _a < _b ? _a : _b;          \
```

```
})

//=========================================================
// Variables
//=========================================================
static uVector_t Xn[N_POINTS+1];

//=========================================================
// Classes
//=========================================================
/** Classe Embedded Fuzzy */
class eFuzzyClass {

    public:
        eFuzzyClass();
       ~eFuzzyClass();

        int configure(void* mC_, int mC_ln,
                      void* mR_, int mR_ln,
                      void* mE_, int mE_ln);
        int run(fuzzySet_t* result);
        void printMatrixC(void);

    private:
        mC_entry_t* mC[17];
        mR_entry_t* mR[37];
        mE_entry_t* mE[4];

        int mC_lines_n;
        int mR_lines_n;
        int mE_lines_n;

        int total_input_lines;
        int total_output_lines;

        int fuzzy( fuzzySet_t a,
                   fuzzySet_t b,
                   fuzzySet_t c,
                   fuzzySet_t d,
                   fuzzySet_t crisp,
                   fuzzySet_t* u );
        int applyRule( fuzzySet_t A1,
                       fuzzySet_t A2,
                       fuzzySet_t* Cu );
        int doCheckIntegrity(void);
        int doExtractUtilInfo(void);
        int doCalculatePertinence(void);
        int doApplyRules(void);
        int doDefuzzy(void);
        fuzzySet_t doFindCentroid(void);
};

extern eFuzzyClass eFuzzy;

#endif // _EFUZZY_H
```

Apêndice 6

Aplicação OpenWRT eFuzzy

```
/*
 * eFuzzy library for OpenWRT.
 * Tiago Ganselli
 *
 * Work done as prerequisite to the Masters Degree
 * in management of telecommunications networks.
 *
 * PUC-Campinas 2014
 */

#include "fuzzy.h"
#include <string.h>

//=========================================================
// Global Variables
//=========================================================

eFuzzyClass eFuzzy;

//=========================================================
// Public Functions
//=========================================================

eFuzzyClass::eFuzzyClass() {

    //
    // Itilialize system internal variables
    //
    total_input_lines = 0;
    total_output_lines = 0;
    memset(Xn, 0, sizeof(Xn));
}

eFuzzyClass::~eFuzzyClass() {
}

/**
 * @brief Initialize the system
 * @param       None
 * @return      0 if success.
 *              -1 if error.
 */
int eFuzzyClass::configure() {
    PINFO("Configuring...");

    //@todo: Dinamicaly allocate these buffers

    mC[0].a= 0   ; mC[0].b= 0   ; mC[0].c= 20  ; mC[0].d= 50  ;
mC[0].pertinence=-1;
    mC[1].a= 20  ; mC[1].b= 50  ; mC[1].c= 50  ; mC[1].d= 80  ;
mC[1].pertinence=-1;
    mC[2].a= 50  ; mC[2].b= 80  ; mC[2].c= 100 ; mC[2].d= 100 ;
mC[2].pertinence=-1;
    mC[3].a= -100; mC[3].b= -90 ; mC[3].c= -75 ; mC[3].d= -60 ;
mC[3].pertinence=-1;
    mC[4].a= -75 ; mC[4].b= -60 ; mC[4].c= -60 ; mC[4].d= -10 ;
mC[4].pertinence=-1;
    mC[5].a= -60 ; mC[5].b= -10 ; mC[5].c= 0   ; mC[5].d= 0   ;
mC[5].pertinence=-1;
```

```
    mC[6].a= 0   ; mC[6].b= 0   ; mC[6].c= 2   ; mC[6].d= 3   ;
mC[6].pertinence=-1;
    mC[7].a= 1   ; mC[7].b= 3   ; mC[7].c= 3   ; mC[7].d= 5   ;
mC[7].pertinence=-1;
    mC[8].a= 3   ; mC[8].b= 5   ; mC[8].c= 11  ; mC[8].d= 11  ;
mC[8].pertinence=-1;
    mC[9].a= 0   ; mC[9].b= 0   ; mC[9].c= 20  ; mC[9].d= 30  ;
mC[9].pertinence=-1;
    mC[10].a=20  ; mC[10].b=40  ; mC[10].c=40  ; mC[10].d=60  ;
mC[10].pertinence=-1;
    mC[11].a=50  ; mC[11].b=70  ; mC[11].c=100 ; mC[11].d=100 ;
mC[11].pertinence=-1;
    mC[12].a=0   ; mC[12].b=20  ; mC[12].c=30  ; mC[12].d=40  ;
mC[12].pertinence=-1;
    mC[13].a=30  ; mC[13].b=40  ; mC[13].c=40  ; mC[13].d=50  ;
mC[13].pertinence=-1;
    mC[14].a=40  ; mC[14].b=50  ; mC[14].c=50  ; mC[14].d=60  ;
mC[14].pertinence=-1;
    mC[15].a=50  ; mC[15].b=60  ; mC[15].c=60  ; mC[15].d=70  ;
mC[15].pertinence=-1;
    mC[16].a=60  ; mC[16].b=70  ; mC[16].c=90  ; mC[16].d=100 ;
mC[16].pertinence=-1;

