Introdução à Lógica Fuzzy com Java

Aprenda a utilizar esta importante técnica de Inteligência Computacional em sistemas Java.

Autores: Diogo da Silva Magalhães Gomes e Max de Castro Rodrigues

O que fazer quando um processo de decisão inerentemente humano deve ser automatizado? Como modelar e implementar uma aplicação de forma que ela responda a estímulos de natureza imprecisa? Que ferramentas e técnicas podemos utilizar para simular computacionalmente o comportamento de um motorista de carro, ou mesmo de um operador de bolsa de valores? Para responder a estas perguntas, o presente artigo apresenta os conceitos, modelagem e aplicações práticas da lógica fuzzy, técnica utilizada para que processos automatizados se apropriem de particularidades do raciocínio humano, como o processo decisório multidisciplinar e a ponderação imprecisa.

Diariamente, precisamos lidar com informações que, por sua natureza, não podem ser representadas por valores numéricos precisos e, apesar disso, precisamos ter a habilidade de analisá-las para tomarmos nossas decisões. São informações que, embora possam ser mensuradas, envolvem certo grau de incerteza e interpretação subjetiva, sujeitas a conclusões divergentes se apuradas por diferentes indivíduos.

Por exemplo, quando precisamos nos deslocar de um ponto para outro não pensamos no número de passos que teremos de caminhar para decidir se tomaremos um ônibus ou se iremos a pé, analisamos somente se o destino é **perto** ou **longe**. Neste contexto, os termos **perto** e **longe** estão, portanto, intrinsecamente ligados à medida de distância, mas de uma forma subjetiva e imprecisa. O que para um indivíduo é considerado **perto**, para outro pode ser **longe**.

A maneira como interpretamos e analisamos estes conceitos ocorre de forma subjetiva, uma característica do raciocínio humano. Esta abstração e a imprecisão nela envolvida nos permitem, porém, levantar um interessante questionamento: a partir de qual número de passos um determinado destino deixa de ser considerado perto e passa a ser considerado longe? Para ilustrar melhor esse cenário, imagine que uma pessoa esteja dentro de sua casa. Em um dado momento, ela começa a caminhar e se afasta de casa, um único passo de cada vez. A cada passo, ela analisa se a distância para retornar para casa é considerada como perto ou longe. Em que momento específico a distância percorrida deixa de ser considerada perto e passa ser considerada longe? Um único passo percorrido entre um instante e outro poderá fazer diferença em sua avaliação, ou seja, um destino considerado perto, com um passo a mais, poderá passar a ser considerado longe?

Este mesmo raciocínio foi empregado no século IV a.C. pelo filósofo Eubulides de Mileto, na proposição que ficou conhecida como Paradoxo Sorites. Ele questiona: “quando um monte de areia deixa de ser um monte de areia, caso retiremos um grão de cada vez?”. Em sua proposição, ele define que se n grãos de areia são um monte, também o seriam $(n-1)$ grãos. Ou seja, um único grão de areia não deveria fazer diferença nesta classificação. O mesmo se diz do cenário oposto: um grão de areia não faz um monte, e adicionando mais um único grão, também não faria diferença. Mas isso significa, por indução, que nunca teremos um monte de areia, por mais que juntemos os grãos um a um.

Outro exemplo que demonstra como a imprecisão está inerentemente presente em problemas no mundo real se refere ao clássico cenário de classificação de um copo como **cheio** ou **vazio** (**Figura 1**). Considere que um copo esteja inicialmente vazio e gradativamente vai sendo preenchido com água, uma única gota sendo adicionada a cada vez. Até que momento poderíamos dizer que o copo está vazio? A partir de que momento específico este copo poderia ser considerado cheio?

Figura 1. Em que momento o copo deixa de ser considerado vazio, e passa a ser considerado cheio?

De fato, o que ocorre é que, assim como um único grão de areia não poderá afetar de imediato a classificação no cenário ilustrado pelo Paradoxo Sorites, também não poderá fazê-lo após uma única gota ser adicionada ao copo, ou após um único passo percorrido, no exemplo anterior. No mundo real, a imprecisão está inerentemente presente nestas situações. Nestes cenários, tanto o copo quanto o monte de areia não podem ser considerados, de imediato, como transitados de um conjunto (vazio) para o outro (cheio), após adicionar uma única unidade. Na realidade, a cada unidade adicionada, esta classificação gradativamente vai deixando de pertencer a um conjunto (ou categoria), e aumentando sua participação (ou pertinência) no conjunto adjacente. Ou seja, a cada gota adicionada, o copo gradativamente vai deixando de ser considerado vazio, e aumentando a sua correspondência com o que se define como sendo considerado cheio.

Na área de computação, porém, é comum buscarmos representações precisas para modelar este tipo de cenário, desenvolvendo algoritmos que tentam quantificar estes fatores utilizando valores precisos. Nestes casos, muitas vezes é especificado um valor limítrofe, a partir do qual a classificação para o cenário muda de um conjunto para o outro, ignorando sua natureza invariavelmente imprecisa, podendo ocasionar perda de informações importantes que deixaram de ser consideradas. Por exemplo, um processo tradicional de classificação poderá arbitrar como **longe** a distância entre dois pontos a partir de um valor pré-estabelecido, não permitindo ponderações subjetivas.

A lógica fuzzy, também conhecida como lógica nebulosa ou difusa, é uma técnica da área de inteligência computacional que nos permite representar modelos que contenham certo grau de incerteza ou imprecisão, características de situações do mundo real. Estas técnicas nos permitem codificar softwares que representem algoritmos mais próximos da forma como funciona o raciocínio humano, obtendo resultados satisfatórios uma vez que valores limítrofes e incertezas do modelo não são ignorados. Na lógica fuzzy, diferentemente da lógica clássica, um elemento pode pertencer **parcialmente** a um conjunto.

O surgimento do conceito de conjuntos fuzzy é atribuído à Lotfi Zadeh, da Universidade da Califórnia, que, em 1965, lançou o artigo “Fuzzy Sets”, introduzindo o assunto no meio acadêmico [1]. Os precursores no uso prático desta técnica em projetos são os japoneses, construindo um dos primeiros sistemas de controle fuzzy de uso crítico na estrada de ferro de Sendai, no Japão, no ano de 1987, controlando aceleração, frenagem e parada das composições nas estações metroviárias.

Sistemas baseados em lógica fuzzy podem ser utilizados em praticamente todas as áreas de conhecimento, como engenharia, matemática, biologia, medicina, etc. Como exemplos de sua utilização prática, podem ser citados os seguintes tipos de sistemas: de controle embarcado, de apoio à decisão, de reconhecimento de faces ou de padrões, de diagnóstico médico, de previsão do tempo, de cálculo e gerenciamento de risco, de controle de tráfego, de condução de veículos autônomos e de diversas outras finalidades [2]. O universo dos sistemas de controle embarcados é um caso especial, pois os exemplos são facilmente encontrados em nosso dia-a-dia. Diversos equipamentos domésticos e urbanos possuem modelos que contemplam a lógica fuzzy em suas diretrizes internas [3], tais como: máquinas de lavar, aspiradores de pó, televisores, caixas registradoras, sistemas de alarme, copiadoras, micro-ondas, foco automático em câmeras fotográficas, etc.

Uma das áreas de aplicação em que técnicas de lógica fuzzy são fortemente empregadas é a de desenvolvimento de jogos de computador, em função de sua característica de tentar reproduzir em um ambiente virtual particularidades do mundo real, inclusive suas imprecisões e aleatoriedades.

Sistemas que utilizam lógica nebulosa podem ter suas instruções em software e utilizar processadores de uso geral, optando por um custo mais baixo; ou ter suas instruções presentes em hardware em Circuitos Integrados de Aplicação Específica (*Application Specific Integrated Circuits*), que acarreta em custos mais elevados, porém obtêm um processamento mais rápido.

Este artigo apresenta os principais conceitos sobre lógica fuzzy e sistemas de inferência fuzzy, demonstrando como utilizar estas técnicas em sistemas Java com o componente JFuzzyLogic. Para ilustrar estes conceitos são apresentados dois casos hipotéticos: a modelagem de um sistema de apoio a decisão da área de RH de uma empresa; e um simulador baseado em um sistema de inferências fuzzy, que utiliza uma base de regras para representar o algoritmo de condução de um veículo em direção a uma vaga de estacionamento.

Lógica Fuzzy x Lógica Clássica

Tradicionalmente, uma proposição lógica clássica possui dois extremos: ou a premissa é completamente falsa, ou é completamente verdadeira. Só existe um resultado possível. Portanto, na matemática clássica, a função que define se um elemento **pertence** ou **não pertence** a um determinado conjunto somente poderá assumir os valores 0 (falso), ou 1 (verdadeiro), conforme a **Figura 2**.

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

Figura 2. Função de pertinência de um conjunto clássico;

Entretanto, na lógica fuzzy, o resultado desta proposição poderá variar em graus de verdade, em que poderemos considerá-lo como **parcialmente** verdadeiro, ou parcialmente falso. Isto é, um elemento pode pertencer parcialmente a um conjunto. Desta forma, a função de pertinência *μ(x)* de um elemento a um conjunto fuzzy pode assumir infinitos valores no intervalo [0,1], entre o totalmente falso e o totalmente verdadeiro, conforme descrito na **Figura 3**.

$$\mu_A(x) : X \rightarrow [0,1]$$

Figura 3. Função de pertinência de um conjunto fuzzy.

Portanto, cada elemento do conjunto fuzzy tem um grau de pertinência, também chamado de grau de inclusão, definido no intervalo [0,1], que descreve a possibilidade do elemento pertencer a este conjunto. Quanto maior o valor, mais compatível o elemento será em relação ao conjunto que o descreve.

Como o elemento pode pertencer parcialmente a um conjunto, é possível haver uma transição gradual da classificação de um elemento entre um conjunto e outro. Ou seja, entre o *falso* e o *verdadeiro* que definem se um elemento pertence a um conjunto na matemática clássica, infinitos valores podem ser assumidos utilizando a lógica fuzzy (**Figura 4**).

Figura 4. Ilustração com representação conceitual da pertinência entre conjuntos clássicos e as infinitas possibilidades tratadas pela lógica nebulosa.

Retornando ao exemplo apresentado pelo Paradoxo Sorites, a cada grão de areia adicionado, o cenário gradativamente aumenta sua participação no conjunto que o descreve como um monte de areia, aumentando o valor do seu grau de inclusão neste conjunto.

Podemos observar cenários desta natureza em diversas situações cotidianas como, por exemplo, quando precisamos classificar um determinado indivíduo por sua faixa etária. Para ilustrar, tomemos como base a classificação oficialmente utilizada pelo Instituto Nacional do Semiárido (INSA), em que a população brasileira é classificada conforme o seguinte critério, ilustrado na **Figura 5**.

Unidades da Federação e Grandes Regiões	Número de habitantes por faixa etária[1]				
	Criança até 11 anos)	Adolescente (12 a 18 anos)	Adulto (19 a 59 anos)	Idoso (60 anos ou mais)	Total
Semiárido	4.722.340	3.244.189	12.027.570	2.604.219	22.598.318
Nordeste	10.949.635	7.346.838	29.329.300	5.456.177	53.081.950
Sudeste	13.477.441	9.263.339	48.096.276	9.527.354	80.364.410
Centro-Oeste	2.684.884	1.774.688	8.360.388	1.238.134	14.058.094
Norte	3.909.162	2.382.544	8.491.279	1.081.469	15.864.454
Sul	4.602.472	3.266.336	16.230.618	3.287.465	27.386.891
Brasil	35.623.594	24.033.745	110.507.861	20.590.599	190.755.799

[1] As faixas etárias definidas para crianças, adolescente e idoso são as preconizadas pelo Estatuto da Criança e do Adolescente (Lei nº 8.069 de 13 de julho de 1990) e pelo Estatuto do Idoso (Lei nº 10.741 de 01 de outubro de 2003), respectivamente

Figura 5. Tabela de classificação etária da população, conforme o INSA. Fonte: http://www.insa.gov.br/censosab /index.php?option=com_content&view=article&id=101&Itemid=100

Este exemplo demonstra claramente um modelo em que são definidos valores limítrofes entre os conjuntos, de maneira que a classificação de um indivíduo em cada faixa etária sofrerá transições abruptas à medida que sua idade avança. Segundo a tabela, uma pessoa com 11 anos, 11 meses e 30 dias é considerada como uma criança. Mas, no dia seguinte, ao completar doze anos, esta mesma pessoa será diretamente reclassificada como adolescente, mesmo que não tenham ocorrido mudanças biológicas ou psicomotoras em sua constituição neste curto período de um dia transcorrido (um único grão de areia, na analogia do Paradoxo Sorites). De fato, assim como ocorre com os grãos de areia, a pessoa, a cada dia que passa, vai gradativamente mudando sua participação (grau de inclusão) de um conjunto para o outro.

