UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

TAINAN MAURI

Aplicação de controladores *Fuzzy para* o equilíbrio frontal de um veículo de duas rodas

Porto Alegre
2018

TAINAN MAURI

Aplicação de controladores *Fuzzy* para o equilíbrio frontal de um veículo de duas rodas

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Engenheiro Eletricista.

ORIENTADOR: Prof. Dr. Jeferson Vieira Flores

Porto Alegre
2018

TAINAN MAURI

Aplicação de controladores *Fuzzy* para o equilíbrio frontal de um veículo de duas rodas

Este projeto foi julgado adequado para a obtenção dos créditos da Disciplina Projeto de Diplomação do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Jeferson Vieira Flores, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Banca Examinadora:

Prof. Dr. Jeferson Vieira Flores, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Profª. Drª. Lucíola Campestrini, UFRGS
Doutora pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Ivan Müller, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Chefe do DELET: _____
Prof. Dr. Ály Ferreira Flores Filho

Porto Alegre
2018

AGRADECIMENTOS

Agradeço, em especial, a quatro grupos de pessoas, responsáveis pela minha formação (profissional e pessoal) e por todas as experiências vividas nestes 5 anos dentro da Engenharia Elétrica da UFRGS.

Primeiro à minha família, em especial à minha mãe, pois esta é um espelho para mim e ajudou a moldar o meu caráter, além de ter despertado meu interesse por exatas desde cedo.

O segundo grupo que merece meus agradecimentos é o de amigos que fiz dentro do curso: Bruno Zanchetta, Carlos Conrado, Daniel Fiala, Élnatan Ariotti, Gabriel Aguiar, Givaldo Vieira, Luciano Bongiorni, Maylan Buffon e Vinícius Batista. O convívio intenso e diário com estes me ajudou a sempre levar o curso com leveza e alegria, além de me propiciar inúmeros momentos de felicidade os quais guardarei para a vida inteira.

Em terceiro lugar, a todos os professores, sem exceção, os quais tiveram profunda influência em minha formação profissional, em especial o professor Jeferson Flores, o qual me ajudou com a orientação deste trabalho e despertou meu interesse pelos sistemas de controle.

Por fim, a todas as pessoas as quais tive o prazer de conviver dentro desta universidade, assim como a todos os brasileiros que financiaram minha educação. Que um dia eu possa retribuir, de alguma forma, o bem que estes me fizeram.

RESUMO

O objetivo deste trabalho de conclusão de curso é implementar três diferentes controladores para manter o equilíbrio frontal de um veículo de duas rodas, bem como analisar e confrontar os resultados de cada um deles. Para tal, foi construído um protótipo do veículo em estudo, com unidade de medida inercial (composta por acelerômetros e giroscópios), *driver* para controle dos motores, bateria para alimentação do veículo, motores DC, placa de desenvolvimento com microcontrolador ATMega 328p e toda a estrutura mecânica necessária. O método de identificação do modelo da planta em estudo foi o de distúrbio em malha fechada, haja vista a instabilidade deste tipo de sistema em malha aberta. Os resultados dos controladores *fuzzy* mostraram que um aumento no número de regras e faixas para cada variável da lógica refletiu em um melhor desempenho deste tipo controlador para a planta em estudo, no que se refere ao tempo de acomodação, *overshoot* e erro em regime. Quanto ao controlador *Neurofuzzy*, pôde-se comprovar uma eficácia no treinamento de sua rede, tanto com dados do PID, quanto com dados do controlador *Fuzzy*. Além disso, o controlador inteligente demonstrou excelentes resultados na estabilização do veículo.

Palavras-chave: Equilíbrio frontal; Veículo de duas rodas; PID; Controlador *Fuzzy*; Controlador *Neurofuzzy*

ABSTRACT

The purpose of this study is to implement three different controllers to maintain the front balance of a two-wheeled vehicle, as well as analyze and compare their results. With this aim, a prototype vehicle was built, with an inertial measurement unit (composed by accelerometers and gyroscopes), driver for motor control, battery for power supply of the vehicle, DC motors, development board with the ATMega 328p microcontroller and all mechanical structure. The method of identification of the model of the plant under study was the closed-loop disturbance, due to the instability of this type system in open-loop. Fuzzy controllers' results showed that an increase in the number of rules and ranges for each logical variable reflected a better closed-loop performance in terms of settling time, overshoot and tracking error. It was proved an efficiency in the training of the Neurofuzzy controller's network, with both PID's data and Fuzzy controller's ones. In addition, the intelligent controller has demonstrated excellent results in stabilizing the vehicle.

Key-Words: Front balance; Two-wheeled vehicle; PID; Fuzzy Controller; Neurofuzzy Controller

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A ideia de veículos elétricos é uma alternativa ao mercado automobilístico, sendo por vezes responsável por mudanças socioeconômicas. Veículos elétricos de duas rodas são um exemplo disso [1]. Estes demonstraram ter um impacto positivo na vida das pessoas, seja para a realização de deslocamentos cotidianos (um trajeto curto até o trabalho, dentro de supermercados e empresas em geral e mesmo para divertimento) ou o cumprimento de tarefas complexas, tais como mapeamentos de esgoto [2] e aplicações militares [3].

Embora o surgimento do primeiro veículo comercial com duas rodas tenha sido no ano 2000 [4], técnicas de controle para o seu equilíbrio frontal vêm sendo estudadas desde então, a fim de encontrar um melhor desempenho para o veículo nos quesitos tempo de acomodação, *overshoot* e erro permanente. Estes parâmetros tornam-se de fundamental importância para a imersão deste tipo de veículo no mercado, visto que os mesmos estão diretamente ligados à questão de segurança na mobilidade [5].

O objetivo deste trabalho é analisar técnicas alternativas de implementação para três diferentes tipos de controladores para o equilíbrio frontal de um veículo de duas rodas, analisando criticamente os seus resultados a fim de se encontrar o melhor controlador para o protótipo em estudo.

Visto que o veículo de duas rodas é um sistema instável em malha aberta, sua identificação será feita através do método de distúrbio em malha fechada, utilizando-se para tal de um único controlador proporcional junto ao sistema. Assim, poder-se-á chegar à função de transferência da planta através de sua relação com a função de distúrbio em malha fechada, a qual poderá ser comparada à de outras plantas similares já modeladas no estado da arte.

Inicialmente, será avaliado um controlador Proporcional Integral Derivativo (PID), visto que este é o controlador mais tradicional para controle de plantas em geral, além de já ter sido utilizado em trabalhos anteriores relacionados ao presente estudo ([6] e [7]). Os dados dos ensaios deste controlador serão armazenados e utilizados posteriormente para o treinamento da rede *neurofuzzy*.

Na sequência, será avaliada a utilização de controladores *fuzzy* com diferentes faixas para cada variável de entrada e saída da lógica, visto que este tipo de controlador também demonstrou eficácia no estado da arte ([8] e [9]) para o equilíbrio de veículos com duas rodas. O interesse na análise destes resultados será avaliar o impacto relacionado a diferentes estruturações dentro do controlador *fuzzy*, no que diz respeito ao número de regras e faixas.

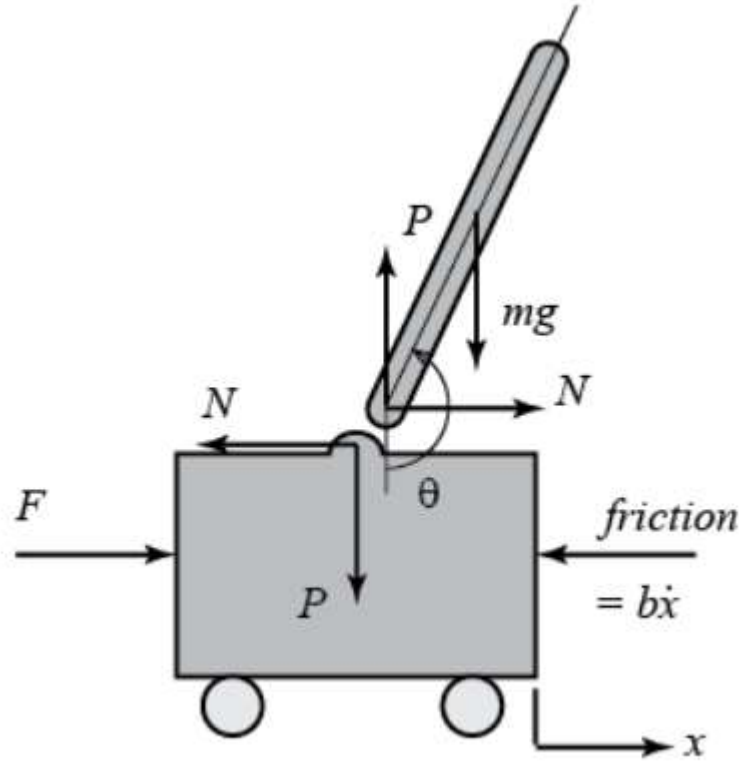
Logo após, os dados dos controladores PID modificado e *Fuzzy* serão utilizados no treinamento do terceiro controlador em estudo, o controlador híbrido *Neurofuzzy*. O intuito será avaliar os resultados tanto de treinamento (mudança de parâmetros), quanto de desempenho do controlador. Por fim, na última seção, os resultados de todos os controladores serão confrontados a fim de se obter uma análise final a respeito do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Modelo do veículo de duas rodas

O modelo matemático do veículo de duas rodas se faz de fundamental importância no projeto do controlador PID, visto que métodos de projeto como o LGR necessitam do modelo matemático do sistema em questão. O problema do equilíbrio frontal do veículo é muito similar ao clássico problema do pêndulo invertido. Com mais precisão, pode-se dizer que a modelagem do veículo de ruas rodas é exatamente a mesma de um pêndulo invertido acoplado a um carro motorizado [10], como mostrado na configuração da Figura 1.