    mR[0].precedenceA= 3; mR[0].precedenceB= 6;  mR[0].consequence= 15;
    mR[1].precedenceA= 3; mR[1].precedenceB= 5;  mR[1].consequence= 16;
    mR[2].precedenceA= 3; mR[2].precedenceB= 4;  mR[2].consequence= 17;
    mR[3].precedenceA= 2; mR[3].precedenceB= 6;  mR[3].consequence= 16;
    mR[4].precedenceA= 2; mR[4].precedenceB= 5;  mR[4].consequence= 15;
    mR[5].precedenceA= 2; mR[5].precedenceB= 4;  mR[5].consequence= 14;
    mR[6].precedenceA= 1; mR[6].precedenceB= 6;  mR[6].consequence= 13;
    mR[7].precedenceA= 1; mR[7].precedenceB= 5;  mR[7].consequence= 14;
    mR[8].precedenceA= 1; mR[8].precedenceB= 4;  mR[8].consequence= 15;
    mR[9].precedenceA= 3; mR[9].precedenceB= 12; mR[9].consequence= 17;
    mR[10].precedenceA=3; mR[10].precedenceB=11; mR[10].consequence=16;
    mR[11].precedenceA=3; mR[11].precedenceB=10; mR[11].consequence=15;
    mR[12].precedenceA=2; mR[12].precedenceB=12; mR[12].consequence=16;
    mR[13].precedenceA=2; mR[13].precedenceB=11; mR[13].consequence=16;
    mR[14].precedenceA=2; mR[14].precedenceB=10; mR[14].consequence=15;
    mR[15].precedenceA=1; mR[15].precedenceB=12; mR[15].consequence=15;
    mR[16].precedenceA=1; mR[16].precedenceB=11; mR[16].consequence=14;
    mR[17].precedenceA=1; mR[17].precedenceB=10; mR[17].consequence=13;
    mR[18].precedenceA=6; mR[18].precedenceB=9;  mR[18].consequence=14;
    mR[19].precedenceA=6; mR[19].precedenceB=8;  mR[19].consequence=14;
    mR[20].precedenceA=6; mR[20].precedenceB=7;  mR[20].consequence=15;
    mR[21].precedenceA=5; mR[21].precedenceB=9;  mR[21].consequence=14;
    mR[22].precedenceA=5; mR[22].precedenceB=8;  mR[22].consequence=15;
    mR[23].precedenceA=5; mR[23].precedenceB=7;  mR[23].consequence=16;
    mR[24].precedenceA=4; mR[24].precedenceB=9;  mR[24].consequence=15;
    mR[25].precedenceA=4; mR[25].precedenceB=8;  mR[25].consequence=16;
    mR[26].precedenceA=4; mR[26].precedenceB=7;  mR[26].consequence=17;
    mR[27].precedenceA=6; mR[27].precedenceB=12; mR[27].consequence=15;
    mR[28].precedenceA=6; mR[28].precedenceB=11; mR[28].consequence=14;
    mR[29].precedenceA=6; mR[29].precedenceB=10; mR[29].consequence=13;
    mR[30].precedenceA=5; mR[30].precedenceB=12; mR[30].consequence=16;
    mR[31].precedenceA=5; mR[31].precedenceB=11; mR[31].consequence=15;
    mR[32].precedenceA=5; mR[32].precedenceB=10; mR[32].consequence=14;
    mR[33].precedenceA=4; mR[33].precedenceB=12; mR[33].consequence=17;
    mR[34].precedenceA=4; mR[34].precedenceB=11; mR[34].consequence=16;
    mR[35].precedenceA=4; mR[35].precedenceB=10; mR[35].consequence=15;
    mR[36].precedenceA=4; mR[36].precedenceB=0;  mR[36].consequence=16;
```