Na matemática clássica, esta classificação poderia ser descrita pelo diagrama exposto na **Figura 6**.

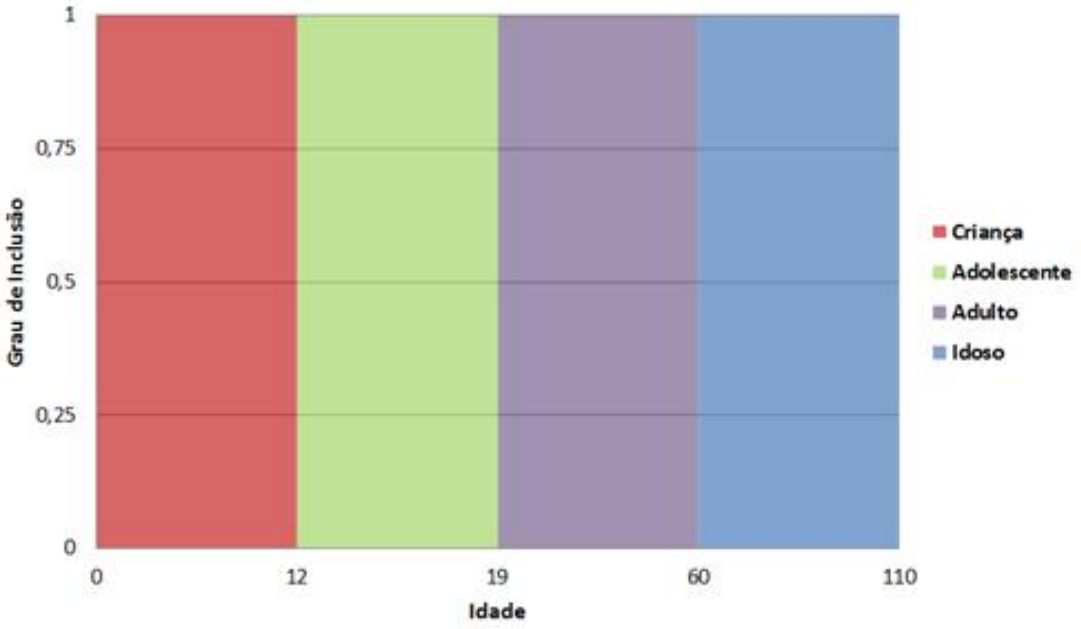

Figura 6. Gráfico ilustrando conjuntos clássicos para classificação etária da população.

Utilizando a notação de conjuntos fuzzy, este mesmo cenário poderia ser representado pelo diagrama ilustrado na **Figura 7**.

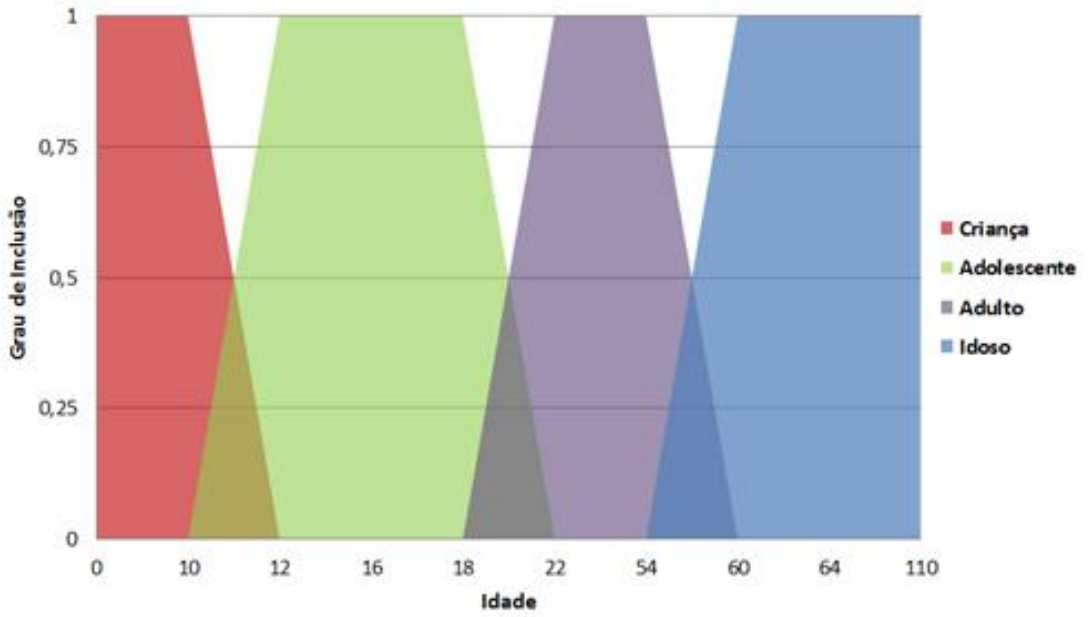

Figura 7. Diagrama ilustrando a classificação etária da população utilizando conjuntos fuzzy.

Destaca-se que as fronteiras entre os conjuntos não são nitidamente definidas, havendo inclusive certa interseção entre elas e, por isso, ocorre uma transição gradativa dos graus de pertinência entre os conjuntos à medida que a idade do indivíduo avança. Podemos observar que o conjunto adolescente assumiu um formato trapezoidal, iniciando-se a partir dos 10 anos, com um grau de pertinência zero, aumentando gradativamente até chegar a um pico aos 14 anos (onde um indivíduo será claramente definido como adolescente) com grau de pertinência igual a um, e depois diminuindo à medida que sua classificação transita para o conjunto seguinte, chegando novamente ao valor zero aos 21 anos.

Portanto, podemos definir que, neste exemplo, uma pessoa de treze anos pertence aos conjuntos *'Criança'* e *'Adolescente'* em diferentes graus de inclusão, mas não pertence aos conjuntos *'Adulto'* e *'Idoso'*, pois nestes dois últimos casos o grau de pertinência é zero. Este indivíduo com treze anos pertence predominantemente ao conjunto adolescente, mas não deixando de ser, porém, classificado também como criança em um menor grau de pertinência.

Este modelo de representação aproxima-se da forma como o entendimento humano naturalmente compreende as imprecisões inerentemente presentes nas situações cotidianas, pois não há fronteiras nitidamente definidas entre os conjuntos, e facilita a maneira como podemos representá-las ao traduzi-las em algoritmos matemáticos.

Sistemas de Inferência Fuzzy

Uma vez apresentado como os conjuntos fuzzy podem ser utilizados para representar as incertezas e imprecisões de um modelo, é necessário compreender como se dará a modelagem de um algoritmo que irá utilizá-los em um sistema real.

Sistemas de inferência fuzzy, ou *Fuzzy Inference Systems* (FIS), buscam representar a modelagem do raciocínio humano em forma de regras, ao invés de um algoritmo explicitamente restrito a modelos matemáticos exatos.

Comumente, quando procuramos expressar nosso raciocínio, naturalmente desenvolvemos uma descrição no formato de um conjunto de implicações lógicas, do tipo **se** (*antecedente*) **então** (*consequente*). À medida que a descrição se desenvolve, combinamos diversas sentenças através da utilização de operadores lógicos "*e*" e "*ou*", como na sentença a seguir: **se** (*antecedente1*) **e** (*antecedente2*) **então** (*consequente*).

Para exemplificar, consideremos um cenário hipotético de um sistema de controle de velocidade de um veículo, que analisa sua velocidade atual e se existe algum obstáculo à sua frente (como outro veículo), para então decidir a aceleração final ou frenagem a ser empregada, conforme o caso. Naturalmente, se houver algum obstáculo, o veículo deverá frear, senão deverá acelerar, até atingir uma velocidade de cruzeiro satisfatória.

Neste cenário, poderíamos formar a seguinte proposição, considerando como entrada as variáveis velocidade e distância do obstáculo: **se** (*velocidade* é *alta*) **e** (*obstáculo* é *perto*) **então** (*aceleração* é *frear*). Neste caso, analisando as entradas, o sistema decidirá que o veículo deve acionar a frenagem (aceleração negativa) caso haja um obstáculo à sua frente e sua velocidade seja considerada alta. Observe que as regras foram descritas sem utilizar valores precisos como entrada, mas apenas os conjuntos (ou partições) fuzzy que descrevem o intervalo lógico a ser considerado em cada proposição. Na lógica tradicional, esta regra provavelmente seria descrita como: "se (velocidade > 100km\h) e (obstáculo <50m) então (aceleração = -3m\s²).

Cabe destacar, porém, que na lógica clássica, uma sentença somente seria inferida caso todas as suas condições fossem verdadeiras. Isto ocorre de forma diferente nos sistemas de inferência fuzzy, pois as premissas assumem graus de verdade. Isto se reflete no resultado do processamento da sentença. Portanto, quanto maiores os graus de inclusão das variáveis de entrada (velocidade e distância do obstáculo), maior será o grau de inclusão da variável de saída no conjunto correspondente, permitindo que a ação de aceleração ou frenagem possa ser executada de maneira suave e progressiva.

O conjunto de regras pode ser considerado como o principal elemento de um sistema de inferências fuzzy, uma vez que representa a descrição do conhecimento de um especialista a respeito do problema a que o sistema se propõe a tratar. O **BOX 1** ilustra a arquitetura geral de um sistema de inferência fuzzy.

BOX 1. Arquitetura de um Sistema de Inferências Fuzzy

A arquitetura de um sistema de inferência fuzzy possui quatro componentes principais: um processador de entrada (*fuzzyfier*), que t ransforma os dados de entrada precisos em representações nebulosas; uma base de conhecimento, representada pelas variáveis ne bulosas, suas funções de pertinência e um conjunto de regras descritas por um especialista; uma máquina de inferência fuzzy, respo nsável pela avaliação das regras e inferência da saída; e um processador de saída (*defuzzifier*), que fornece um número real como s aída da inferência. A **Figura 8** ilustra conceitualmente a arquitetura geral de um sistema de inferências fuzzy.

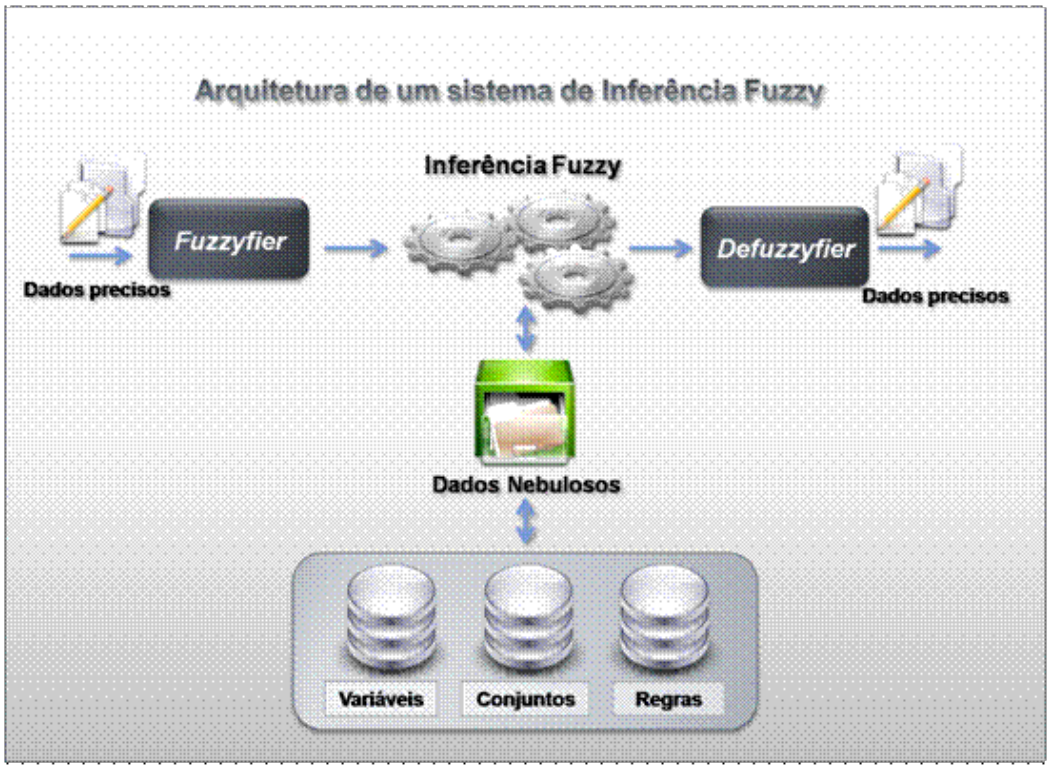

Figura 8. Arquitetura geral de um sistema de inferências fuzzy

Desta forma, os dados precisos fornecidos como entradas para o sistema de inferências são transformados em dados nebulosos (*fuzzification*). Estes são processados pelo núcleo de cálculo, em função do mapeamento de regras definidas e então, transformados novamente em dados precisos (*defuzzification*), que são retornados como saída pelo sistema de inferências. É importante destacar que tanto as entradas como a saída de um sistema de inferências fuzzy são valores numéricos, que são internamente processados utilizando-se definições fuzzy.