Figura 1 - Configuração do pêndulo invertido acoplado a um carrinho



Fonte: [10]

Neste sistema, a entrada é dada pela força aplicada no carro e as saídas pela posição angular do pêndulo (θ) em relação ao eixo vertical e pela posição do carro (X) no eixo x. Somando-se as forças que atuam apenas na direção horizontal do carinho, tem-se:

$$M\ddot{x} + b\dot{x} + N = F \tag{1}$$

Sendo “M” a massa do carro, N a força de reação e “b” o coeficiente de fricção. Tomando-se agora o somatório de forças que atuam apenas na direção horizontal do pêndulo:

$$N = m\ddot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2 sen\theta \tag{2}$$

Substituindo-se a equação (2) da força de reação em (1):

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}cos\theta = F \tag{3}$$

Com relação ao eixo perpendicular ao pêndulo, tem-se o somatório de forças:

$$Psen\theta + Ncos\theta - mgsen\theta = ml\ddot{\theta} + m\ddot{x}cos\theta \tag{4}$$

Pode-se ainda somar os centroides em torno do pêndulo:

$$-Plsen\theta - Nlcos\theta = I\ddot{\theta} \tag{5}$$

Combinando-se (4) e (5) segue que:

$$(I + ml^2)\ddot{\theta} + mglsen\theta = -ml\ddot{x}cos\theta \tag{6}$$

Antes de continuar o desenvolvimento das equações (3) e (6), o qual levará a solução final para a função de transferência da planta, faz-se de fundamental importância analisar os termos não lineares presentes nas duas equações (3) e (6). Os métodos tradicionais de projeto de controladores PID assumem que o sistema é linear invariante no tempo, logo o modelo da planta deverá ser linearizado. Assumindo que o veículo oscilará dentro de uma faixa pequena, em relação à linha vertical de referência, ou seja, em $\theta = \pi$ segue que são válidas as aproximações:

$$\begin{aligned} cos\theta &\approx -1 \\ sen\theta &= -\theta \\ \dot{\theta}^2 &= 0 \end{aligned} \tag{7}$$

Substituindo-se os termos linearizados nas equações (3) e (6) tem-se:

$$(M+m)\ddot{x} + b\dot{x} - ml\ddot{\theta} = F \tag{8}$$

$$(I+ml^2)\ddot{\theta} - mgl\theta = ml\ddot{x} \tag{9}$$

Aplicando-se agora a Transformada de Laplace (com condições iniciais nulas) em (8) e (9):

$$(M+m)X(s)s^2 + bX(s)s - ml\theta(s)s^2 = F(s) \tag{10}$$

$$(I+ml^2)\theta(s)s^2 - mgl\theta(s) = mlX(s)s^2 \tag{11}$$

Visto que o intuito é encontrar-se a função de Transferência da saída θ para a entrada F, isola-se X(s) na equação (11):

$$X(s) = [\frac{I+ml^2}{ml} - \frac{g}{s^2}]\theta(s) \tag{12}$$

Por fim, substituindo-se (12) em (10) e rearranjando os termos, pode-se chegar a função de transferência para o veículo de duas rodas:

$$\frac{\theta(s)}{F(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \tag{13}$$

onde,

$$q = [(M+m)(I+ml^2) - (ml)^2]$$

Observa-se que o modelo encontrado apresenta três pólos e 1 zero, além de ser considerado um sistema SISO (uma entrada e uma saída). O mesmo também é função apenas de fatores construtivos (M,m,l,I) e físicos (g,b). Destaca-se ainda que a função de transferência que relaciona a posição do veículo no eixo x poderia ter sido obtida a partir destas equações, obtendo-se desta forma um sistema SIMO (uma entrada e múltiplas saídas). Entretanto, visto que o intuito deste trabalho visa controlador apenas a posição angular do veículo, esta última não foi desenvolvida.

2.2 Controlador Proporcional Integral Derivativo (PID)

Estima-se que mais de 90% de malhas de controle empregam o controlador PID ([11]), haja vista sua eficácia em controle de processos. O controlador PID se baseia em três ações básicas de controle: a proporcional, a integral e a derivativa. Os três termos de um controlador PID cumprem três requisitos comuns da maioria dos problemas de controle. A ação integral rende um erro de estado estacionário zero em um rastreamento de referência constante. Por outro lado, o termo proporcional responde imediatamente ao erro, mas normalmente não atinge a precisão desejada. Em algumas plantas, esse efeito pode levar a grandes erros transitórios quando o controle PI é usado. A ação derivativa combate esse problema [11]. Genericamente, o controlador PID pode ser expresso como:

$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}) \tag{14}$$

Analisando-se a expressão (14), percebe-se que existem 3 parâmetros a serem sintonizados neste controlador, cada um referente a uma ação: K (ganho proporcional), T_i (tempo de integração) e T_d (tempo de derivação).

O controlador proporcional, quando sozinho, é a estrutura mais simples de um PID. Entretanto, esta não atinge erro nulo em regime para plantas do tipo zero (sem polos na origem), podendo, desta forma, não estabilizar este tipo de sistema. Quando um sistema é do tipo 1 (1 polo na origem), pode-se, satisfatoriamente, controlar processos apenas com o controlador proporcional [11]. Com o aumento do ganho, os polos do sistema em malha fechada tendem a ser empurrados para o semiplano esquerdo, a fim de se obter a estabilidade do mesmo. Entretanto, percebe-se que o *overshoot* aumenta com o aumento de K, tornando o sistema instável após certo limite. Ressalta-se ainda que o erro em regime nunca será nulo apenas com a ação proporcional. Segue na expressão (15) a equação do controlador proporcional:

$$u(t) = K.e(t) \tag{15}$$

Quando o objetivo é eliminar ou reduzir o erro de seguimento, faz-se necessário a adição da ação integral. É válido se dizer, porém, que se faz necessária a ação proporcional conjunta à integral, para que ocorram melhoras no transitório e estabilidade para alguns sistemas. Na

expressão (16), é mostrada a equação do controlador PI. Já na expressão (17) é apresentada a equação do controlador após a aplicação da Transformada de Laplace:

$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau) \tag{16}$$

$$C_{pi}(s) = \frac{u(s)}{e(s)} = \frac{K(s+\frac{1}{Ti})}{s} \tag{17}$$

Pela expressão (17) fica claro que o controlador PI adiciona um polo na origem e um zero no inverso do tempo de integração.

A principal função da ação derivativa é estimar a evolução do sinal de erro Td segundos (tempo de derivação) à frente, aumentando a rapidez de reação do processo e fazendo com que oscilações iniciais sejam rejeitadas [12]. As equações no domínio do tempo e após a aplicação da Transformada de Laplace são apresentadas nas expressões (18) e (19), respectivamente:

$$u(t) = K(e(t) + T_d\frac{de(t)}{dt}) \tag{18}$$

$$C_{pd}(s) = \frac{u(s)}{e(s)} = \frac{K(1+pT\ )(s+\frac{p}{1+pTd})}{(s+p)} \tag{19}$$

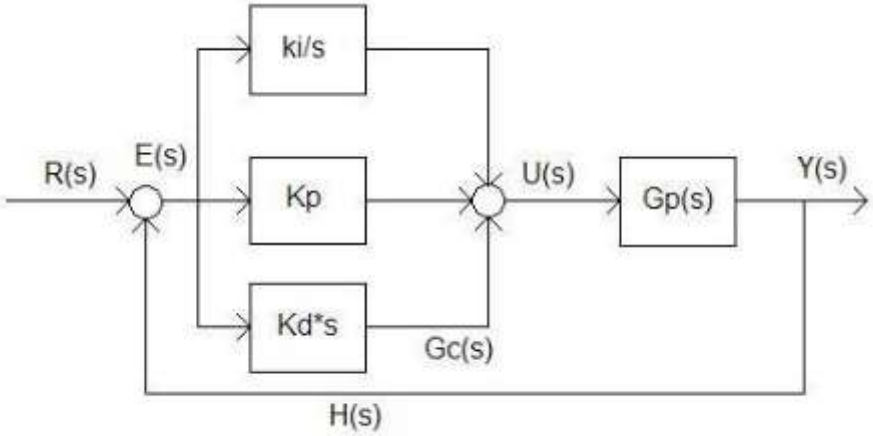
Analisando-se a equação (19), percebe-se que o controlador proporcional derivativo adiciona ao sistema um zero e um polo. Este último normalmente é alocado em altas frequências, e possui duas importantes funções: limitar o ganho em altas frequências e não interferir na dinâmica do sistema.