```
    mE[0].value = 30;  mE[0].index = 3;
    mE[1].value = -86; mE[1].index = 3;
    mE[2].value = 11;  mE[2].index = 3;
    mE[3].value = 7;   mE[3].index = 3;

    PINFO("Configured");

    return 0;
}

/**
 * @brief Run the fuzzy system over the data
 * @param       None
 * @return      0 if success. -1 if error.
 */
int eFuzzyClass::run(fuzzySet_t* result) {
    PINFO("Running...");

    //
    // check matrix integrity
    //
//    ret = doCheckIntegrity();
//    if ( ret ) {
//        PERROR("in doCheckIntegrity()");
//        PERROR(ret);
//        return ret;
//    }

    //
    // calculate the number of input and output lines
    //
    doExtractUtilInfo();

    //
    // fuzzy - calculate mC pertinence
    //
    doCalculatePertinence();

    //
    // apply rules
    //
    doApplyRules();

    //
    // defuzzy
    //
    doDefuzzy();

    //
    // find centroid
    //
    *result = doFindCentroid();

    PINFO("Success");
    return 0;
}

/**
 * @brief
```

```
 * @param
 * @return
 */
void eFuzzyClass::printMatrixC() {

    for (int i=0; i<mC_lines_n; i++) {
        printf("\t%d\t%d\t%d\t%d\t", (int)mC[i].a,\
                                     (int)mC[i].b,\
                                     (int)mC[i].c,\
                                     (int)mC[i].d );
//        printf( mC[i].pertinence );
    }

}

//=========================================================
// Private Functions
//=========================================================

/**
 * @brief       Calculate pertinence of the crisp input value.
 * @param[in]   a, b, c, d. Modal values.
 * @param[in]   crisp. Input CRISP value.
 * @param[out]  u. The calculated pertinence.
 * @return      0 if success. -1 if error.
 */
int eFuzzyClass::fuzzy( fuzzySet_t a,
                        fuzzySet_t b,
                        fuzzySet_t c,
                        fuzzySet_t d,
                        fuzzySet_t crisp,
                        fuzzySet_t* u ) {
    //
    // Edge test
    //  By definition, when two crisp values are the same
    //  the highest pertinence must be used.
    //  Ifs are independet to result in 1 in the case of
    //  an edge crisp.
    //
    if ( crisp == a || crisp == d ) {
        *u = 0;
    }
    if ( crisp == b || crisp == c ) {
        *u = 1;
    }

    //
    // Simple Fuzzyfication
    //  Deal with the non-edge values
    //
    if ( crisp < a ) {          //crisp is smaller than a
        *u = 0;
    } else if ( crisp < b ) {   //crisp between a and b
        *u = (crisp-a)/(b-a);
    } else if ( crisp < c ) {   //crisp between b and c
        *u = 1;
    } else if ( crisp < d ) {   //crisp between c and d
        *u = (d-crisp)/(d-c);
    } else if ( crisp > d ) {   //crisp greater than d
        *u = 0;
```

```
    } else {
        //Should never get here
        return -1;
    }

    return 0;
}

/**
 * @brief       Apply rule over input data.
 * @param[in]   A1, A2. Input pertinence values from mC.
 * @param[inout]Cu. Target output pertinence value from mC.
 * @return      1 if rule was implyed. 0 if not.
 */
int eFuzzyClass::applyRule( fuzzySet_t A1,
                            fuzzySet_t A2,
                            fuzzySet_t* Cu ) {
    fuzzySet_t u;

    u = min(A1, A2);

    // max(u,*Cu)
    if ( u > *Cu ) {
        //Rule was implyed
        *Cu = u;
        return 1;
    }

    //Rule was not implyed
    return 0;
}

/**
 * @brief
 * @param
 * @return
 */
int eFuzzyClass::doCheckIntegrity() {
    return 0;
}

/**
 * @brief   Extract some useful info from the input matrix set.
 * @param   None.
 * @return  0 if success. -1 if error.
 */
int eFuzzyClass::doExtractUtilInfo() {

    //
    //Calculate the number of lines for each input matrix
    //
    mC_lines_n = ( sizeof(mC)/sizeof(mC_entry_t) );
    mR_lines_n = ( sizeof(mR)/sizeof(mR_entry_t) );
    mE_lines_n = ( sizeof(mE)/sizeof(mE_entry_t) );