Apoio a decisão em um sistema de RH

O exemplo a seguir é um estudo de caso simples [4], que demonstra como é realizado o processo completo de modelagem de uma funcionalidade utilizando lógica fuzzy. Hipoteticamente, a necessidade é sinalizada pelo setor de RH de uma empresa, informando que é preciso agregar uma nova funcionalidade ao sistema atual. O objetivo da nova funcionalidade é poupar o analista de RH da tarefa de determinação dos valores de gratificações dadas aos funcionários, evitando a parcialidade e promovendo a isenção reivindicada pelo quadro de funcionários insatisfeitos.

Atualmente, o especialista (analista de RH) atem-se a dois critérios: tempo de experiência e tempo de formação/capacitação. De posse das informações preliminares, o projetista (analista/desenvolvedor/programador) observa que a nova funcionalidade possui critérios nos quais as fronteiras de decisão não são óbvias, possuindo características de uma decisão estritamente humana, suscetível à parcialidade e equívocos tendenciosos. O projetista então conclui que a lógica da nova funcionalidade se encaixa nos conceitos da lógica fuzzy e decide pela técnica para a solução desejada.

Consultando mais detalhadamente o especialista, fonte de informação fundamental em um projeto com lógica fuzzy, o projetista conclui que os critérios de experiência e de capacitação serão usados como variáveis de entrada do sistema, sendo a experiência expressa entre 0 e 30 anos, enquanto o tempo de capacitação é expresso entre 0 e 15 anos. A gratificação é a variável de saída, com valores entre R$0,00 e R$1.000,00. Estes intervalos são conhecidos como o universo de discurso das variáveis.

A próxima etapa do projeto é particionar o universo de discurso, ou seja, segmentar os intervalos de modo a poder atribuir a cada um deles um termo linguístico. Tais termos são a representação de como o especialista compreende as divisões internas dos critérios. Sendo assim, o especialista definiu que a variável de entrada *experiência* se divide em três segmentos: pouca, média e muita; conforme a **Figura 9**. A variável de entrada *capacitação* seguiu uma divisão similar em três partições: fraca, média e forte; conforme a **Figura 10**. E a saída, representada pela variável *gratificação*, foi dividida em cinco termos: muito baixa, baixa, média, alta e muito alta; conforme a **Figura 11**.

Com os intervalos e suas partições já especificados, o projetista precisa definir a forma dos conjuntos fuzzy, ou seja, a geometria da área de inclusão dos conjuntos, podendo ter diversas feições como: triangular, trapezoidal, gaussiana, etc.; sendo a mais simples e usual a forma triangular de comportamento linear. Os critérios usados para a seleção da forma do conjunto podem ser imprecisos e incertos, por isso geralmente eles são selecionados através de tentativa e erro, através de experimentação continuada. Para os objetivos deste exemplo, foi arbitrada a feição triangular para todos os conjuntos.

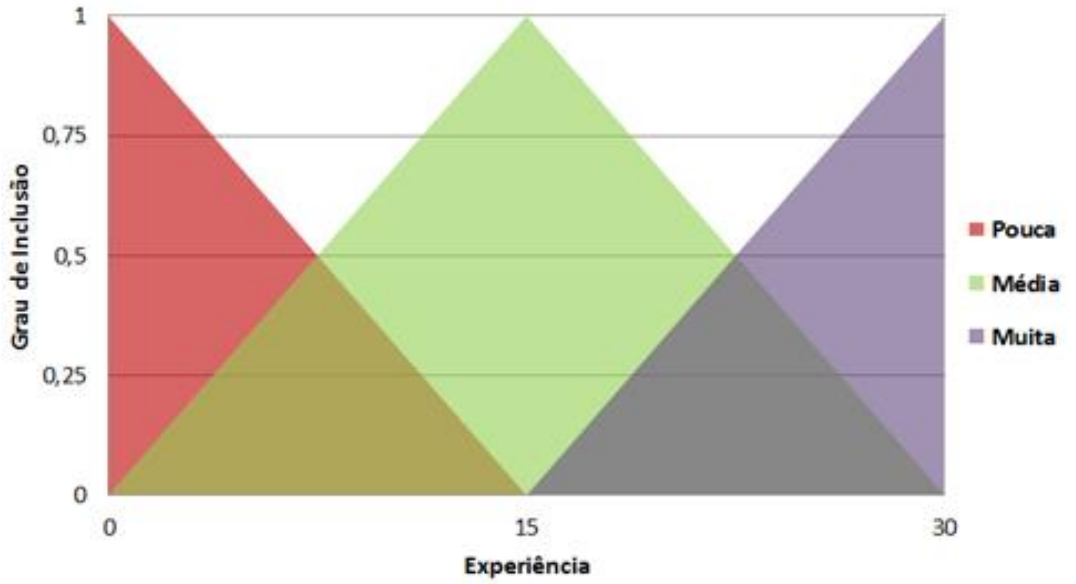

Figura 9. Conjunto fuzzy definido para a variável *experiência*.

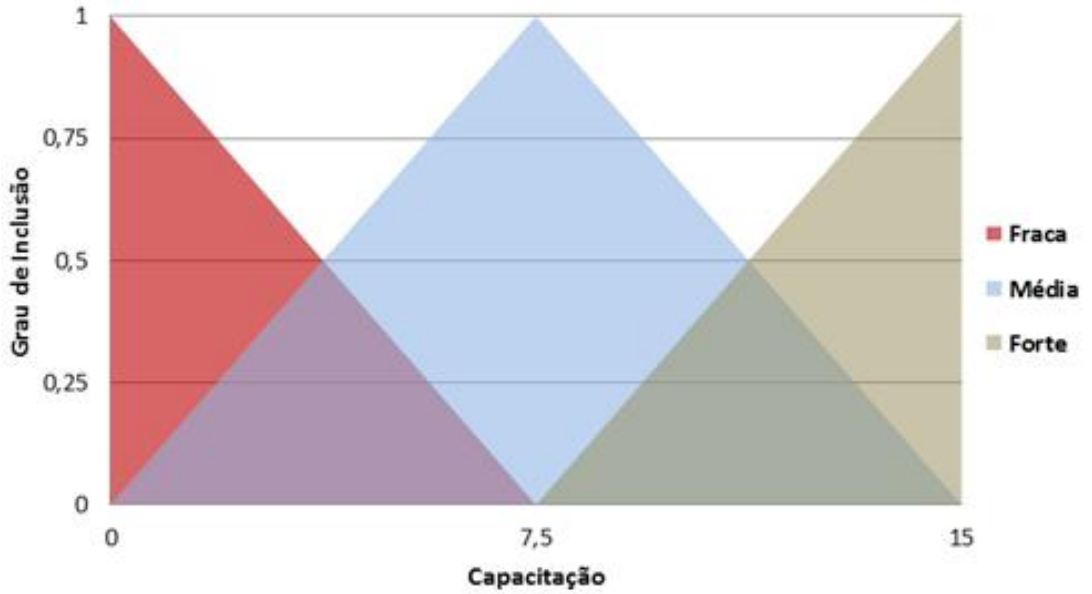

Figura 10. Conjunto fuzzy definido para a variável *capacitação*.

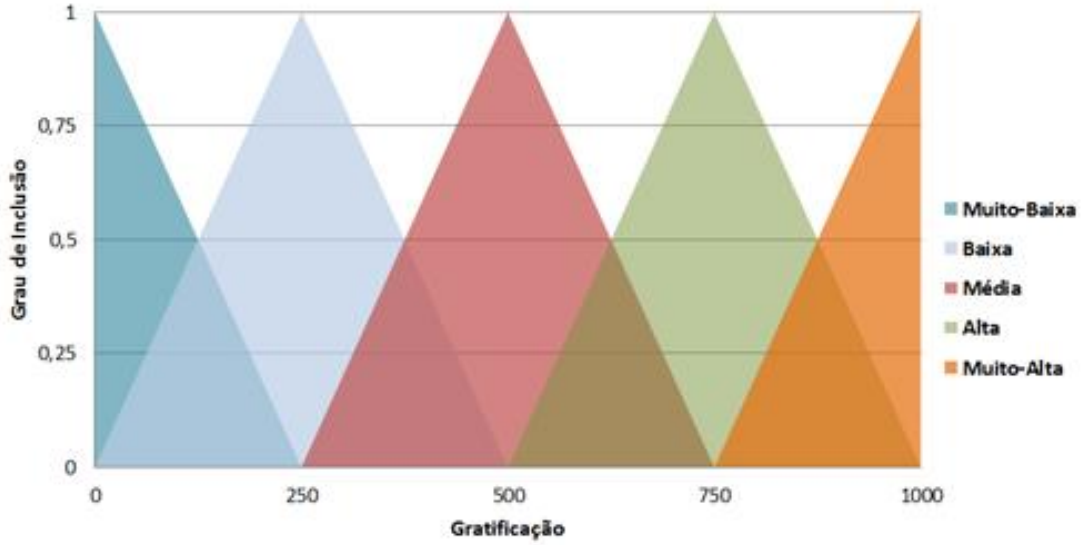

Figura 11. Conjunto fuzzy definido para a variável de saída *gratificação*.

Observando a **Figura 10**, nota-se que o profissional com 10 anos de capacitação seria avaliado pelo processo como parcialmente "Média" e parcialmente "Forte", pois existe um grau de inclusão para estes dois conjuntos, sendo mais presente em "Média". De forma similar, alguém com uma experiência de 20 anos estaria incluído nos conjuntos "Média" e "Muita". Nota-se que os conjuntos de pertinência, ou seja, as formas triangulares, encaminham a ponderação feita pela lógica fuzzy, determinando a proporção de inclusão nos critérios específicos.

Na sequência, o especialista determina as regras que direcionarão a decisão feita pelo sistema. A constituição das mesmas se faz com o encadeamento de condições e uma conclusão, na forma: *SE <condição1> E <condição2> ENTÃO <conclusão>*. A conclusão indica a qual conjunto a variável de saída pertence. As regras para o problema de valoração de gratificação, definidas pelo especialista, são as seguintes:

1. SE *capacitação* é "Fraca" E *experiência* é "Pouca" ENTÃO *gratificação* é "Muito-Baixa"
2. SE *capacitação* é "Fraca" E *experiência* é "Média" ENTÃO *gratificação* é "Baixa"
3. SE *capacitação* é "Fraca" E *experiência* é "Muita" ENTÃO *gratificação* é "Média"
4. SE *capacitação* é "Média" E *experiência* é "Pouca" ENTÃO *gratificação* é "Baixa"
5. SE *capacitação* é "Média" E *experiência* é "Média" ENTÃO *gratificação* é "Média"
6. SE *capacitação* é "Média" E *experiência* é "Muita" ENTÃO *gratificação* é "Alta"
7. SE *capacitação* é "Forte" E *experiência* é "Pouca" ENTÃO *gratificação* é "Média"
8. SE *capacitação* é "Forte" E *experiência* é "Média" ENTÃO *gratificação* é "Alta"
9. SE *capacitação* é "Forte" E *experiência* é "Muita" ENTÃO *gratificação* é "Muito-Alta"

Uma vez definidos os conjuntos de pertinência e as regras, o sistema usando lógica fuzzy já pode ser implementado utilizando uma linguagem qualquer e uma API especialista que será responsável pela fuzzificação e defuzzificação. A API receberá as regras e os conjuntos fuzzy como parâmetros além das variáveis de entrada, de forma que gere a variável de saída *gratificação*. Existem diversas APIs disponíveis para executar a tarefa, sendo a JFuzzyLogic a indicada aqui.

