Relatório Técnico U1

Aplicação de um Sistema de Inferência Fuzzy para Reconhecimento de Padrões em Sinais Neurais Sintéticos

- ## 1.1. Objetivo do Projeto:

Desenvolver um algorítmo, em Linguagem C, que permita a aplicação de um sistema de inferência fuzzy para reconhecimento de padrões em sinais neurais sintéticos, produzidos por um algoritmo que será inclusive o trabalho da unidade I, bem como em sinais neurais obtidos de um encefalograma feito em um rato, comparando-os a posterior e verificando, através de um sistema fuzzy, uma ferramenta muito poderosa para análises desse tipo, o que o resultado significa e implica de maneira geral.

- ## 1.2. Justificativa

Um dos principais motivadores que me fez optar por esse projeto também é justamente uma das razões que me fez ingressar nessa graduação de Bacharelado em Tecnologia da Informação: a interdisciplinaridade entre as áreas de Saúde e Tecnologia, tais como interface Cérebro-Máquina e Lógica Fuzzy, bem como o desejo de aplicar tais conceitos nesse projeto que, apesar de ser difícil e desafiador, devido à complexidade e maneira ao qual está sendo construído pelas 3 unidades, o mesmo é direcionado para áreas promissoras, complexas e que muito me agradam para trabalhar e estudar no futuro.

- ## 1.2. Problema Resolvido:

O principal desafio do projeto seria como resolver o problema de interpretar dados contínuos, ruidosos e por vezes ambíguos usando lógica computacional no caso dos sinais neurais orgânicos, bem como a regulagem adequada da lógica fuzzy, pois a mesma não tem uma generalização universal que possa ser usada. Tal situação é válida pois, dessa forma, é possível traduzir um grande fluxo de dados brutos em classificações mais precisas e de fácil entendimento, principalmente pelo fato da fuzzificação não trabalhar como na lógica booleana, ou seja, apenas com 0 e 1, e sim com qualquer número real entre 0 e 1, que é mais realista pensando na interpretação desse tipo de dado, como por exemplo: um comando para ir a esquerda, pela lógica booliana, significa virar totalmente para esquerda (1, verdadeiro) ou não (0, falso), enquanto na fuzzy, o comando para ir a esquerda pode variar entre total (1), muito (0.8), metade (0.5)…, portanto, muito mais orgânico e realístico para o que é almejado, ainda que dependa da análise humana ao final.

Desta forma, se tudo ocorrer como planejado, o projeto vai conseguir entregar um resultado que permita inferir, depois de todo tratamento e processamento, um resultado que permitirá dizer o estado que se encontra um indivíduo que apresenta aqueles dados em seu encefalograma, podendo a posterior ser expandido a uma imensa variadade de situações (dia-a-dia, internações, stress pós-traumático, pacientes acometidos com doenças neurológicas, etc.)

ANÁLISE TÉCNICA

Metodologia: Ferramentas Utilizadas

O desenvolvimento foi realizado utilizando o **Visual Studio Code** como ambiente de edição, com o **compilador GCC** para compilação, tendo sido feito testes em ambientes **Windows e Linux**. A escolha dessas ferramentas se deu por sua ampla utilização na indústria, suporte à linguagem C e compatibilidade com bibliotecas de uso geral, inclusive para a implementação da fuzzificação que será feito nas unidades II e III.

- ## 2.2. Aplicação dos Conceitos da U1

A seguir detalharei, de forma superficial, de que maneira usei os comandos abordados na primeira unidade para essa etapa inicial do projeto.

- **Comandos Condicionais (If/Else):**

  As estruturas if/else foram as ferramentas para implementar essa capacidade do programa em tomar decisões, sendo o uso mais visível na função `interpretarSinal`, onde o valor do sinal era comparado com as constantes `LIMIAR_ALERTA` e `LIMIAR_REPOUSO` para decidir qual classificação retornar. Também foi importante para o comando `executarPulsoSimulacao,` garantindo que o sinal nunca saísse do intervalo válido [0, 100].

- **Comando de Repetição (For):**

  Propósito Estratégico: O comando `for` foi o mecanismo que criou essa linha do tempo para a simulação, pois se assim não fosse, haveria apenas uma única estatística, que não poderia ser comparada ou atribuído sentido. O laço principal em `main`, `for (int i = 0; i < NUMERO_DE_PULSOS; i++)`, é um dos pilares do programa. Ele garantiu que o processo de gerar, analisar, armazenar e exibir um sinal fosse repetido um número fixo de vezes.

- **Funções:**

A decisão de modularizar o código em múltiplas funções foi a principal tática para garantir a organização, legibilidade e estilistica do projeto. Em vez de um bloco de código único na `main`, foi criado subgrupos, onde cada função tem uma missão única e bem definida. Essa divisão de trabalho tornou a função `main` mais enxuta, que apenas orquestra as chamadas das outras linhas de comando, tornando o fluxo geral fácil e intuitivo, sendo essa as 4 funções

  - **`interpretarSinal()`**:Sua única missão era receber um valor de sinal bruto e, com base em regras predefinidas, retornar uma classificação (`const char*`). Isso isolou completamente a lógica de decisão do resto do programa.