    //
    //Calculate the number of input and output lines in the system
    //
    for(int i=0; i<mE_lines_n; i++) {
        total_input_lines = total_input_lines + mE[i].index;
```

```
    }

    total_output_lines = mC_lines_n - total_input_lines;

    return 0;
}

/**
 * @brief   Calculate pertinence for each line of matrix C.
 * @param   None.
 * @return  0 if success. -1 if error.
 */
int eFuzzyClass::doCalculatePertinence() {
    PINFO("Running Fuzzy on matrix C...");

    int mC_offset = 0;

    //Run for each line of matrix E
    for(int mE_curr_line=0; mE_curr_line<mE_lines_n; mE_curr_line++) {
        //Fill matrix C lines that are indicated in current matrix E line
        for(int i=0; i<mE[mE_curr_line].index; i++) {
            fuzzy(mC[i+mC_offset].a,\
                  mC[i+mC_offset].b,\
                  mC[i+mC_offset].c,\
                  mC[i+mC_offset].d,\
                  mE[mE_curr_line].value,\
                  &mC[i+mC_offset].pertinence );
        }
        //Update matrix C offset: It is the sum of mE index to the current
line.
        mC_offset = mC_offset + mE[mE_curr_line].index;
    }

    return 0;
}

/**
 * @brief   Apply rules and populate output pertinences on mC.
 * @param   None.
 * @return  0 if success. -1 if error.
 */
int eFuzzyClass::doApplyRules() {
    PINFO("Applying rules...");

    // -1 (and +1) are there because rules
    // begin in 1, but here they begin in 0.

    for(int i=0; i<mR_lines_n; i++) {
        if ( mR[i].precedenceB > 0 ) {      //Normal rule application
            if ( applyRule( mC[(mR[i].precedenceA-1)].pertinence,\
                            mC[(mR[i].precedenceB-1)].pertinence,\
                            &mC[(mR[i].consequence-1)].pertinence ) ) {
                PPRINTF("Rule %d implyed\n", i+1);
            }
        }
        else if (mR[i].precedenceB == 0 ) { //Second rule is zero
            PINFO("Second rule is zero.");
            //@todo: Test this
            mC[(mR[i].consequence-1)].pertinence = \
                    max(mC[(mR[i].precedenceA-1)].pertinence,\
```

```
                        mC[(mR[i].consequence-1)].pertinence);
        }
        else {                                  //Error. Should not happen.
            PERROR("precedenceB is negative.");
            return -1;
        }
    }

    return 0;
}

/**
 * @brief
 * @param
 * @return
 */
int eFuzzyClass::doDefuzzy() {
    PINFO("Running DeFuzzy on matrix C...");

    fuzzySet_t greater_x_value = 0;
    fuzzySet_t step = 0;

    //
    // Find the greatest value from input matrix
    //
    for ( int i=total_input_lines; i<mC_lines_n; i++) {
        greater_x_value = max( greater_x_value, mC[i].a );
        greater_x_value = max( greater_x_value, mC[i].b );
        greater_x_value = max( greater_x_value, mC[i].c );
        greater_x_value = max( greater_x_value, mC[i].d );
    }

    //
    // Calculate the step necessary to fill all points and fill buffer
    //
    step = greater_x_value/N_POINTS;

    Xn[0].x = 0;                            //Init first element

    for ( int i=1 ; i<(N_POINTS+1); i++) {  //Fill other elements
        Xn[i].x = Xn[i-1].x+step;
    }

    //
    // Calculate each point in Xn vector
    //
    for ( int i=0; i<(N_POINTS+1); i++) {
        fuzzySet_t max_cut;
        max_cut = 0;

        //Iteration runs for every output line of the system
        // in order to find the lowest value between the
        // Alfa Cut and the fuzzy value on a given point.
        // The greatest value is choosen for each output line
        for ( int j=total_input_lines; j<mC_lines_n; j++) {
            //Calculate the minimum between Alfa Cut and the
            // point's pertinence
            fuzzySet_t jCut;
            fuzzySet_t ju;
```

```
            fuzzy(mC[j].a, mC[j].b, mC[j].c, mC[j].d, Xn[i].x, &ju);

            jCut = min( mC[j].pertinence, ju);