Uma visão segmentada do processo de fuzzificação e defuzzificação pode ser observada no diagrama da **Figura 12**, onde é avaliado o caso de um profissional com 10 anos de capacitação e 20 anos de experiência. O processo usa a capacitação e a experiência informada para rastrear as regras que foram ativadas e conclui que o profissional é merecedor de uma gratificação de R$600,00. Observando as regras já definidas e o diagrama da **Figura 12**, nota-se que as regras 5, 6, 8 e 9 foram ativadas por atenderem as condições impostas. Por exemplo, a regra 5 foi ativada porque atendeu às duas condições necessárias: a *capacitação* é "Média" e a *experiência* é "Média", levando a variável de saída *gratificação* a ter um valor proporcional ao grau de inclusão das duas condições atendidas. O grau de inclusão obtido no conjunto de saída em cada regra, representado pela área em negrito dos triângulos da variável *gratificação*, é definido por um cálculo matemático de responsabilidade da API. Com os graus de inclusão dos conjuntos fuzzy de *gratificação* já ponderados pela API (terceira coluna na direita do diagrama), a defuzzificação é efetuada, gerando a saída única, que é o valor de gratificação esperado como resultado final do processo de decisão nebuloso.

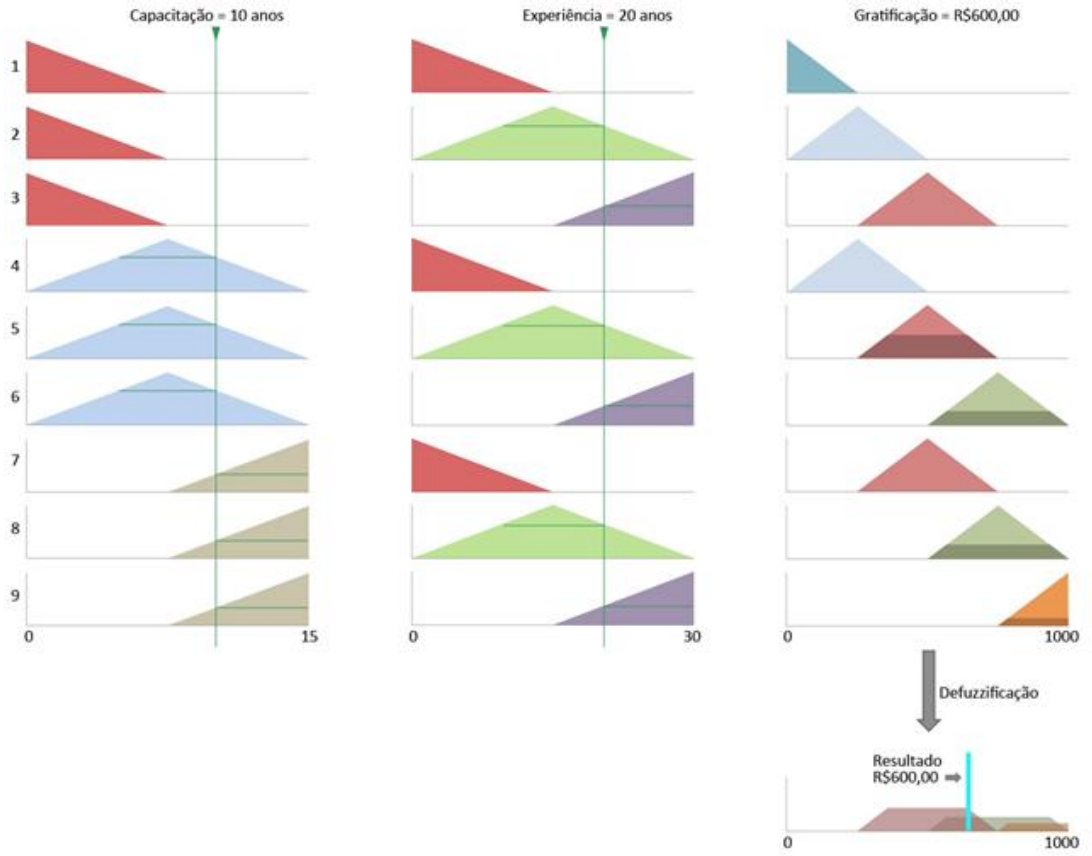

Figura 12. Processo de fuzzificação e defuzzificação.

Neste exemplo, o método de defuzzificação utilizado para obter o valor final, descrito no último gráfico, foi o Centro de Gravidade. Existem diversos métodos possíveis de serem utilizados nos processos de fuzzificação e defuzzificação executados pela API, de maneira que cada método efetua sua formulação matemática própria. O que ocorre no interior da API, como é esperado, fica totalmente transparente para o projetista e seu detalhamento não faz parte do escopo deste trabalho. Caso o leitor deseje saber mais, é preciso um estudo mais aprofundado por parte do mesmo em literatura especializada [1].

O exemplo da determinação do valor de gratificação, apoiado pela lógica fuzzy, esclarece o processo de modelagem de uma solução orientada a regras de negócio, encontrada no dia-a-dia de muitos sistemas corporativos. No próximo exemplo, mais extenso, o enfoque será dado a um sistema de controle, onde a modelagem, a formulação matemática e a codificação são bem detalhadas.

O Simulador de Estacionamento de um Veículo

Para demonstrar na prática a construção de um sistema fuzzy, será apresentada a implementação em Java de um controlador baseado no clássico problema do estacionamento de um veículo, amplamente referenciado na literatura técnica acadêmica [5]. Este controlador utiliza um sistema de inferências baseado em regras para conduzir um veículo autônomo em direção a um local predeterminado, representado por uma vaga de estacionamento.

O simulador será desenvolvido em dois passos: o primeiro corresponde à implementação do controlador fuzzy, que é o núcleo de inferências, que efetivamente processa as entradas e a saída em cada passo, e é o responsável por calcular a trajetória do veículo. O segundo passo é o simulador gráfico, utilizado para testar e validar o comportamento do controlador fuzzy, permitindo ao usuário visualizar a trajetória percorrida e efetuar ajustes na modelagem dos parâmetros do controlador, caso necessário.

O objetivo do simulador é estabelecer um algoritmo inteligente capaz de guiar um veículo, localizado em uma determinada posição do pátio, em direção a uma vaga de estacionamento previamente demarcada. O veículo pode estar em uma determinada posição x em relação ao eixo horizontal, e rotacionado em um ângulo φ em relação a este eixo. O sistema de inferências fuzzy irá atuar como se fosse o motorista do veículo, especificando o ângulo em que a roda deve ser girada para que o veículo seja guiado em direção ao local definido. A condição de parada é que o veículo esteja localizado na posição central em relação ao eixo horizontal, e alinhado perpendicularmente a este eixo, na área demarcada como sendo a vaga de estacionamento.

Como anteriormente apresentado, os sistemas de inferência fuzzy são modelados à semelhança do raciocínio de um especialista

humano. Neste caso, o especialista que poderá descrever como um veículo deve ser estacionado seria um motorista. Para descrever os passos necessários, basicamente ele explicaria que: “se o veículo está localizado à direita do ponto de destino, o volante deve ser girado para a esquerda; se o veículo está à direita, o volante deve ser girado para a esquerda”. Se o veículo, porém, estiver alinhado com a vaga de estacionamento, o volante deve ser mantido na posição central, para o veículo seguir em frente. As regras devem considerar ainda o ângulo do veículo em relação ao ponto de destino, que deve ser rotacionado para que fique alinhado verticalmente na vaga de estacionamento.

Dito isso, inicialmente iremos identificar as partições fuzzy para cada variável de entrada e estabelecer o conjunto de regras para representar o conhecimento do motorista na forma de um sistema de inferências fuzzy.

Neste cenário, consideraremos duas variáveis de entrada: a posição ***x*** (distância no eixo horizontal), e ***φ*** (ângulo do veículo em relação ao eixo horizontal). O sistema de inferências irá calcular o valor da variável de saída ***θ*** (ângulo da roda do veículo), que corresponde à direção em que o veículo deve ser encaminhado enquanto se desloca a cada passo. A condição de parada é de que ele deve estar posicionado dentro da área demarcada (posição central no eixo horizontal), em um ângulo de aproximadamente 90º (ângulo vertical). Para simplificação do problema, a posição ***y*** do veículo não será considerada no cálculo da inferência. A **Figura 13** representa as variáveis de entrada e saída consideradas pelo simulador.

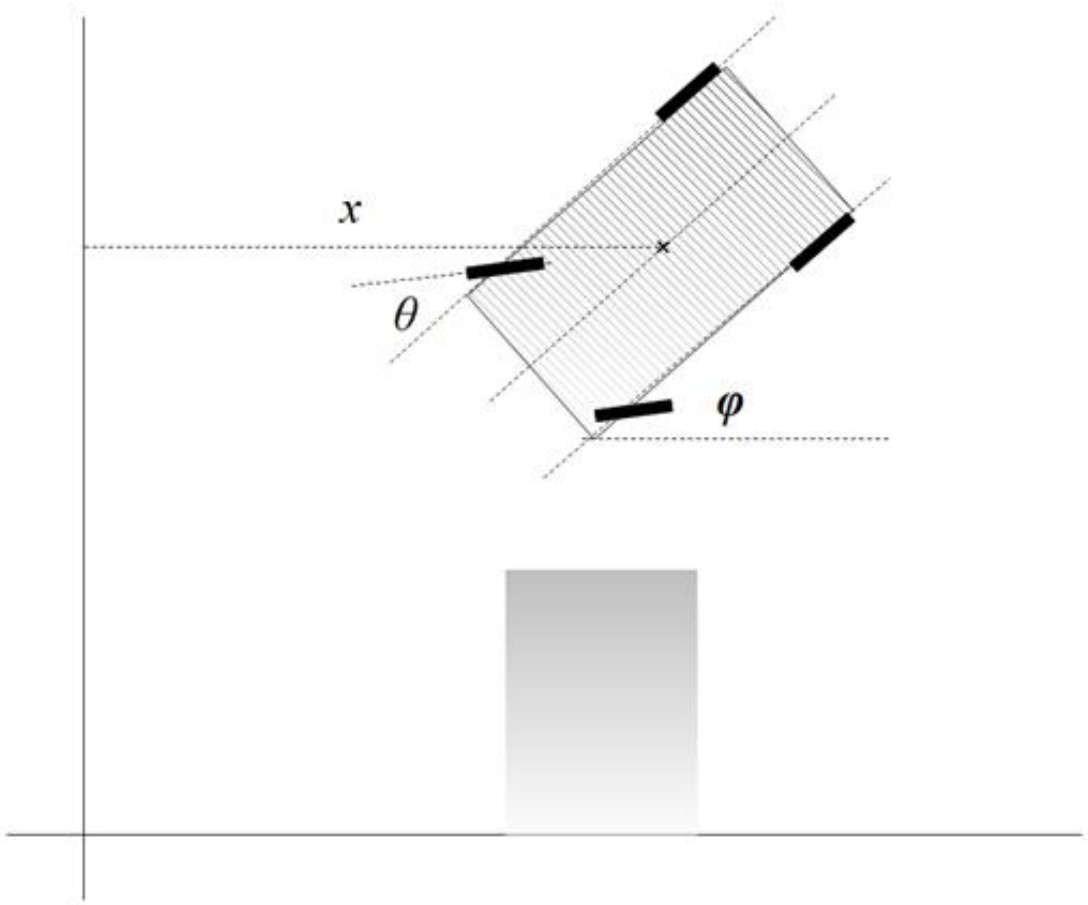

Figura 13. Esquema das variáveis de entrada e saída consideradas pelo controlador fuzzy.