A equação completa do controlador PID, a qual junta os efeitos do controlador PI e PD, é dada pela expressão (20):

$$C_{pid}(s) = \frac{u(s)}{e(s)} = \frac{K(1+pTd)\left(s^2+\frac{1+pTi}{Ti(1+pTd)}s+\frac{p}{Ti(1+pTd)}\right)}{s(s+p)} \tag{20}$$

Ao todo, o controlador completo adiciona dois zeros e dois polos ao sistema em malha fechada. Segue, na Figura 2, o diagrama de blocos do controlador PID:

Figura 2 - Diagrama de blocos do controlador PID



Fonte: [6]

2.3 Controladores *Fuzzy*

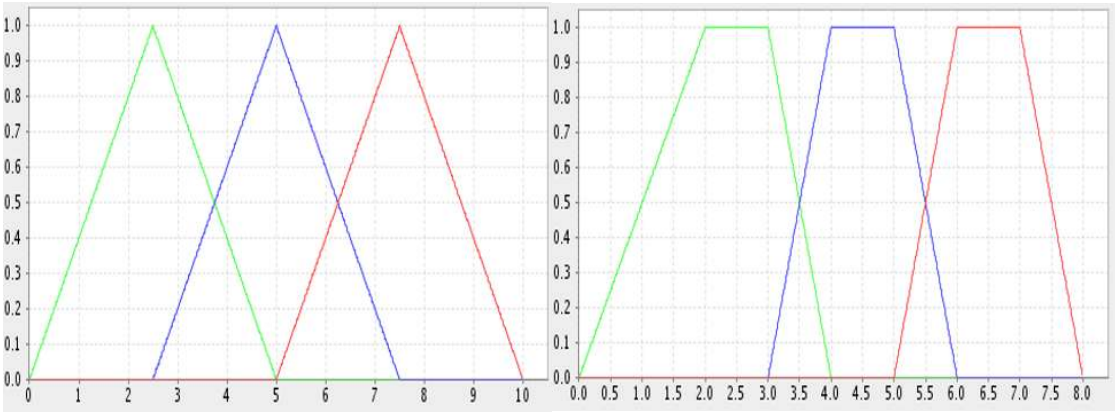
A tradução da palavra inglesa *fuzzy* para a língua portuguesa ajuda, de certa forma, a explicitar a diferença entre este tipo de lógica e a binária. “Nebuloso” reforça a ideia de que existem valores intermediários entre os tradicionais (e absolutos) valores “0” e “1”. Não se tem apenas branco e preto. Entre estes existe uma gama infinita de tons que podem ser interpretados de diferentes formas segundo linguísticas qualitativas/subjetivas e até mesmo quantitativas. E aqui se tem outra importante variável que aproxima a lógica *fuzzy* do ser humano: a linguística. Pessoas possuem diferentes percepções a respeito de quase tudo. Dificilmente existem verdades (ou visões) absolutas a respeito de alguma situação, de forma que uma das proposições da lógica *fuzzy* é justamente realizar divisões das variáveis de entrada e saída em faixas. No caso prático do veículo de duas rodas, por exemplo, pode-se dividir a variável de entrada “erro” (definido aqui como a diferença entre a posição angular de referência e a leitura do ângulo do sensor) em diversas faixas, de modo que esta não fique restrita apenas às classificações “erro grande” e “erro pequeno”. Dessa forma, ter-se-iam, por exemplo, as subdivisões do tipo: muito grande, grande, medio, pequeno e muito pequeno.

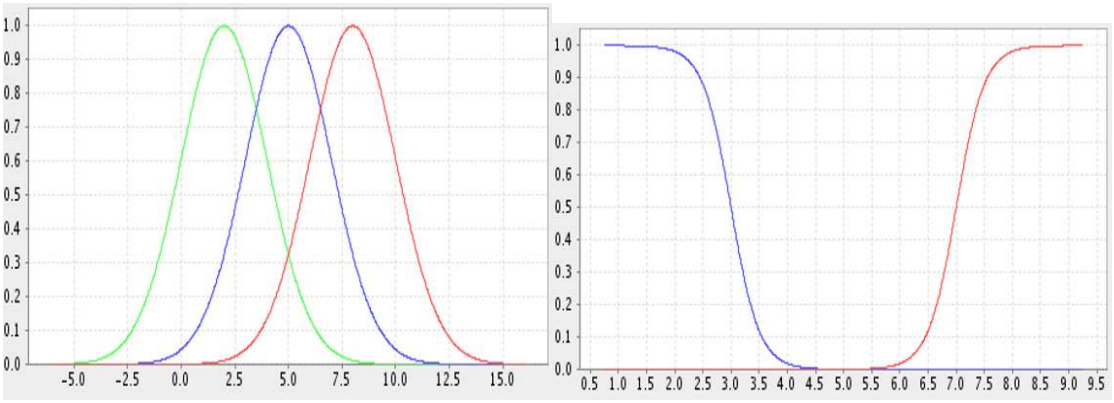
A lógica *fuzzy* é uma técnica utilizada para resolução de problemas genéricos [13], composta por uma série de processos que envolvem manipulações de valores de variáveis. Através de processos conhecidos como *fuzzyficação* e *defuzzyficação*, entradas são manipuladas de forma a gerar saídas soluções para o problema, tudo isso baseado em um sistema de regras pré-definido. Outras importantes características que tornam a lógica menos ou mais robusta,

para determinado tipo de problema, são o formato das funções de pertinência (das entradas e saídas) e a definição das regras que regirão a lógica *fuzzy*. A técnica se mostra útil na resolução de problemas de sistemas de controle, tais como o do pêndulo invertido ([8] e [9]), visto que muitas vezes a modelagem deste tipo de sistema é complexa e com várias não linearidades.

Seguindo este raciocínio, faz-se necessário, após a divisão das variáveis em faixas, a distribuição das mesmas dentro de uma certa gama de valores, aplicáveis ao problema em específico. Continuando com o exemplo do parágrafo anterior, poder-se-ia definir o intervalo 7° a 12° para a faixa "erro mediano" e o intervalo 22° a 33° para a faixa "erro muito grande". Nota-se que esta definição é puramente baseada em uma visão intuitiva. Para projetos que exijam um alto grau de precisão/confiabilidade, faz-se necessário a subdivisão destes intervalos através da opinião de especialistas na área ou mesmo por métodos computacionais ([13]) (redes neurais e algoritmos genéticos). Feita esta etapa, deve-se agora definir um certo grau de pertinência que deve ser dado a cada valor encontrado em cada faixa. Nada mais lógico do que se fazer isso através de funções, denominadas "funções de pertinência" [13], e que tem por objetivo atribuir um valor entre 0 e 1 (0% e 100%) para todo e qualquer valor que entrar na lógica *fuzzy*. Este é o chamado processo de *fuzzyficação*, o qual depende do formato escolhido para as funções de pertinência. Por isso, depende-se, novamente, da ação de um especialista ou de algum algoritmo inteligente para que se possa escolher a melhor função para representação das variáveis. A Figura 3 ilustra quatro modelos distintos e tradicionais para funções de pertinência. É importante salientar-se que quão maior for o número de cruzamentos entre as faixas, maior será a fidelidade de representação do problema em questão.

Figura 3 – (Da esquerda para a direita e de cima para baixo) Funções de pertinência nos modelos triangular, trapezoidal, sino e *sigmoidal*





Fonte: [13]

Após a definição das faixas para cada variável, dos intervalos que compõe cada uma destas e das funções de pertinência que as representam, faz-se necessária a criação de um critério que possa escolher quais valores ou saídas serão atribuídos a quais entradas. A este critério dá-se o nome de "regras". Estas, mais uma vez baseadas em especialistas e/ou técnicas computacionais, fazem a conexão entre as variáveis de entrada e saída. O exemplo da Tabela 1 ilustra bem essa ideia para um controlador *fuzzy*, utilizado para o equilíbrio de um veículo de duas rodas, com 2 entradas (erro e variação de erro) e 1 saída (sinal de controle), sendo 2 faixas para cada variável de entrada e 3 faixas para a variável de saída da lógica. Nota-se que o operador utilizado na construção das regras foi o operador "e", resultando sempre em uma multiplicação dos valores anteriormente *fuzzyficados*. É válido de se dizer também que o aumento do número de regras está diretamente ligado à eficácia da lógica, bem como ao custo computacional exigido pelo algoritmo [13].