- **`executarPulsoSimulacao()`**: Encapsulou a lógica de criar a variação aleatória, aplicar essa variação ao sinal anterior e garantir que o novo valor não ultrapassasse os limites de 0 e 100.
- **`exibirEstadoSimulacao()`**: Apresentar os dados de um único pulso de forma clara e formatada no terminal, separando a lógica de processamento da lógica de exibição.
- **`exibirResumoHistorico()`**:Exibir um resumo de seus registros no final da simulação.

- **Vetores:**

  O vetor foi a estrutura de dados escolhida para atuar como a memória de longo prazo do programa. Pois o vetor `int historicoDeSinais[NUMERO_DE_PULSOS]` foi declarado para ter exatamente o tamanho da nossa simulação. A cada iteração `i` do laço `for`, o `sinalAtual` gerado era armazenado na posição `historicoDeSinais[i]`.

  Assim sendo, ao final do laço, não tínhamos apenas o último resultado, mas uma série temporal completa e ordenada de toda a atividade do “cérebro sintético”.

• 2.3. Estrutura de Dados

Neste tópico, falarei de cada ponto de forma individual, pois cada um exerce função singular e igualmente importante para o projeto em questão.

- **`int sinalAtual`:**
  - **Descrição:** Principal variável de estado da simulação. Representa o valor da medição em um determinado instante. Sua característica mais importante era a não aleatoriedade, ou seja**,** o valor de `sinalAtual` no final de um pulso era usado como base para calcular o valor no pulso seguinte, criando uma fluidez “orgânica” para o sinal.
- **`double tempoSimulado`:**
  - **Descrição:** Esta variável atuou como o cronômetro do programa. A cada pulso, ela era incrementada pela constante `PASSO_DE_TEMPO`. Seu propósito era dar contexto a cada medição, transformando uma simples sequência de “x” pulsos em uma simulação que ocorria ao longo de “y” segundos virtuais. O uso do tipo `double` foi para permitir incrementos fracionários, aumentando o realismo por se tratar de uma “onda cerebral” sintética.
- **`int historicoDeSinais[NUMERO_DE_PULSOS]`:**
  - **Descrição:** Como detalhado acima, este vetor foi a principal estrutura de armazenamento. Era um array de inteiros, projetado para ser o arquivo histórico da simulação. Cada célula do vetor corresponde a uma medição feita naquele momento, armazenando o valor do `sinalAtual` naquele instante.

3. Implementação e Reflexão

3.1. Dificuldades Encontradas

Não tive tantos problemas, em alguns casos a copilação dava erro por algum erro gramatical meu, em outros o printf não funcionava da forma como eu gostaria, mas talvez minha principal dor de cabeça tenha sido com relação ao nome dos arquivos, por algum motivo que ainda tenho dificuldade de compreensão, ao utilizar acentos, barra de espaço ou qualquer notação desse tipo, eu tinha muita dificuldade em fazer o arquivo funcionar.

3.2. Soluções Implementadas

O erro gramatical era resolvido pois o próprio terminal, via de regra, dizia qual parte não estava sendo compreendida para leitura, então a solução era correções pontuais; o printf foi resolvido ao pesquisar e utilizar um outro comando, chamado “fflush(stdout)”, que “forçou” o funcionamento; sobre o problema acerca do nome dos arquivos, a solução foi adotar um padrão de nomenclatura simples (sem acentos ou espaços, tudo minúsculo), garantindo a máxima compatibilidade com o compilador GCC e as ferramentas de linha de comando.

3.3. Organização do Código

*E*m muitos momentos, enquanto eu editava e, principalmente, testava, quando algo não saia exatamente como eu gostaria, no caso, sem ser um erro de copilação e sim uma resposta diferente daquilo que eu almejava, eu tinha MUITA dificuldade em encontrar o problema, a ponto de em, pelo menos, uma oportunidade eu precisar recomeçar do zero pois sequer entendia onde estava o erro e como poderia consertar. Eu diria que para projetos pelo menos medianos, a organização do código é mais do que uma mera formalidade ou embelezamento do código, é uma necessidade para que o mesmo tenha vida útil ou possa ser aprimorado. Ao utilizar várias funções, eu pude deixar o “Main” para chamar as funções de fora definindo-as e executando-as na ordem correta, como um cérebro, enquanto as funções externas eram o resto do corpo.

3.4. Conclusão: Aprendizados e Melhorias Futuras

Aprendi bastante com esse projeto, mas, na verdade, ele ainda está engatinhando, visto que ainda é necessário muita coisa ser feita para ele de fato ser aquilo que almejo, dentre essas coisas, posso citar: a utilização não apenas de sinais neurais sintéticos no projeto, como também a adição de sinais neurais reais, obtidos de um eletroencefalograma de um rato, de maneira que eu possa comparar os resultados; como tratarei esses dados, pois diferente do sintético, possui ruído e outras coisas que precisam ser contornadas; o procedimento de fuzzificação, que ainda preciso estudar como farei, bem como se existe alguma biblioteca específica que eu possa usar ou se precisarei fazer isso do zero.