Ao iniciar a simulação, o veículo deve ser automaticamente direcionado por uma trajetória que o conduza para a posição final determinada pela vaga no estacionamento. O deslocamento do veículo é simulado pelo sistema em uma sequência de passos, de forma que a cada passo o sistema de inferências receba como entrada a posição ***x*** e o ângulo ***φ***, e calcule o ângulo ***θ*** para a roda. A nova posição ***x'*** e o novo ângulo do veículo ***φ'*** serão, portanto, atualizados em função do valor ***θ*** retornado pelo controlador. A variável ***w*** é a constante que representa a distância fixa em que o veículo é deslocado a cada passo.

A **Figura 14** descreve as funções que determinam como o veículo será movimentado a cada passo. Estas equações de deslocamento serão utilizadas pelo simulador gráfico para movimentar o veículo em sua trajetória calculada pelo controlador.

Cabe destacar que no passo seguinte da iteração as variáveis ***x*** e ***φ*** atualizadas serão novamente submetidas como entradas para o controlador, e um novo valor de saída ***θ*** será calculado em cada passo. Este laço de repetição será executado até que a condição de parada seja atendida: a posição ***x*** deve estar próxima ao ponto central, e o ângulo ***φ*** próximo a 90º.

$$\varphi' = \varphi + \theta$$

$$x' = x + w\cos(\varphi')$$

$$y' = y + w\sin(\varphi')$$

Figura 14. Equações que descrevem as funções de movimento do veículo a cada passo.

A interface proposta para o simulador de estacionamento

O processo de construção e modelagem de um sistema de inferências fuzzy ocorre de maneira iterativa. Neste cenário, o sistema deve ser submetido a diversos testes e ajustes em seus parâmetros de modelagem, até que os resultados obtidos sejam satisfatórios.

Uma vez construído o controlador fuzzy, utiliza-se um ambiente de simulação que fornece uma interface para visualizar graficamente a resposta do controlador em função dos valores de entrada, permitindo uma melhor interpretação dos resultados para providenciar os ajustes necessários. A interface gráfica para o ambiente de simulação objetiva reproduzir todos os requisitos que foram especificados na construção do controlador fuzzy, simulando um ambiente próximo ao real para a execução dos testes. Obtendo-se resultados satisfatórios, o controlador poderá então ser implantado definitivamente no dispositivo ou software para o qual foi projetado.

A **Figura 15** ilustra a interface gráfica proposta para o simulador, demonstrando a imagem estilizada do veículo a ser movimentado, os botões para permitir o início/pausa da simulação, o botão para rotação do veículo em ambas as direções e a opção para habilitar o recurso de rastro de trajetória. O botão de rastro possibilita visualizar as diferentes trajetórias percorridas pelo veículo em sucessivas simulações. Ademais, o veículo pode ser arrastado com o mouse para qualquer posição da tela, e rotacionado a partir dos botões de rotação, para determinar a configuração inicial da simulação. Esta interface é construída com Swing e utiliza recursos simples de Java2D para posicionamento e rotação do veículo.

No canto superior direito, um quadro apresenta a situação atual das variáveis envolvidas na simulação: a posição do veículo, seu ângulo de rotação e o ângulo da roda.

Figura 15. Interface gráfica para o simulador de estacionamento.

Construção do Controlador Fuzzy

O Controlador Fuzzy corresponde ao núcleo de inferências responsável por calcular o valor apropriado para a saída, a partir dos valores fornecidos como entrada, considerando os parâmetros definidos em uma base de conhecimento fuzzy. Neste caso, o controlador irá determinar o ângulo da roda $\boldsymbol{\theta}$ em função da posição $\boldsymbol{x}$ e do ângulo de rotação $\boldsymbol{\varphi}$ do veículo.

Este núcleo de processamento deve ser independente de interface gráfica. Ou seja, objetiva-se que o controlador possa ser desenvolvido, testado e ajustado de maneira a ser posteriormente implantado em um sistema específico, seja ele um sistema embarcado (em um veículo autônomo), um carrinho de brinquedo ou um jogo de computador, por exemplo. Se os critérios utilizados na modelagem permanecerem válidos, o controlador deverá responder da mesma forma em quaisquer desses ambientes.

Identificação das partições Fuzzy

O primeiro passo para a construção do sistema de inferências é determinar as partições nebulosas (ou conjuntos) em que cada variável de entrada pode ser dividida. Cada partição representa um intervalo de dados em que o universo de discurso poderá ser dividido, e possui um termo linguístico a ela associada.

Recorrendo ao exemplo anterior da classificação etária da população, a variável idade foi dividida em quatro partições nebulosas, definidas pelos termos linguísticos: criança, adolescente, adulto e idoso. Cada partição possui um intervalo de dados válido para cada faixa etária.

De forma semelhante, é necessário dividir as variáveis de entrada em partições que representem a localização e a rotação do veículo, assim como o ângulo da roda a ser calculado como saída. Esta divisão é feita de forma a representar a maneira como o entendimento humano interpreta o universo de possibilidades para cada variável, e é determinada pelo especialista que está analisando o problema.

Portanto, em relação à posição do veículo (posição $\boldsymbol{x}$), definiu-se que ele se desloca ao longo de um eixo horizontal, podendo ocupar diferentes regiões neste eixo. Analisando subjetivamente o intervalo de dados, poderíamos estabelecer, a princípio, que o veículo pode estar localizado em uma posição central neste eixo, ou estar localizado à esquerda, ou à direita. Especificando um pouco melhor as possibilidades, podemos inserir mais duas regiões (ou partições) a este cenário: a posição centro-esquerda e a posição centro-direita, localizadas respectivamente entre as posições central e esquerda, e central e direita.

O universo de discurso referente à posição ***x*** pode ser então subdividido em cinco partições nebulosas: à esquerda (LE), centro-esquerda (LC), centro (CE), centro-direita (RC) e direita (RI). Cada partição representa um intervalo em que o veículo pode ser posicionado. Neste exemplo, assumiremos que o eixo horizontal possui 100 pontos no total (universo de discurso) e os intervalos para cada partição nebulosa poderiam ser especificados conforme a **Figura 16**, onde a posição central concentra-se no entorno do ponto x=50.

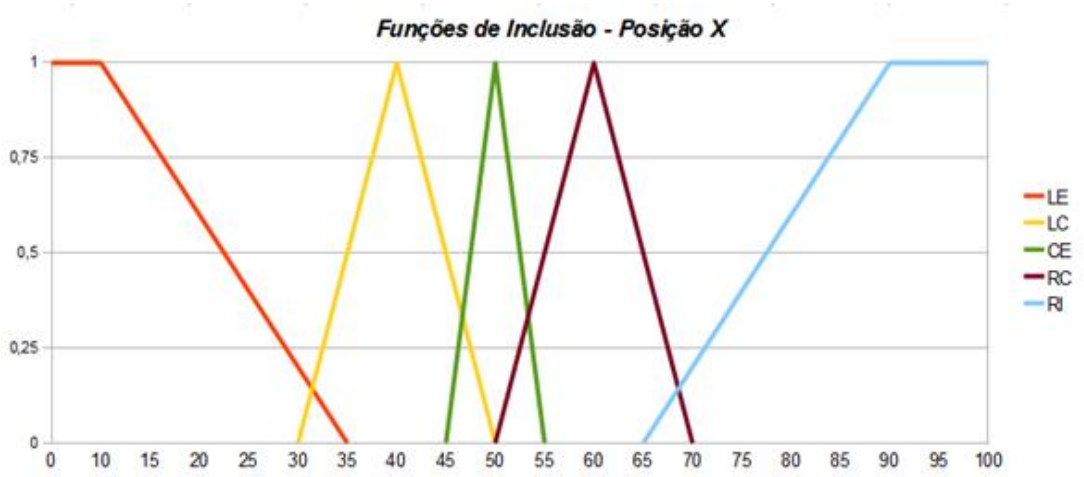

Figura 16. Partições fuzzy para a variável de entrada *posição do veículo*.

Observe no gráfico (**Figura 16**) que as funções de pertinência fuzzy possuem formatos triangulares ou trapezoidais, que definem o intervalo válido para cada conjunto. O conjunto CE, por exemplo, começa no ponto x=45 com grau de inclusão zero, chega a um ápice no ponto x=*50*, que possui grau de inclusão igual a um, e por fim, no ponto x=55, retorna ao grau de inclusão zero. Esta função triangular, portanto, é definida pelos pontos {45, 50, 55}.

Seguindo a mesma linha de raciocínio, deve-se estabelecer as partições para a variável ângulo do veículo (***φ***), considerando que o veículo pode estar rotacionado em um ângulo qualquer dentro do universo de discurso de 360º. Para que o veículo seja corretamente estacionado, ele deve estar alinhado com a vaga de estacionamento, de maneira perpendicular ao eixo horizontal e, portanto, em um ângulo de 90º. Desta forma, a primeira região facilmente identificável é a que representa o ângulo vertical do veículo (VE), situada no entorno de 90º. A partir dela, podemos especificar as demais partições, à medida que o veículo estiver rotacionado um pouco mais para a direita ou para a esquerda. Neste caso, optou-se por considerar o universo de discurso no intervalo entre -90º e 270º, totalizando 360º, para que o conjunto vertical, correspondendo ao ângulo de 90º, possa ser representado no centro das demais partições. A **Figura 17** ilustra graficamente como estas partições estão dispostas ao longo do universo de discurso, tomando por base a posição vertical como referência inicial, alinhada à vaga de estacionamento. Observa-se que entre as partições as fronteiras não são claramente definidas, ou seja, existe uma gradação na transição entre as partições.

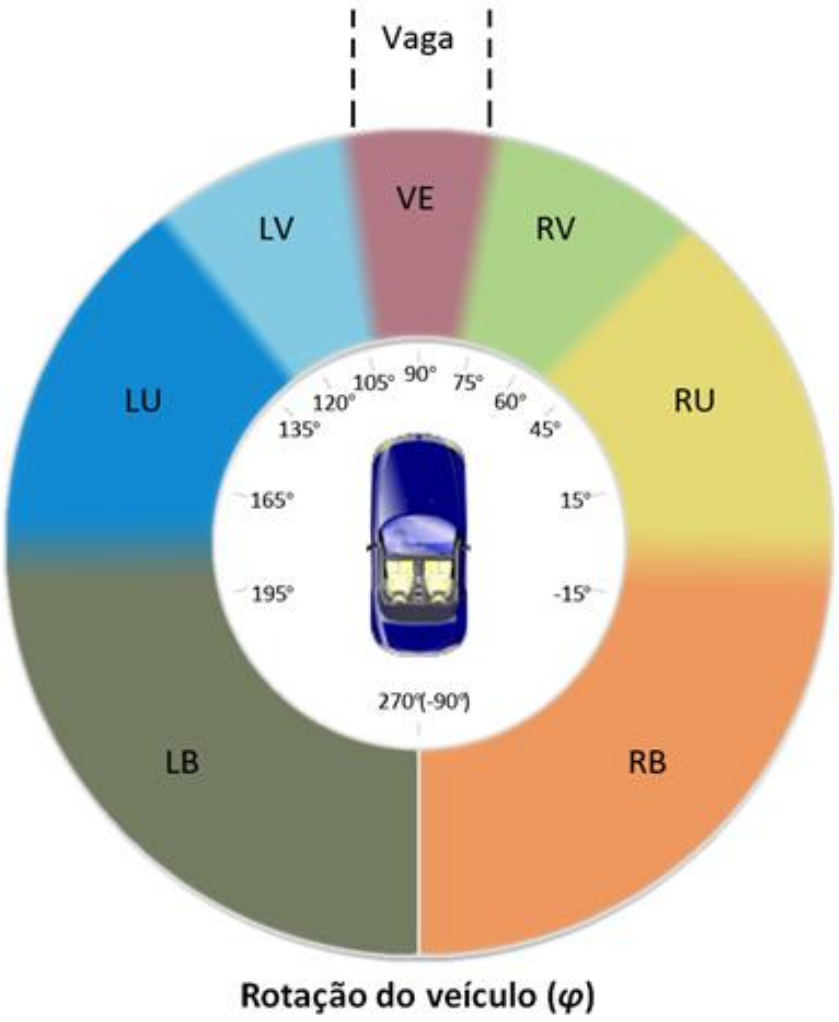

Figura 17. Representação gráfica das partições para a variável rotação do veículo (***φ***).

Desta forma, as seguintes possibilidades poderiam ser consideradas para as partições nebulosas da variável ***φ***: inferior-direito (RB), superior-direito (RU), vertical à direita (RV), vertical (VE), vertical à esquerda (LV), superior à esquerda (LU) e inferior à esquerda (LB). Os intervalos de dados estão descritos conforme ilustrado pela **Figura 18**.