Tabela 5 - Exemplo da criação de regras para o problema do equilíbrio frontal

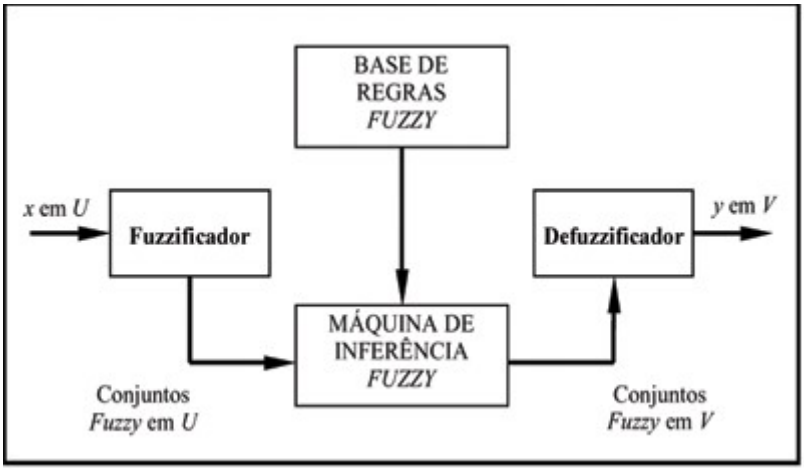
Se erro pequeno e variação de erro pequena, então sinal de controle é pequeno
Se erro pequeno e variação de erro grande, então sinal de controle é médio
Se erro grande e variação de erro pequena, então sinal de controle é médio
Se erro grande e variação de erro grande, então sinal de controle é grande

Realizadas a *fuzzyficação* e a construção das regras, finalmente é chegada a hora de se gerar a saída final, a qual será a possível solução do problema em questão. A esta ação destina-se o processo de *defuzzyficação*. Neste, os valores dos graus de pertinência resultantes de cada regra são "mapeados" para as funções de pertinência associadas às variáveis de saída. Desta forma, ter-se-á um valor real para cada regra. Como última ação, este processo deve calcular, através de algum método, o valor final real da saída da lógica. Os métodos mais tradicionais encontrados são o da média dos valores e o método do centroide, sendo este último mais preciso em relação ao primeiro [13]. Salienta-se ainda que existe um outro método para se chegar a

valores solução para o problema, o qual utiliza os Polinômios de Sugeno (explicado na próxima seção) para encontrar a melhor saída para o problema. Este método ganhou importância devido à sua exclusiva utilização pelo modelo ANFIS – *Adaptive-Network-Based Fuzzy Inference System* [14].

A Figura 4 mostra o diagrama de blocos para o sistema do veículo de duas rodas sobre a atuação de um controlador *Fuzzy* com uma entrada (erro) e 1 saída (sinal de controle).

Figura 4 - Diagrama de blocos para o controlador Fuzzy



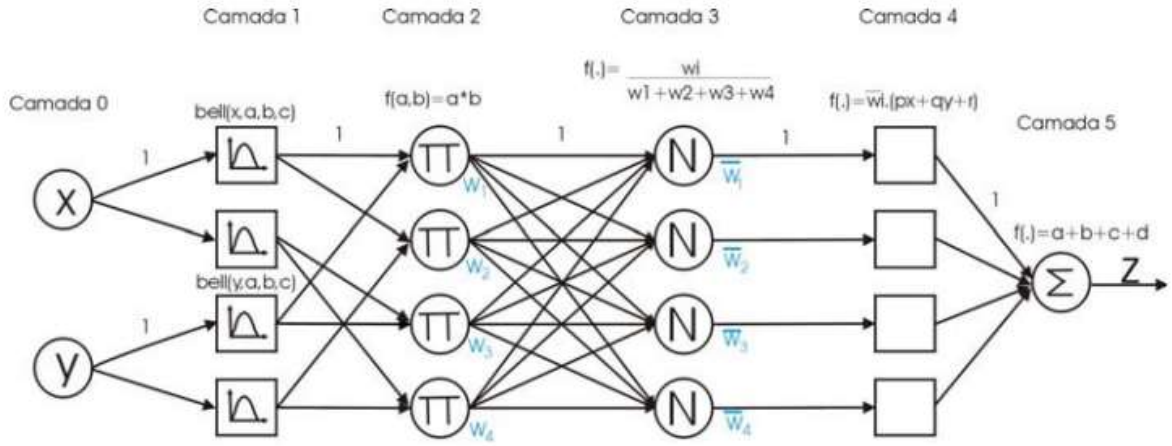
Fonte: [13]

2.4 Controladores *Neurofuzzy*

Os controladores *Neurofuzzy* se utilizam das vantagens de duas técnicas inteligentes: a lógica *fuzzy* e as redes neurais artificiais (RNAs). De um modo geral, um sistema *fuzzy* é implementado sobre uma rede neural (adaptada a lógica). Desta forma, o controlador híbrido herda as funções de pertinência associadas à cada variável da lógica e a construção de regras desta, podendo, através de algum algoritmo de aprendizado, treinar a rede para ajustar de forma inteligente os parâmetros de cada faixa.

Um dos principais modelos de controladores *neurofuzzy* aplicado a sistemas de controle é o ANFIS [14]. Sua representação clássica poder ser visualizada na Figura 5:

Figura 5 - Representação clássica de um modelo ANFIS



Fonte: [15]

Analisando-se a Figura 5, percebe-se que este controlador *neurofuzzy* é composto por 6 camadas (a camada 0 representa neste caso duas entradas hipotéticas do sistema, cada uma representada por um neurônio). A camada 1 é composta pelas faixas componentes das funções de pertinência associadas à cada entrada. Ou seja, ter-se-á um neurônio para cada faixa que representa uma variável. No exemplo da Figura 5, cada entrada foi dividida em duas faixas, sendo a função *bell* a de pertinência (com parâmetros de ajuste “x”, “a”, “b” e “c”). Percebe-se também que cada entrada se liga somente às faixas que a representam.

As saídas da camada 1 nada mais são do que os valores *fuzzyficados* das entradas (graus de pertinência atribuídos a cada variável de acordo com a faixa de representação), as quais servirão de entrada para a camada 2. Esta última é composta pelas regras de construção da lógica *fuzzy*. Ou seja, tantos serão os neurônios desta camada quanto forem o número de regras (no exemplo da Figura 5 têm-se um total de 4 regras). Em consequência, o resultado de cada regra representará a saída de cada neurônio, a qual será entrada para a camada 3.

Na camada 3 ocorre o processo de normalização dos pesos (definidos aqui como saídas das regras da lógica *fuzzy* e representados como “W”). Nela, cada peso é dividido pelo somatório de todos os pesos, resultando em um peso normalizado ($\overline{W}$). Esta camada possui o mesmo número de neurônios da camada anterior.

Na camada 4 ocorre a associação de cada peso normalizado com um polinômio de Sugeno. Este último pode ser visto como uma função das entradas, $f(x, y)$, tendo como parâmetros livres "p", "q" e "r". Segue na expressão (20) a saída de cada neurônio componente desta camada:

$$Saída\ do\ neurônio\ \text{"i"} = \overline{W_i}(p_i x + q_i y + r_i) \quad (21)$$

Por último, na camada 5, ocorre o somatório das saídas da camada anterior, resultando no sinal de controle final. Entretanto, para se obter um sinal de controle satisfatório para este modelo de controlador *neurofuzzy*, faz-se necessário um adequado treinamento da rede projetada. E, para tal, dois aspectos são de fundamental importância: o banco de dados, contendo os exemplos da rede, e o algoritmo de treinamento, o qual fará de fato o ajuste dos parâmetros livres.

O banco de dados deve conter informações relevantes do sistema a ser controlado ([14]). Normalmente o que se quer é uma tabela que relacione valores de entrada(s) com o de saída(s), a cada instante amostrado do processo. Assim a rede será treinada sobre um certo tipo de "supervisão", visto que lhe será dito qual(is) saída(s) deve(m) ser aplicada(s) a qual(is) entrada(s). Essas informações podem ser coletadas tanto de banco de dados públicos quanto de ensaios experimentais. A Tabela 2 ilustra um banco de dados fictício para o sistema do pêndulo invertido de duas entradas e uma saída, com três amostras.

Tabela 6 - Banco de dados fictício para um sistema de duas entradas e uma saída

Instante (ms)	Erro (°) - Entrada 1	Variação do erro (°/s) - Entrada 2	Saída (PWM)
2,50	15,43	10,54	76%
22,50	2,34	1,34	5%
42,50	9,87	4,56	15%

O algoritmo mais comum a ser utilizado com o modelo ANFIS é o *backpropagation*, haja vista sua simplicidade e eficácia em trabalhos anteriores ([15]). A principal vantagem na utilização deste tipo de algoritmo está na retroprogação do erro para cada neurônio que compõe a rede, bem como no ajuste dos parâmetros livres.