            //Find max between the calculated value and the
            // previous value. The result is the max calculated
            // value for the output lines
            max_cut = max(jCut, max_cut);
        }
        //Store final value for this point
        Xn[i].u = max_cut;
    }

    return 0;
}

/**
 * @brief
 * @param
 * @return
 */
fuzzySet_t eFuzzyClass::doFindCentroid() {
    //The centroid method is the weighted average of the points:
    // result = sum( Xn[i].x*Xn[i].u )/( sum(Xn[i].u) )
    // with i in the range from 0 to the max value of N_POINTS
    fuzzySet_t numerator = 0;
    fuzzySet_t denominator = 0;
    fuzzySet_t result = 0;

    for ( int i=0; i<N_POINTS+1; i++) {
        numerator = numerator + ( Xn[i].x * Xn[i].u );
        denominator = denominator + Xn[i].u;
    }

    //Carefull with numerator or denominator overflow...
    result = numerator/denominator;

    return result;
}
```

```
/*
 * eFuzzy library for OpenWRT.
 * Tiago Ganselli
 *
 * Work done as prerequisite to the Masters Degree
 * in management of telecommunications networks.
 *
 * PUC-Campinas 2014
 */

#ifndef _EFUZZY_H
#define _EFUZZY_H

#include <stdio.h>

//=============================================================
// Configuration
//=============================================================
//#define PRINT_INFO
//#define PRINT_DEBUG
//#define PRINT_ERROR

#define N_POINTS        100

//=============================================================
// Types
//=============================================================
typedef float fuzzySet_t;

typedef struct {
    fuzzySet_t x;
    fuzzySet_t u;
}uVector_t;

struct mC_entry_t {
    fuzzySet_t a;
    fuzzySet_t b;
    fuzzySet_t c;
    fuzzySet_t d;
    fuzzySet_t pertinence;
};

struct mR_entry_t {
    int precedenceA;
    int precedenceB;
    int consequence;
};

struct mE_entry_t {
    fuzzySet_t value;
    fuzzySet_t index;
};

//=============================================================
// Macros
//=============================================================
#ifdef PRINT_INFO
    #define PINFO(txt)  printf(txt); printf("\n");

    //@todo: use vargs
```

```
    #define PPRINTF(txt, val)    \
    do{                          \
        printf(txt, val);\
    }while(0);

    #define PINFOVAL(txt, val)

    #define PINFOLN(txt, val)
#else
    #define PINFO(txt)
    #define PPRINTF(txt, val)
    #define PINFOVAL(txt, val)
    #define PINFOLN(txt, val)
#endif

#ifdef PRINT_ERROR
    #define PERROR(txt)          \
    do{                          \
        Serial.write(__func__); \
        Serial.write(": ");      \
        Serial.write("!ERROR ");\
        Serial.write(txt);       \
        Serial.write('\n');      \
    }while(0);
#else
    #define PERROR(txt)
#endif

#define max(a,b)                 \
({                               \
    __typeof__ (a) _a = (a);     \
    __typeof__ (b) _b = (b);     \
    _a > _b ? _a : _b;           \
})

#define min(a,b)                 \
({                               \
    __typeof__ (a) _a = (a);     \
    __typeof__ (b) _b = (b);     \
    _a < _b ? _a : _b;           \
})

//=============================================================
// Variables
//=============================================================
static mC_entry_t mC[17];
static mR_entry_t mR[37];
static mE_entry_t mE[4];

static uVector_t Xn[N_POINTS+1];

//=============================================================
// Classes
//=============================================================
/** Classe Embedded Fuzzy */
class eFuzzyClass {

    public:
        eFuzzyClass();
       ~eFuzzyClass();
```

```
        int configure();
        int run(fuzzySet_t* result);
        void printMatrixC(void);

    private:
        int mC_lines_n;
        int mR_lines_n;
        int mE_lines_n;

        int total_input_lines;
        int total_output_lines;

        int fuzzy( fuzzySet_t a,
                   fuzzySet_t b,
                   fuzzySet_t c,
                   fuzzySet_t d,
                   fuzzySet_t crisp,
                   fuzzySet_t* u );
        int applyRule( fuzzySet_t A1,
                       fuzzySet_t A2,
                       fuzzySet_t* Cu );
        int doCheckIntegrity(void);
        int doExtractUtilInfo(void);
        int doCalculatePertinence(void);
        int doApplyRules(void);
        int doDefuzzy(void);
        fuzzySet_t doFindCentroid(void);
};

extern eFuzzyClass eFuzzy;

#endif // _EFUZZY_H
```