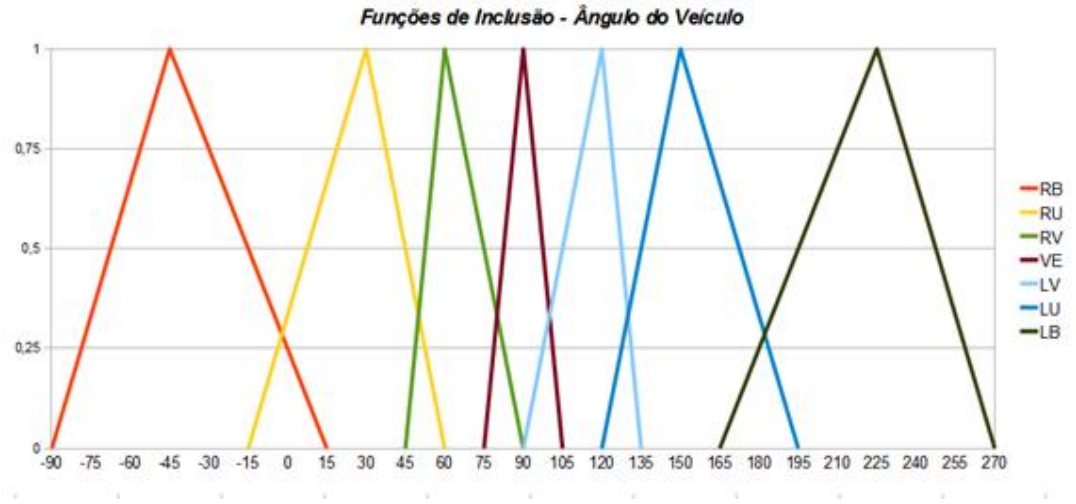

Figura 18. Partições fuzzy para a variável de entrada *ângulo do veículo*.

Para a variável de saída ***θ***, que corresponde ao ângulo da roda, optou-se por criar sete partições para subdividir o universo de discurso, que se situa no intervalo entre -30º a +30º. Em termos gerais, esta variável define o quanto a roda deve ser girada para a direita, para a esquerda, ou mantida alinhada ao centro, para direcionar o veículo ao local desejado. A roda alinhada ao centro, no entorno do ângulo de rotação de 0º, é representada pela partição Zero (ZE). As demais partições foram definidas tomando esta por referência, caso a roda seja girada em maior ou menor ângulo, positivo (para a esquerda) ou negativo (para a direita). Portanto, as seguintes partições foram especificadas para esta variável (**Figura 19**): negativo grande (NB), negativo médio (NM), negativo pequeno (NS), zero (ZE), positivo pequeno (PS), positivo médio (PM) e positivo grande (PB).

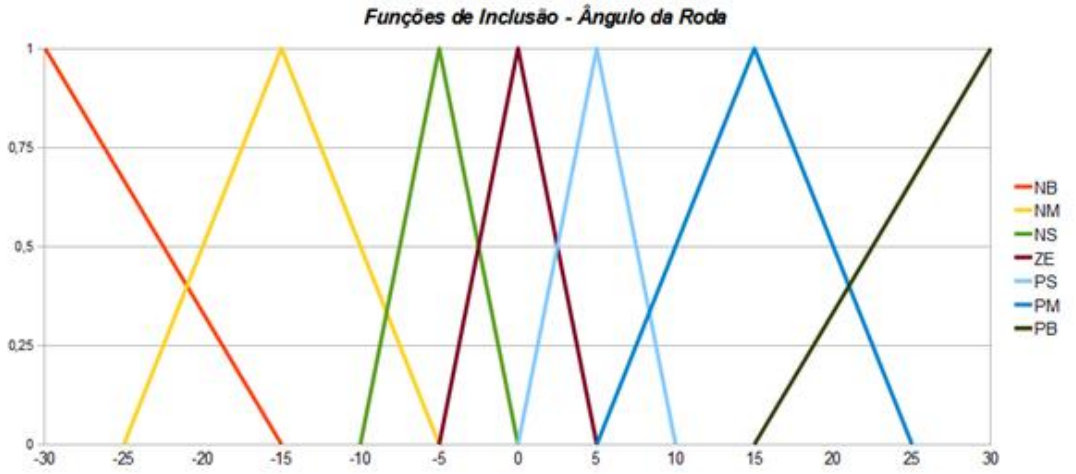

Figura 19. Partições fuzzy para a variável de saída *ângulo da roda*.

A **Tabela 1** resume como foram determinadas as partições nebulosas para cada variável de entrada e para a variável de saída, e os respectivos intervalos para as suas funções de pertinência. Os valores de cada intervalo serão utilizados posteriormente, no tópico que apresenta a implementação do controlador, para definir os parâmetros necessários aos cálculos da inferência.

Partições Nebulosas e Rótulos					
Variável de entrada *φ - ângulo do veículo*		**Variável de Entrada *x – deslocamento horizontal***		**Variável de Saída *φ- ângulo da roda***	
RB	*Right Below* (-90, -45, -15)	**LE**	*Left* (0, 0, 10, 35)	**NB**	*Negative Big* (-30, -30, -15)
RU	*Right Upper* (-15, 30, 60)	**LC**	*Left Center* (30, 40, 50)	**NM**	*Negative Med* (-28, -15, -5)
RV	*Right Vertical* (45, 67, 90)	**CE**	*Center* (45, 50, 55)	**NS**	*Negative Small* (-10, -5, 0)
VE	*Vertical* (75, 90, 105)	**RC**	*Right Center* (50, 60, 70)	**ZE**	*Zero* (-5, 0, 5)
LV	*Left Vertical* (90, 112, 135)	**RI**	*Right* (65, 90, 100, 100)	**OS**	*Positive Small* (0, 5, 10)
LU	*Left Upper* (120, 150, 195)			**PM**	*Positive Med* (5, 15, 25)
LB	*Left Below* (165, 225, 270)			**PB**	*Positive Big* (15, 30, 30)

Tabela 1. Partições fuzzy e intervalos utilizados pelo sistema de inferências.

Construção da base de regras

Uma vez definidos todos os conjuntos fuzzy para as variáveis de entrada e de saída, o próximo passo é definir o conjunto de regras do sistema de inferências fuzzy, que correlaciona as implicações lógicas entre os conjuntos de entrada (condições) e o conjunto de saída (consequência).

Estas regras são construídas à semelhança de como seria o raciocínio de um condutor humano nesta situação. Ou seja: se o veículo estiver à esquerda da vaga, a roda deverá ser girada para a direita. Se o veículo estiver alinhado com o centro da vaga, o ângulo da roda

deve ser mantido.

Depois, algumas regras devem ser combinadas para concluir a lógica da inferência: se o veículo estiver localizado na posição central, mas estiver rotacionado um pouco à direita (ângulo menor que 90º), então a roda deve ser girada um pouco à esquerda (em um ângulo positivo), para redirecionar o veículo.

Estas regras são então transcritas na forma de sentenças lógicas que correlacionam as partições definidas para as variáveis de entrada com sua correspondente consequência lógica da variável de saída.

Utilizando as definições anteriormente descritas para as partições fuzzy, o exemplo acima seria transcrito como: *"se o veículo estiver localizado na posição central (CE) e estiver rotacionado um pouco à direita, em ângulo menor que 90º (RV), então a roda deve ser girada um pouco em um ângulo positivo (OS), para redirecionar o veículo"*. Isto é, a regra para o controlador fuzzy seria interpretada como: "**se** (*posição x* é CE) **e** (*ângulo do veículo* é RV) **então** (*ângulo da roda* é OS)".

A **Tabela 2** descreve uma configuração completa utilizada para esta base de regras, que determina o comportamento do algoritmo do veículo. Observe que o número de regras é resultado da combinação direta entre o número de conjuntos fuzzy disponíveis nas variáveis de entrada. Portanto, uma vez que a variável posição x possui cinco conjuntos, e a variável ângulo do veículo φ possui sete conjuntos, teremos um total de 35 regras descritas para definir a cobertura de regras completa para todas as situações possíveis. A **Tabela 2** deve ser lida da seguinte maneira:

- Se (x é LE) e (φ é RB) então (θ é PS);
- Se (x é LC) e (φ é VE) então (θ é NM).

x / φ	**LE**	**LC**	**CE**	**RC**	**RI**
RB	PS	PM	PM	PB	PB
RU	NS	OS	PM	PB	PB
RV	NM	NS	OS	PM	PB
VE	NM	NM	ZE	PM	PM
LV	NB	NM	NS	OS	PM
LU	NB	NB	NM	NS	OS
LB	NB	NB	NM	NM	NS

Tabela 2. Base de regras para o sistema de inferências, combinando as entradas para determinar o valor de θ.

Cabe destacar que no modelo do simulador ocorre um retro-processamento. Ou seja, a cada iteração, o sistema de inferências avalia as regras descritas para a posição atual das variáveis de entrada e retorna um valor de saída. As variáveis de entrada (posição e ângulo do veículo) são atualizadas em função do ângulo da roda calculado pelo controlador, e no passo seguinte são novamente submetidas para processamento pelo sistema de inferências. Por isso, durante o trajeto do veículo, diferentes regiões e regras serão ativadas, até que a condição de parada do veículo seja atendida.

A base de regras é inicialmente construída a partir da representação do raciocínio de um especialista, e deve ser ajustada em sucessivas iterações até que os resultados sejam satisfatórios. Portanto, em sistemas fuzzy, é fundamental que sejam executados diversos ciclos de testes e ajustes nos parâmetros de configuração para que se possa obter os resultados desejados. Observado os resultados dos testes, deve-se avaliar se a base de regras está correta e oportunamente promover ajustes no formato ou no intervalo das partições nebulosas para as variáveis de entrada e saída.

Implementação do controlador com JFuzzyLogic

O processamento dos cálculos da inferência envolve diversos cálculos matemáticos, que ficam a cargo do JFuzzyLogic [6], um componente *open source* inteiramente escrito em Java e que implementa um *Fuzzy Logic Controller* (FLC) completo, baseado na especificação IEC 61131-7. Sua utilização objetiva evitar reescrever todo um conjunto de algoritmos e modelos matemáticos complexos necessários ao processamento das informações fuzzy, adotando uma ferramenta robusta e amplamente utilizada, concentrando-se apenas no assunto principal do artigo.

As definições dos parâmetros do controlador fuzzy são realizadas através de um arquivo texto no formato FLC, que será utilizado pelo JFuzzyLogic. O arquivo FLC define as variáveis e partições fuzzy, além da base de regras, sendo composto pelos seguintes blocos:

- Um bloco para declaração das variáveis de entrada e saída;
- Interface *fuzzification*, que traduz os valores numéricos das entradas em representações fuzzy;
- Interface *defuzzification*, que traduz os valores fuzzy em uma saída com valor numérico.

A **Listagem 1** apresenta o formato geral de um arquivo do tipo FLC. Cada bloco de definições do arquivo FLC será descrito nos tópicos a seguir.

Listagem 1. Formato geral de um arquivo FLC.

```
FUNCTION_BLOCK simulador    // Início do bloco de definições

      VAR_INPUT              // Definição das variáveis de entrada
          nome_variavel_entrada: REAL;
      END_VAR

      VAR_OUTPUT             // Definição das variáveis de saída
          nome_variavel_saida: REAL;
      END_VAR

      FUZZIFY nome_variavel_entrada
          // definição das partições fuzzy e seus intervalos para cada variável de entrada
          TERM PARTICAO_X := (0.0, 0) (10.0, 1) (20.0, 0)
      END_FUZZIFY

      DEFUZZIFY nome_variavel_saida
          // definição das partições fuzzy e seus intervalos para cada variável de saída

          TERM PARTICAO_Y := (0.0, 0) (50.0, 1) (100.0, 0) ;
          METHOD : COG;  // Método de defuzzificação (Padrão é o Centro de Gravidade)
          DEFAULT := 0;  // Valor default caso nenhuma regra seja ativada
      END_DEFUZZIFY

      RULEBLOCK No1
          // Definição do conjunto de regras para o controlador Fuzzy. Este bloco irá descrever
          // as correlações entre as partições da variável de entrada com uma partição da variável
          // de saída

          AND : MIN;  // Método MIN utilizado no processamento do operador lógico AND
          ACT : MIN;  // Método de ativação
          ACCU : MAX; // método de acumulação

          // Início da descrição de cada regra
          // RULE 1 : IF variavel_entrada1 IS PARTICAO1 AND variavel_entrada1 IS particao2  THEN variavel_saida IS particaoX;

      END_RULEBLOCK

  END_FUNCTION_BLOCK
```

Definições do arquivo FLC

O primeiro bloco do arquivo FLC declara as variáveis de entrada (VAR_INPUT) e de saída (VAR_ OUTPUT) que serão consideradas pelo controlador, atribuindo-lhes um nome que será utilizado nos demais blocos do arquivo e também no código Java para a passagem de parâmetros durante o cálculo da inferência. No caso do simulador de estacionamento, as variáveis de entrada são *posicao_x* e *angulo_veiculo* e a variável de saída é *angulo_roda* (**Listagem 2**).