Quando um exemplo (um conjunto de entrada(s) com sua(s) respectiva(s) saída(s)) é mostrado à rede, cada neurônio irá calcular sua respectiva saída de acordo com o critério que lhe foi imposto (*fuzzyficação*, regras, *defuzzyficação*, normalização, soma,...). Ao final, tendo-se a saída da rede (saída da última camada) e a saída exata (fornecida pelo banco), pode-se calcular o erro cometido pelo controlador. Normalmente o erro de adaptação calculado é o "erro quadrático" (22), dado simplesmente pelo quadrado da diferença da saída da rede pela saída exata, em cada ponto "l" (cada exemplo):

$$E_p^l = \sum_{i=1}^{Nout} (d_s^l - X_{K,i})^2 \tag{22}$$

É comum definir-se a derivada do erro quadrático (23), visto que, na maioria das vezes, a variação deste torna-se mais relevante do que o erro quadrático em si [14], além de diminuir impacto no custo computacional quando comparado ao erro quadrático.

$$\delta_{k,i} = \frac{dE_p}{dX_{k,i}} = \sum_{m=1}^{N(k+1)} \frac{dE_p}{dX_{k+1,m}} \frac{df_{k+1,m}}{dX_{k,i}} = \sum_{m=1}^{N(k+1)} \delta_{k+1,m} \frac{df_{k+1,m}}{dX_{k,i}} \tag{23}$$

Percebe-se, analisando a expressão (23), que o cálculo de sinal de erro da camada k depende do resultado do sinal de erro da camada k+1. Assim fica comprovada a retropropagação proposta pelo algoritmo *backpropagation* [16].

Por fim, para ajuste dos parâmetros livres da rede, utiliza-se a derivada do sinal de erro, calculada anteriormente, para se chegar à variação do parâmetro, através da fórmula (24):

$$\Delta_{\alpha_p} = -n \frac{dE_p}{dX_{k,i}} \frac{df_{k,i}}{d\alpha} = -n.\delta_{k,i} \frac{df_{k,i}}{d\alpha} \tag{24}$$

Sendo, $\frac{df_{k,i}}{d\alpha}$ a variação da função de ativação do neurônio (polinômio de Sugeno ou função de pertinência) em relação ao parâmetro que se deseja ajustar. A letra N é definido como coeficiente de aprendizado ([16]) e é configurado de maneira empírica, com o intuito de acelerar o processo de treinamento da rede. Ao final de cada iteração (após cada exemplo apresentado à rede), basta somar a variação calculada do parâmetro ao parâmetro da iteração anterior. Ao final de todas as iterações (número aqui definido pelo construtor da rede), ter-se-á o resultado do treinamento.

3 Metodologia

Nesta seção são apresentados os procedimentos utilizados para a obtenção de um modelo localmente linear da planta e o projeto dos controladores PID modificado, *Fuzzy* e *Neurofuzzy*. Além disso é detalhada a construção do veículo a ser utilizado no experimento, bem como de todos os periféricos utilizados na implementação dos métodos de controle desenvolvidos.

3.1 Construção da estrutura do veículo

A construção do veículo de duas rodas envolve duas partes principais: uma elétrica e outra mecânica. A primeira engloba o microcontrolador para processamento dos dados, a ponte H para controle dos motores, a bateria para alimentação do circuito e o sensor para aquisição de dados. Já a segunda está ligada ao posicionamento e fixação dos motores e rodas, bem como da estrutura de apoio para os componentes eletrônicos (haste pendular e plataforma de apoio).

Foram utilizados dois motores DC de 6 V (com corrente máxima de 100 mA), os quais foram acoplados a rodas de borracha com diâmetro externo de 6 cm. A haste utilizada para representação do pêndulo invertido foi projetada para ter um comprimento de 7 cm, sendo feita de alumínio, a qual foi presa aos motores através de porcas e parafusos. No topo da estrutura foi acoplada à haste uma plataforma de madeira com 9 cm^2, a fim de que a mesma se adequasse às dimensões da placa de desenvolvimento. A plataforma também foi fixada na haste a partir de porcas e parafusos.

A escolha do microcontrolador foi de suma importância para a realização do estudo em questão, haja vista que este deveria possuir memórias e processamento adequados aos tipos de controladores e rotinas a serem implementados. Assim, após um estudo técnico e análise de custo benefício, escolheu-se o microcontrolador da MICROCHIP ATMega 328P. Este possui uma frequência de 16 MHz e memórias RAM e ROM de 2 KB e 32 KB, respectivamente. Outra vantagem na escolha do microcontrolador se deu pelo mesmo já vir embutido em uma placa de desenvolvimento Arduino Uno com periféricos que facilitaram as conexões de forma geral. Salienta-se também que, a fim de proteger o microcontrolador contra quedas e outras intempéries, foi utilizada uma caixa protetora externa à placa. Na Figura 6 é mostrada a placa de desenvolvimento que contempla o microcontrolador.

Para leitura dos dados de entrada do sistema, foi necessário se fazer um estudo a respeito de alguma unidade de medida inercial que possuísse no mínimo 3 acelerômetros e 3

giroscópios. Através da aceleração (entregue pelo acelerômetro) e da variação de posição em torno de um eixo (fornecida pelo giroscópio), é possível de se fazer uma fusão das informações para chegar ao ângulo de inclinação do veículo em relação ao eixo de referência (variável de entrada desejada). Após uma pesquisa técnica e de custo benefício, optou-se pela utilização da unidade de medida inercial MPU 6050, visto que a mesma possui 3 acelerômetros e 3 giroscópios. Sua comunicação I2C também é relativamente simples e a resolução de 10 bits permite uma precisão adequada para o trabalho. A unidade de medida inercial considerada também pode ser visualizada na Figura 6.

O sistema escolhido para controle dos motores foi o *driver* L298N, composto por duas pontes H, as quais permitem, através de quatro pinos digitais, controlar o sentido de rotação dos motores. O *driver* também possui 2 pinos que permitem a realização do controle de velocidade dos motores por meio de uma modulação PWM. A alimentação da placa é de 6-36V e a corrente máxima fornecida é de 2A. A placa pode ser visualizada na Figura 6.

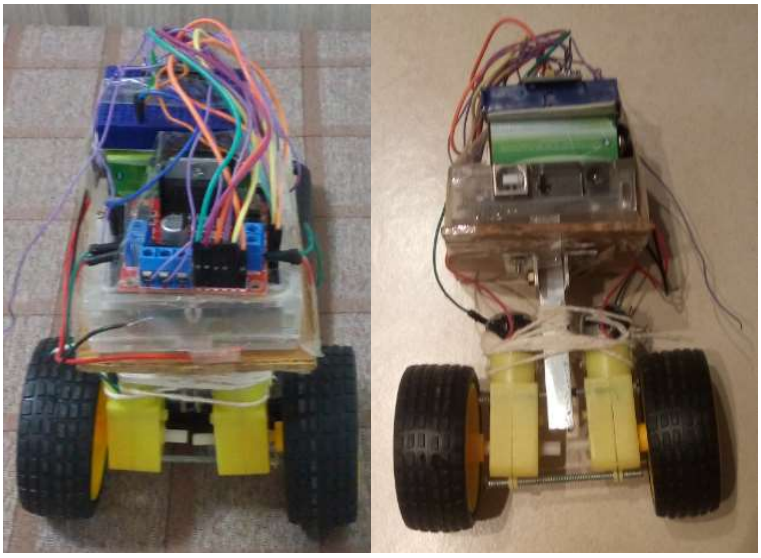
Para a alimentação do circuito foi utiliza uma bateria de 9 V e corrente máxima de 250 mA. A escolha se deu pela adequação da mesma ao sistema (atende os requisitos de tensão e corrente, além de ser leve (15g)) e também por esta permitir que o veículo não ficasse preso a uma fonte de alimentação.

Visto que o microcontrolador não possui memória volátil, fez-se necessária a incorporação de algum componente que permitisse armazenar os dados para que os mesmos fossem utilizados como exemplos de treinamento para a rede *Neurofuzzy*. Escolheu-se então o módulo cartão SD, o qual permite o armazenamento de dados em um cartão SD. A comunicação do módulo com o microcontrolador se dá pelo protocolo SPI, fato que não interfere na troca de dados entre o ATMega 328P e o MPU6050.

Figura 6 - (Da esquerda para a direita) Unidade de medida inercial MPU6050, Placa de desenvolvimento com o microcontrolador ATMega 328P e *Driver* controlador L298N

A Figura 7 mostra o protótipo final do veículo (vistas frontal e superior), com as partes mecânica e eletrônica acopladas.

Figura 7 - Protótipo final do veículo de duas rodas (vistas superior e frontal)



3.2 Ensaios para identificação do modelo da planta

O projeto do controlador PID (via LGR) para o veículo de duas rodas exige a obtenção do modelo da planta em questão por alguns motivos específicos. Primeiro, o modelo matemático serve para dar uma ideia da estrutura da função de transferência, mas os valores que fidedignamente representarão a planta serão obtidos pelos parâmetros extraídos dos ensaios. Segundo, o modelamento permitirá a realização de simulações prévias à implementação do controlador, sendo esta uma etapa importante no projeto de sistemas de controle. Terceiro, a planta em estudo foge aos padrões típicos assumidos na concepção dos métodos tradicionais de ajuste de controladores PID baseados em características elementares do comportamento do processo [11], visto que a mesma deve atuar em uma faixa restrita para ser considerada linear. Ainda, existe uma preocupação maior com os parâmetros de desempenho (erro em regime, *overshoot* e tempo de acomodação), visto que o correto funcionamento do veículo depende fortemente do bom desempenho destes parâmetros. Por fim, poder-se-á comparar o resultado do controlador PID com os demais controladores projetados.