Listagem 2. Declaração das variáveis de entrada e saída no arquivo FLC.

```
VAR_INPUT    // Define variáveis de entrada
   posicao_x: REAL;
   angulo_veiculo: REAL;
  END_VAR

  VAR_OUTPUT   // Define variáveis de saída
   angulo_roda: REAL;
  END_VAR
```

O segundo bloco (FUZZIFY) define quais parâmetros serão considerados na etapa de fuzzificação para cada partição, ou seja, como os valores de entrada serão transformados em representações fuzzy. Esta declaração é realizada a partir das definições dos intervalos para os conjuntos fuzzy de cada variável, conforme anteriormente apresentado. Por exemplo, a variável de entrada *posicao_x* é composta por cinco conjuntos fuzzy: à esquerda (LE), centro-esquerda (LC), centro (CE), centro-direita (RC) e direita (RI). Cada conjunto possui uma função de pertinência que define seu grau de inclusão dentro de um intervalo válido. Pode-se observar na **Figura 11** que a função de pertinência para o conjunto CE inicia com um valor zero no ponto x=45, chega ao ápice (com valor 1) no ponto x=50, e retorna ao valor zero no ponto x=55. Portanto, a função de pertinência triangular para este conjunto é definida pelos pontos em que a função possui o valor de mínimo inicial, máximo e mínimo final: {45; 50; 55}. No arquivo FLC, o conjunto será descrito pela seguinte representação: *TERM CE := (45.0, 0) (50.0, 1) (55.0, 0);*.

As definições dos conjuntos fuzzy para as variáveis de entrada *posicao_x* e *angulo_veiculo* estão descritas na **Tabela 1**, e sua correspondente declaração no arquivo FLC será realizada conforme a **Listagem 3**.

Listagem 3. Descrição dos intervalos para cada conjunto fuzzy considerado pelas variáveis de entrada.

```
FUZZIFY posicao_x
   TERM LE := (0.0, 0) (0.0, 1) (10.0, 1) (35.0, 0) ;
   TERM LC := (30.0, 0) (40.0, 1) (50.0, 0) ;
   TERM CE := (45.0, 0) (50.0, 1) (55.0, 0) ;
   TERM RC := (50.0, 0) (60.0, 1) (70.0, 0) ;
   TERM RI := (65.0, 0) (90.0, 1) (100.0, 1) (100.0, 0) ;
  END_FUZZIFY

  FUZZIFY angulo_veiculo
   TERM RB := (-90.0, 0) (-45.0, 1) (15.0, 0) ;
   TERM RU := (-15.0, 0) (30.0, 1) (60.0, 0) ;
   TERM RV := (45.0, 0) (67.0, 1) (90.0, 0) ;
   TERM VE := (75.0, 0) (90.0, 1) (105.0, 0) ;
   TERM LV := (90.0, 0) (112.0, 1) (135.0, 0) ;
   TERM LU := (120.0, 0) (150.0, 1) (195.0, 0) ;
   TERM LB := (165.0, 0) (225.0, 1) (270.0, 0) ;
  END_FUZZIFY
```

O próximo bloco (DEFUZZIFY) corresponde às definições que serão utilizadas na etapa de defuzzificação, ou seja, como as informações processadas pelo controlador fuzzy serão traduzidas em números precisos a serem fornecidos como saída. Este bloco é descrito da mesma maneira como o anterior, especificando as representações do intervalo de cada conjunto fuzzy utilizado na variável de saída (**Tabela 1**), conforme a **Listagem 4**. Este bloco também define qual método de defuzzificação será utilizado para ponderar o valor final da inferência a partir das regras ativadas. Neste exemplo, utilizamos o método Centro de Gravidade (COG – *Center of Gravity*), que melhor atende a maioria dos casos. Para necessidades mais específicas, a API do JFuzzyLogic oferece outros métodos disponíveis, que podem ser consultados em sua documentação.

Listagem 4. Descrição dos conjuntos e intervalos considerados para a variável de saída.

```
DEFUZZIFY angulo_roda
   TERM NB := (-30.0, 0) (-30.0, 1) (-15.0, 0) ;
   TERM NM := (-25.0, 0) (-15.0, 1) (-5.0, 0) ;
   TERM NS := (-10.0, 0) (-5.0, 1) (0.0, 0) ;
   TERM ZE := (-5.0, 0) (0.0, 1) (5.0, 0) ;
   TERM PS := (0.0, 0) (5.0, 1) (10.0, 0) ;
   TERM PM := (5.0, 0) (15.0, 1) (25.0, 0) ;
   TERM PB := (15.0, 0) (30.0, 1) (30.0, 1) ;
   METHOD : COG; // Método de defuzzificação. Utilizando Center of Gravity
   DEFAULT := 0; // Valor a ser utilizado caso nenhuma regra seja ativada
  END_DEFUZZIFY
```

A quarta e última etapa corresponde à definição das regras que representam as implicações lógicas entre os conjuntos fuzzy das variáveis de entrada e da variável de saída. Esta base de regras representa o comportamento do algoritmo do controlador, e são descritas a partir da composição apresentada pela **Tabela 2**. A sintaxe de declaração da regra segue o formato if-then: if posicao_x is LE and angulo_veiculo is RB then angulo_roda is OS. A **Listagem 5** apresenta as 35 regras que compõem a base de conhecimento para as inferências do controlador fuzzy.

Ao utilizar operadores lógicos AND e OR, é necessário especificar o método que será utilizado para o processamento das regras. É comum utilizar o método MIN para o operador AND, e o método MAX para o operador OR. Devem-se definir também os métodos de ativação e de acumulação, onde na maioria dos casos os valores default que vêm disponibilizados no template do arquivo FLC podem ser utilizados, ou seja, MIN para ativação e MAX para acumulação. O método de ativação define como as partições de entrada de uma regra ativada afetarão a saída correspondente. O método de acumulação determina o fator de ponderação a ser aplicado quando múltiplas regras são ativadas. Além destes, outros métodos também são disponibilizados pelo JFuzzyLogic em sua documentação, e podem ser utilizados em necessidades mais específicas de customização.

Listagem 5. Declaração do mapeamento das regras para o controlador fuzzy.

```
RULEBLOCK No1
   AND : MIN;  // Método MIN utilizado no processamento do operador lógico AND
   ACT : MIN;  // Método de ativação
   ACCU : MAX; // método de acumulação

   RULE 1 : IF Posicao_x IS LE AND Angulo_veiculo IS RB  THEN Angulo_Roda IS PS;
   RULE 2 : IF Posicao_x IS LE AND Angulo_veiculo IS RU  THEN Angulo_Roda IS NS;
   RULE 3 : IF Posicao_x IS LE AND Angulo_veiculo IS RV  THEN Angulo_Roda IS NM;
   RULE 4 : IF Posicao_x IS LE AND Angulo_veiculo IS VE  THEN Angulo_Roda IS NM;
   RULE 5 : IF Posicao_x IS LE AND Angulo_veiculo IS LV  THEN Angulo_Roda IS NB;
```

```
  RULE 6 : IF Posicao_x IS LE AND Angulo_veiculo IS LU  THEN Angulo_Roda IS NB;
  RULE 7 : IF Posicao_x IS LE AND Angulo_veiculo IS LB  THEN Angulo_Roda IS NB;
  RULE 8 : IF Posicao_x IS LC AND Angulo_veiculo IS RB  THEN Angulo_Roda IS PM;
  RULE 9 : IF Posicao_x IS LC AND Angulo_veiculo IS RU  THEN Angulo_Roda IS PS;
  RULE 10 : IF Posicao_x IS LC AND Angulo_veiculo IS RV  THEN Angulo_Roda IS NS;
  RULE 11 : IF Posicao_x IS LC AND Angulo_veiculo IS VE  THEN Angulo_Roda IS NM;
  RULE 12 : IF Posicao_x IS LC AND Angulo_veiculo IS LV  THEN Angulo_Roda IS NM;
  RULE 13 : IF Posicao_x IS LC AND Angulo_veiculo IS LU  THEN Angulo_Roda IS NB;
  RULE 14 : IF Posicao_x IS LC AND Angulo_veiculo IS LB  THEN Angulo_Roda IS NB;
  RULE 15 : IF Posicao_x IS CE AND Angulo_veiculo IS RB  THEN Angulo_Roda IS PM;
  RULE 16 : IF Posicao_x IS CE AND Angulo_veiculo IS RU  THEN Angulo_Roda IS PM;
  RULE 17 : IF Posicao_x IS CE AND Angulo_veiculo IS RV  THEN Angulo_Roda IS PS;
  RULE 18 : IF Posicao_x IS CE AND Angulo_veiculo IS VE  THEN Angulo_Roda IS ZE;
  RULE 19 : IF Posicao_x IS CE AND Angulo_veiculo IS LV  THEN Angulo_Roda IS NS;
  RULE 20 : IF Posicao_x IS CE AND Angulo_veiculo IS LU  THEN Angulo_Roda IS NB;
  RULE 21 : IF Posicao_x IS CE AND Angulo_veiculo IS LB  THEN Angulo_Roda IS NM;
  RULE 22 : IF Posicao_x IS RC AND Angulo_veiculo IS RB  THEN Angulo_Roda IS PB;
  RULE 23 : IF Posicao_x IS RC AND Angulo_veiculo IS RU  THEN Angulo_Roda IS PB;
  RULE 24 : IF Posicao_x IS RC AND Angulo_veiculo IS RV  THEN Angulo_Roda IS PM;
  RULE 25 : IF Posicao_x IS RC AND Angulo_veiculo IS VE  THEN Angulo_Roda IS PM;
  RULE 26 : IF Posicao_x IS RC AND Angulo_veiculo IS LV  THEN Angulo_Roda IS PS;
  RULE 27 : IF Posicao_x IS RC AND Angulo_veiculo IS LU  THEN Angulo_Roda IS NS;
  RULE 28 : IF Posicao_x IS RC AND Angulo_veiculo IS LB  THEN Angulo_Roda IS NM;
  RULE 29 : IF Posicao_x IS RI AND Angulo_veiculo IS RB  THEN Angulo_Roda IS PB;
  RULE 30 : IF Posicao_x IS RI AND Angulo_veiculo IS RU  THEN Angulo_Roda IS PB;
  RULE 31 : IF Posicao_x IS RI AND Angulo_veiculo IS RV  THEN Angulo_Roda IS PB;
  RULE 32 : IF Posicao_x IS RI AND Angulo_veiculo IS VE  THEN Angulo_Roda IS PM;
  RULE 33 : IF Posicao_x IS RI AND Angulo_veiculo IS LV  THEN Angulo_Roda IS PS;
  RULE 34 : IF Posicao_x IS RI AND Angulo_veiculo IS LU  THEN Angulo_Roda IS PS;
  RULE 35 : IF Posicao_x IS RI AND Angulo_veiculo IS LB  THEN Angulo_Roda IS NS;

 END_RULEBLOCK

 END_FUNCTION_BLOCK   // Final do bloco de definições do algoritmo
```

Para facilitar a descrição do arquivo FLC, o JFuzzyLogic oferece também um plugin para o Eclipse, com alguns recursos muito úteis ao desenvolvedor, como: autocomplete, código com sintaxe colorida, visualização gráfica das partições das variáveis, etc. As instruções para instalação e uso do plugin do Eclipse está disponível no manual do JFuzzyLogic [7].

Implementação do controlador em Java

Uma vez concluídas as definições descritas no arquivo FLC, a implementação da classe Java para o controlador fuzzy torna-se muito simples. O primeiro passo é carregar o arquivo FLC e obter uma instância da classe **FIS**, que representa o controlador fuzzy. Esta mesma instância será utilizada durante toda a simulação. Para isto, utiliza-se o método **load()**, especificando o local do arquivo FLC:

```
// Instancia o controlador a partir das definições do arquivo FLC

 FIS fis = FIS.load(“estacionamento.fis”);
```

Opcionalmente, a instância de **FIS** também pode ser carregada dinamicamente, a partir do conteúdo com as definições do arquivo FLC, permitindo maior flexibilidade, uma vez que ele pode ser gerado dinamicamente pela aplicação:

```
File arquivoFis = new File(SimuladorEstacionamento.class.getResource("estacionamento.fis").toURI());

 String conteudoArquivoFis = new String(Files.readAllBytes(arquivoFis.toPath()));
 fis = FIS.createFromString(conteudoArquivoFis, true);
```

Em cada iteração da simulação, os dados referentes à posição e ângulo atuais do veículo são passados para o controlador utilizando o método **setVariable()** da classe **FIS**. O primeiro parâmetro do método corresponde ao nome da variável, como declarado no arquivo FLC, e o segundo é o valor de entrada para esta variável. Após definir os parâmetros de entrada, deve-se chamar o método **evaluate()** da instância **FIS** do controlador, que irá processar os cálculos para a inferência, de acordo com a base de conhecimento definida no arquivo FLC. O valor calculado para a saída, correspondente ao ângulo da roda, é obtido através do método **getVariable(“ângulo_roda”).getValue()**.

A passagem dos parâmetros para a classe **FIS** e a lógica para o cálculo da inferência em cada passo estão encapsuladas pelo método **calculaInferencia()**, da classe **Controlador**. Este método recebe como parâmetros a posição e o ângulo atuais do veículo, e retorna o valor do ângulo da roda calculado pelo controlador (**Listagem 6**).

O código completo da classe **Controlador**, responsável por instanciar a classe **FIS** e calcular a inferência é apresentado **Listagem 7**.

Listagem 6. Passagem de parâmetros e chamada ao cálculo da inferência pelo controlador FIS.

```
public double calculaInferencia(double posicaoX, double anguloVeiculo){
      this.fis.setVariable("Posicao_x", posicaoX);
      this.fis.setVariable("Angulo_veiculo", anguloVeiculo);

      this.fis.evaluate();
      double anguloRoda = this.fis.getVariable("Angulo_Roda").getValue();
      return anguloRoda;
  }
```

Listagem 7. Classe responsável pelo Controlador.

```
package br.nebula.veiculo;

  import java.io.File;
  import java.nio.file.Files;
  import java.util.logging.Level;
  import java.util.logging.Logger;
  import net.sourceforge.jFuzzyLogic.FIS;

  public class Controlador {

      private FIS fis;

      public Controlador() {
          iniciaInferencia();
      }

      private void iniciaInferencia() {
          try {
              File arquivoFis = new          File(SimuladorEstacionamento.class.getResource("estacionamento.fis").toURI());
              String conteudoArquivoFis = new String(Files.readAllBytes(arquivoFis.toPath()));

              fis = FIS.createFromString(conteudoArquivoFis, true);
              System.out.println("Instancia de inferencias carregada com sucesso");
          } catch (Exception ex) {
              Logger.getLogger(SimuladorEstacionamento.class.getName()).log(Level.SEVERE, "Erro ao abrir o arquivo", ex);
          }
      }

      public double calculaInferencia(double posicaoX, double anguloVeiculo){
          this.fis.setVariable("Posicao_x", posicaoX);
          this.fis.setVariable("Angulo_veiculo", anguloVeiculo);
          this.fis.evaluate();
          double anguloRoda = this.fis.getVariable("Angulo_Roda").getLatestDefuzzifiedValue();
          return anguloRoda;
      }
  }
```

Para realizar a movimentação do veículo, o valor do ângulo da roda retornado pelo controlador deve ser utilizado para calcular o novo ângulo de rotação e a nova posição do veículo, que será deslocado em uma distância fixa em cada iteração. Este cálculo é realizado conforme descrito pela equação apresentada na **Figura 14**. A **Listagem 8** apresenta o código em Java do simulador, responsável por reposicionar e rotacionar o veículo em função do valor retornado pela inferência a cada passo.

Listagem 8. Movimentação do veículo, a partir do valor retornado pelo sistema de inferências.

```
anguloRoda = controlador.calculaInferencia(this.veiculo.getPosX(),this.veiculo.getAngulo());

int delta = 10; // constante que indica a distância fixa deslocada a cada passo

this.veiculo.setAngulo( this.veiculo.getAngulo() + anguloRoda);
double angulo_rad = (Math.PI * this.veiculo.getAngulo()) / 180; // obtendo angulo em radianos
// calculando nova posição x e y, deslocando o veículo na distância delta
double posicaoX = this.veiculo.getPosX() + (delta * Math.cos(angulo_rad));
double posicaoY = this.veiculo.getPosY() + (delta * Math.sin(angulo_rad));
this.veiculo.setPosXY(posicaoX , posicaoX );
this.veiculo.rotate(anguloVeiculo);
```

Este simples algoritmo, implementado com o componente JFuzzyLogic, é suficiente para que o veículo se desloque em sua trajetória em direção ao local pré-determinado. Cabe observar que toda a modelagem do algoritmo inteligente foi definida no arquivo FLC, restando ao código Java apenas a passagem dos parâmetros e as atualizações da interface em função dos valores calculados pelo controlador

fuzzy.

Com o controlador fuzzy implementado, podemos utilizar a interface gráfica do simulador para executar algumas rodadas de simulação, considerando diferentes posições e ângulos iniciais para o veículo. Habilitando o recurso de rastro, podemos observar que as trajetórias descritas durante as simulações foram satisfatoriamente encaminhadas para a posição final sinalizada pelo local de estacionamento, conforme ilustrado na **Figura 20**.

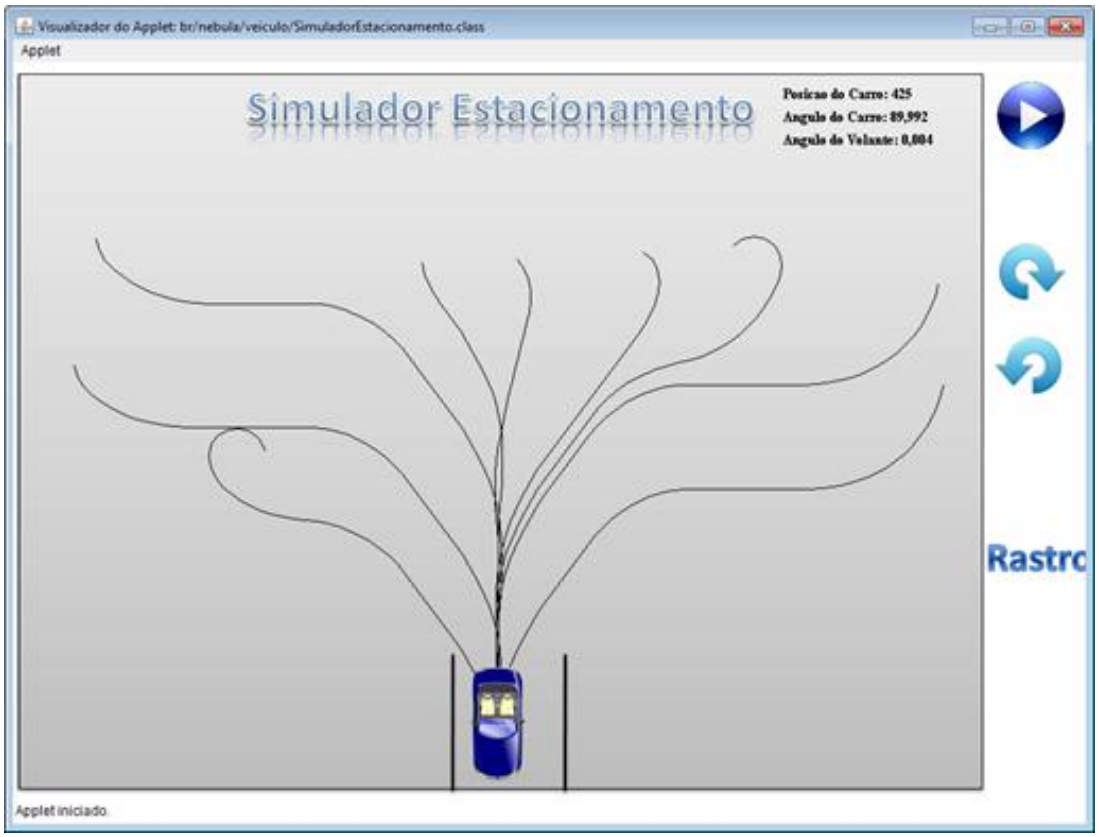

Figura 20. Interface gráfica ilustrando as trajetórias do veículo em simulações utilizando diferentes configurações iniciais.

Após uma bateria de experimentos para testar o controlador através do simulador, o controlador está pronto para se tornar parte de um sistema de controle embarcado.

Uma interessante aplicação futura seria implementar este simulador em um modelo físico, utilizando uma placa Arduino como integrador de hardware e sensores de posicionamento para determinar as variáveis de entrada. As mesmas definições para o controlador fuzzy poderiam ser reutilizadas neste caso.

O exemplo do sistema autônomo de controle de direção, apoiado por lógica fuzzy, tenta elucidar o processo de modelagem e codificação de uma solução de automação simplificada. Soluções similares são muito encontradas em plantas industriais, onde um controlador é projetado e um simulador é construído para testar a eficiência do controlador ainda em ambiente computacional. Uma vez homologado o controlador em ambiente simulado, ele é promovido para os experimentos em ambiente real. Após o processo de homologação, ele é incorporado a um software ou hardware para uso na linha de produção.

Alguns processos de decisão são próprios do raciocínio humano, normalmente entendidos pela maioria das pessoas como impossíveis de serem absorvidos por um processo automático. Noção válida, porém superada, pois há algum tempo a lógica fuzzy já existe para contrariar este entendimento. Assim, alguns processos de decisão baseados em incerteza podem, através da lógica fuzzy, ser modelados e aplicados a software e hardware de modo a serem automatizados.

A automação desejada, seja ela para um sistema corporativo, seja ela para um sistema de controle de automação industrial, deve ser elaborada por um projetista (desenvolvedor) de forma a guiar o especialista na definição dos conjuntos fuzzy e das regras. Uma vez munido destes insumos, o projetista pode implementar a lógica em uma linguagem como Java utilizando uma biblioteca como a JFuzzyLogic.

A partir dos exemplos práticos apresentados aqui, pode-se observar que é possível construir soluções com lógica nebulosa mesmo com pouco conhecimento matemático nesta área. Inúmeras necessidades podem ser solucionadas desta forma, basta que o projetista compreenda que algumas decisões humanas são suscetíveis de serem automatizadas. Além disso, técnicas de inteligência

computacional baseadas em lógica fuzzy podem ser utilizadas em diversas situações na área de computação, especialmente em modelos que possuam alto grau de incerteza e imprecisão.

Referências

[1] Livro *Fuzzy logic: a practical approach*.

McNeill, F. M., & Thro, E. (2014).Fuzzy logic: a practical approach. Academic Press.

[3] Exemplos de sistemas embarcados extraídos do Livro *Fuzzy logic for embedded systems applications*.

Ibrahim, A. (2003).Fuzzy logic for embedded systems applications. Newnes.

Links

[2] Artigo *Real-Life Applications of Fuzzy Logic*.
http://www.hindawi.com/journals/afs/2013/581879/

[4] Capítulo do livro com o exemplo de sistema de RH.
http://ptgmedia.pearsoncmg.com/images/0135705991/samplechapter
/0135705991.pdf

[5] Artigo sobre lógica fuzzy, que menciona o clássico problema do estacionamento de um veículo.
http://www2.ica.ele.puc-rio.br/Downloads/41/LN-Sistemas%20Fuzzy.pdf

[6] Página do componente JFuzzyLogic.
http://jfuzzylogic.sourceforge.net/html/index.html

[7] Manual do componente JFuzzyLogic, com instruções sobre o plugin para Eclipse.
http://jfuzzylogic.sourceforge.net/html/manual.html#plugin

DevMedia