Dentre os diferentes métodos existentes para identificação do modelo da planta, optou-se pelo levantamento dos parâmetros da função de transferência em malha fechada. A escolha

se deu pela dificuldade em se fazer experimentos com plantas deste tipo em malha aberta, haja vista sua instabilidade. Também não seria apropriado utilizar-se unicamente de plantas similares que já foram modeladas matematicamente em trabalhos passados, visto que na prática todo sistema envolve suas particularidades.

Inicialmente definiu-se empiricamente um ganho proporcional ao erro, o qual fosse capaz de reagir a perturbações na saída do sistema (empurrões no veículo) sem deixar o carrinho cair. Logo após, posicionou-se o veículo na referência desejada, sendo esta definida com o veículo paralelo ao eixo perpendicular ao chão (eixo vertical). A partir daí, distúrbios foram aplicados ao veículo, fazendo com que o mesmo oscilasse acima e abaixo da linha imaginária de referência (dentro de uma certa faixa). O diagrama de blocos da Figura 9 ilustra o sistema descrito, juntamente a perturbação aplicada à saída. Os dados foram coletados de forma a se obter uma tabela relacionando a entrada (sinal de controle) com a saída (ângulo de inclinação do veículo), a cada instante amostrado pelo microcontrolador. A Figura 8 mostra os dados coletados em forma de gráfico, onde foi aplicada uma mudança de variáveis para que o valor de zero graus nos dados correspondesse ao valor de pi graus do modelo teórico.

Figura 8 - Dados extraídos do ensaio com a planta em malha fechada

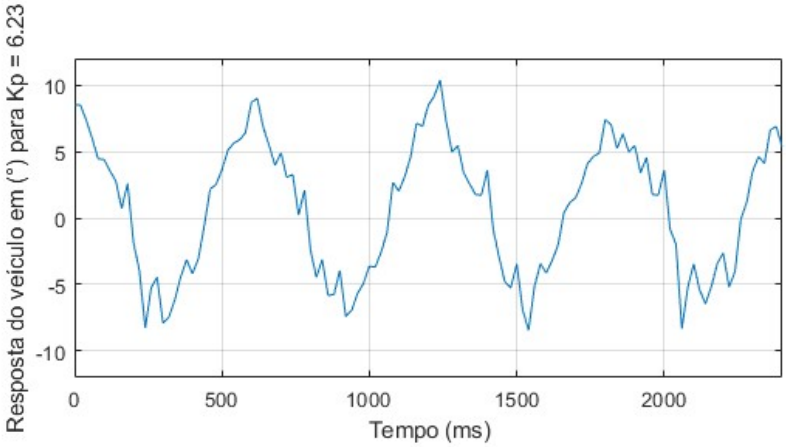
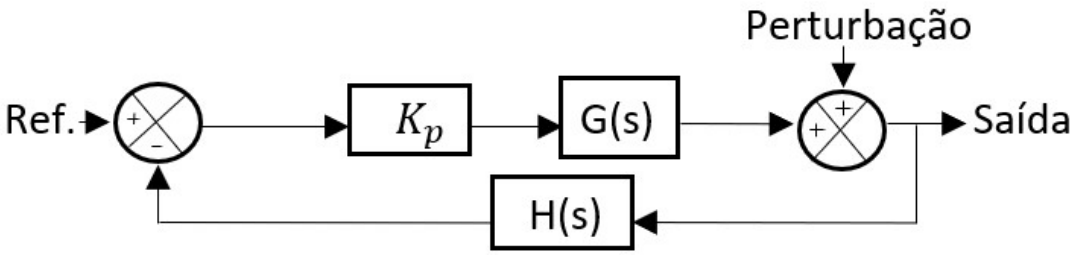


Figura 9 - Diagrama de blocos do sistema com a perturbação aplicada à saída



Seguindo-se, fez-se uso do *System Identification Toolbox*, do MATLAB, a fim de que se pudesse carregar os dados do ensaio e, posteriormente, extrair-se a função de distúrbio em

malha fechada (T_d). Optou-se pela estimação de um modelo de função de transferência, visto que o sistema controla apenas um tipo de variável (ângulo de inclinação). Assim não se fez necessária a escolha por um modelo de espaço de estados, o qual seria adequado para trabalhos relacionados ao controle de posição em mais de um eixo. Um modelo de terceira ordem foi escolhido com base no modelo matemático da planta apresentado na equação (13). Assim pôde-se chegar a função de distúrbio desejada, a qual segue na equação (25). Através dela foi possível se obter a função de transferência da planta através do desenvolvimento da equação (26):

$$T_d(s) = \frac{1.762\ 10^{-5}\ s^3 + 6.048\ 10^{-6}\ s^2 - 0.0003554\ s - 7.898\ 10^{-5}}{1.762\ 10^{-5}\ s^3 + 6.04810^{-6}\ s^2 + 0.0001467\ s - 7.898\ 10^{-5}} \tag{25}$$

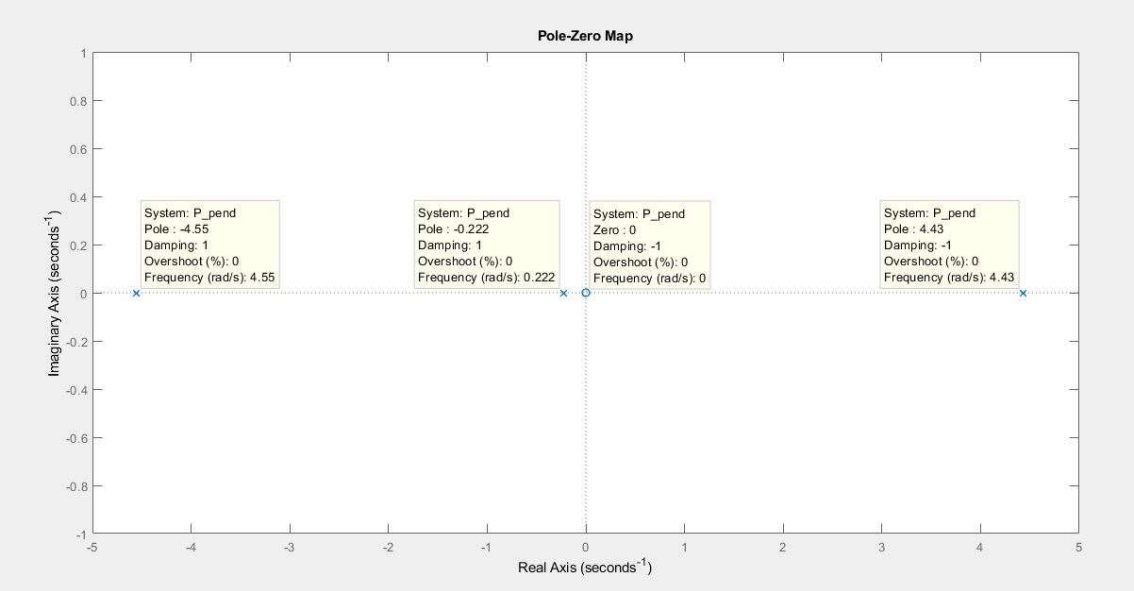
$$1 + K_p.\,G(s) = \frac{1}{T_d(s)} \tag{26}$$

$$G(s) = \frac{\left(1 - T_d(s)\right)}{\left(K_p * T_d(s)\right)} =$$

$$G(s) = \frac{8.059\ 10^{-5}\ s}{1.762\ 10^{-5}s^3 + 6.048\ 10^{-6}s^2 - 0.0003554\ s - 7.898\ 10^{-5}} \tag{27}$$

Analisando-se a função de transferência obtida experimentalmente (27) e o modelo matemático teórico (13), constata-se que o modelo auferido está coerente com a planta em estudo. Isso pode ser comprovado pela ordem do sistema (ordem 3) e pela localização de um dos polos no semi-plano direito (instabilidade do sistema). A Figura 10 mostra a localização dos polos e zeros da planta.

Figura 10 - Localização dos polos e zeros da planta em estudo



3.3 Projeto do controlador PID modificado

Após a identificação da função de transferência do veículo de duas rodas, iniciou-se o projeto do controlador PID. O método utilizado para desenvolvimento do controlador foi o de alocação de polos e zeros via Root *Locus*, utilizando-se da ferramenta *rltool* (do MATLAB) para tal.

Partindo-se de referências ([11]), pode-se dizer que o controle de posição para plantas de segunda ou terceira ordem com motores exige que o controlador clássico possua tanto uma ação integrativa quanto uma derivativa, além da parcela proporcional. Isso se deve à necessidade de compensação para a rápida dinâmica do sistema e para a rejeição de perturbação de carga (já esperada pela instabilidade do sistema em malha aberta). Ainda, visto que a função de transferência da planta apresenta um zero em zero, será necessário o acréscimo de mais um integrador (polo na origem) e um zero. A adição deste polo e zero faz com que a estrutura do controlador deixe de ser a apresentada na seção 2.2, sendo denominado de controlador PID modificado.

Definida a estrutura do controlador, faz-se necessário decidir a localização de cada um dos polos e zeros, além do ganho proporcional. A parte derivativa adiciona ao sistema um polo e um zero, já a parte integral dupla adicionará 2 polos na origem e 2 zeros em posições a serem definidas. O outro polo da parte derivativa deve ser alocado em altas frequências, conforme descrito na seção 2.2. Assim sobram os três zeros para serem alocados, juntamente com a definição do ganho proporcional.

Decidiu-se pela escolha da posição dos zeros e análise do ganho proporcional como parâmetro variante do *Root Locus*. As alocações foram feitas de modo a se obter os melhores resultados de forma geral, com um particular cuidado para o *overshoot*, haja vista que para veículos de duas rodas este parâmetro é considerado de extrema importância no conforto dos passageiros [8].

A melhor localização encontrada para os zeros foi de -0.8, -0.9 e -0.7. Para o polo em altas frequências, a melhor posição ficou em -62 (não houve alterações na dinâmica do sistema para posições superiores). A Figura 12 mostra o *Root Locus*, a equação do controlador obtida e

a resposta do sistema para um distúrbio de saída, correspondente ao ângulo de zero grau. Já a Figura 13 traz a ampliação do *Root Locus* na origem, para melhor visualização.

Figura 11 - *Root Locus*, equação do controlador e resposta ao salto para o sistema

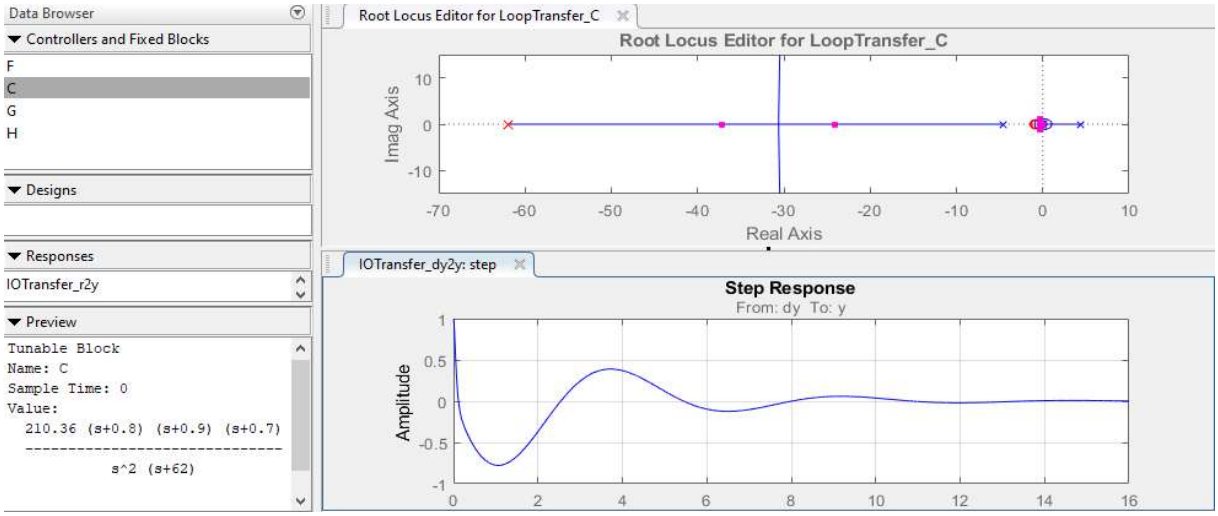
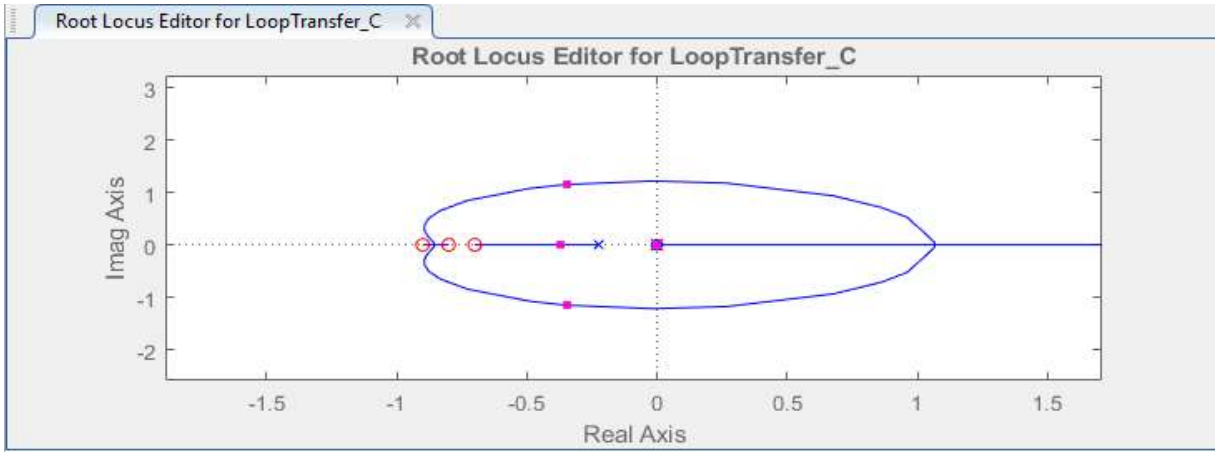


Figura 12 - Ampliação do *Root Locus* do sistema na origem



3.4 Projeto dos controladores *Fuzzy*

O projeto do controlador *fuzzy* para o veículo de duas rodas em questão envolve três particularidades que devem ser analisadas com certa minúcia, a fim de que se possam obter resultados satisfatórios. São eles: a definição do tipo e quantidade de funções de pertinência para cada variável de entrada e saída; a elaboração das regras da lógica e consequente inferência para cada situação; a análise do custo computacional para a implementação do controlador.

O objetivo das funções de pertinência associadas às entradas (erro do ângulo de inclinação do veículo em relação ao eixo vertical e variação deste) e saída (sinal de controle em

PWM) é estabelecer, com o máximo grau de fidelidade, o quanto cada variável representa, em termos quantitativos, de cada faixa. Sendo assim, optou-se pela implementação de dois controladores *fuzzy*, o primeiro com três faixas para cada variável e o segundo com cinco faixas, com o intuito de se investigar o quanto se melhora a o desempenho do controlador com o aumento do número de faixas.

A escolha do tipo de função de pertinência para representação das variáveis está diretamente ligada ao custo computacional. Quanto mais complexa for a função que descreve cada uma das regiões (maior número de operações), maior será o número de instruções e processamento demandados. Esta situação está diretamente ligada ao problema de limitação de hardware. Assim, a função escolhida para as variáveis, tanto de entrada quanto de saída, foi a triangular, visto que esta possui uma implementação computacional relativamente simples quando comparada a outras funções típicas como gaussiana e *sigmoidal*.

Nas Figuras 14, 15, 16, 17, 18 e 19 são apresentadas as funções de pertinência para as duas variáveis de entrada do sistema e para a saída deste, para cada controlador, com suas respectivas faixas (definidas empiricamente antes dos ensaios experimentais).

Figura 13 - Funções de Pertinência associadas à entrada "erro" para o para o controlador *Fuzzy* com 3 faixas (aqui o eixo das abcissas representa a faixa completa para a variável erro)

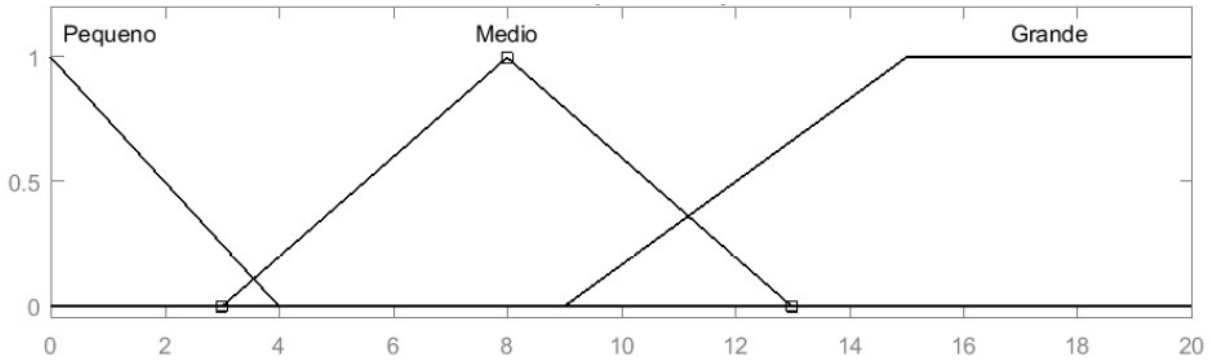


Figura 14 - Funções de Pertinência associadas à entrada "variação do erro" para o para o controlador *Fuzzy* com 3 faixas (aqui o eixo das abcissas representa a faixa completa para a variável variação do erro)

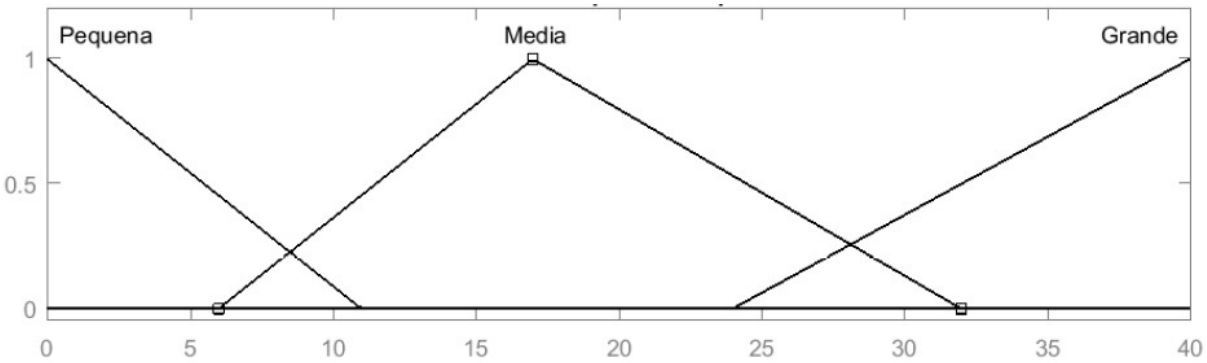


Figura 15 - Funções de Pertinência associadas à saída “sinal de controle” para o controlador *Fuzzy* com 3 faixas (aqui o eixo das abcissas representa a faixa completa para a variável sinal de controle)

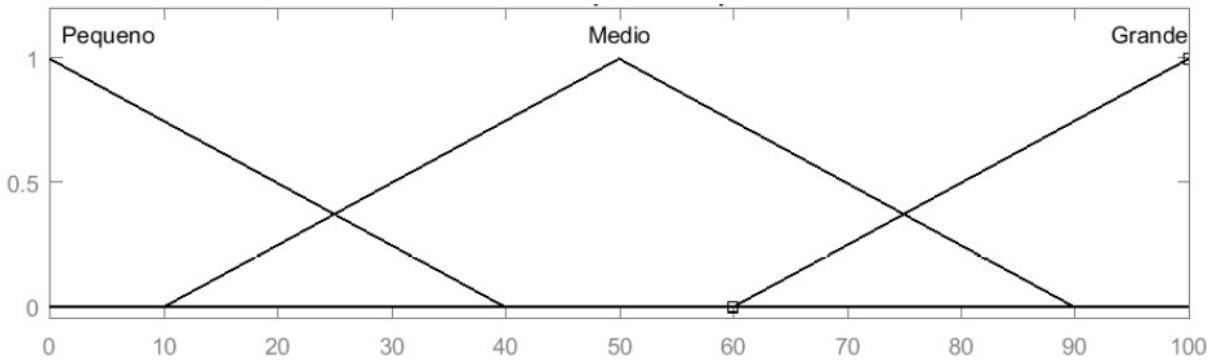


Figura 16 - Funções de Pertinência associadas à entrada "erro" para o para o controlador *Fuzzy* com 5 faixas (aqui o eixo das abcissas representa a faixa completa para a variável erro)

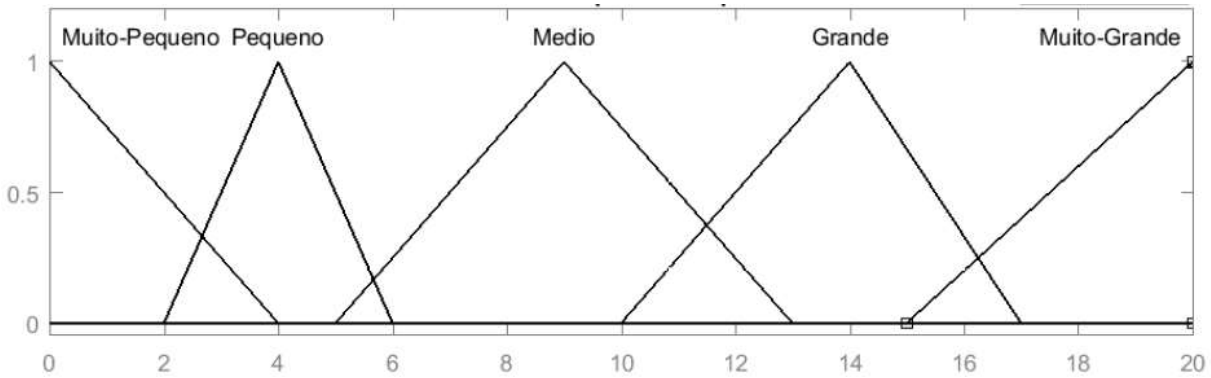


Figura 17 - Funções de Pertinência associadas à entrada "variação do erro" para o para o controlador *Fuzzy* com 5 faixas (aqui o eixo das abcissas representa a faixa completa para a variável variação do erro)

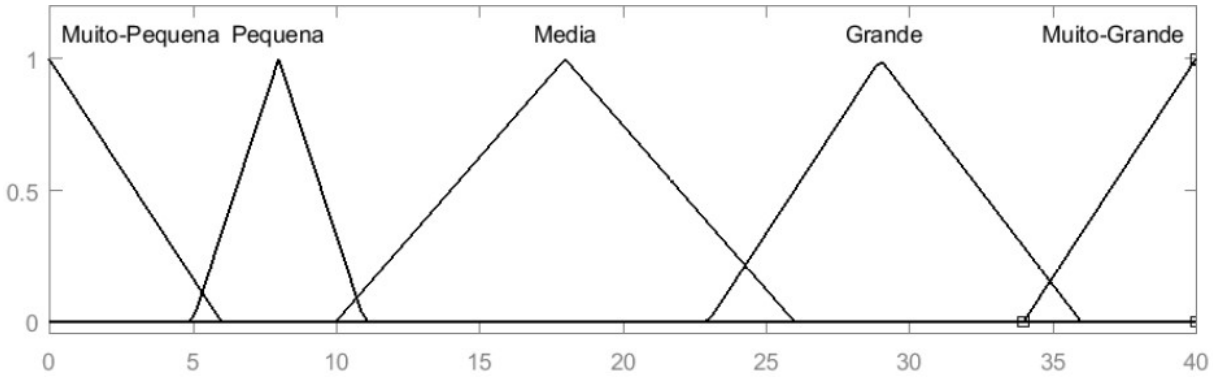
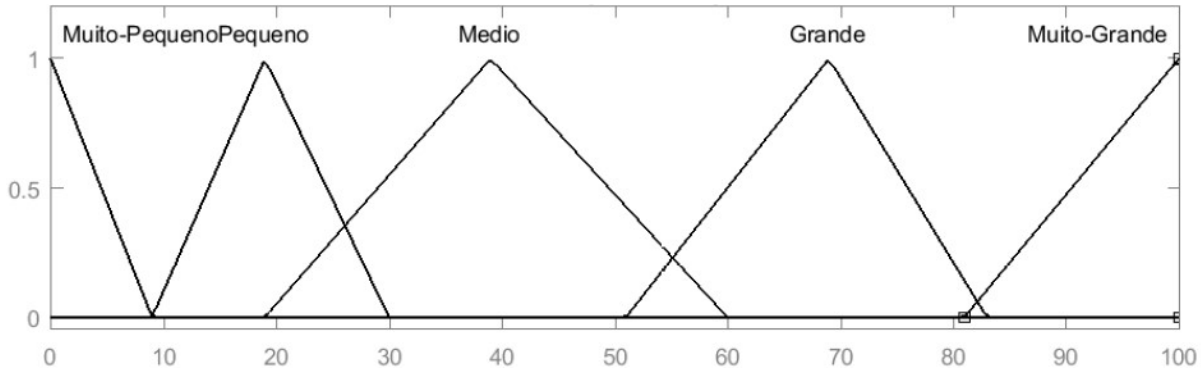


Figura 18 - Funções de Pertinência associadas à saída “sinal de controle” para o para o controlador *Fuzzy* com 5 faixas (aqui o eixo das abcissas representa a faixa completa para a variável sinal de controle)



Com relação à determinação das regras, é de extrema importância que as mesmas sejam definidas segundo alguma lógica que esteja coerente com as diferentes situações (ligadas ao problema do equilíbrio frontal do veículo) que possam acontecer com o decorrer do tempo. Assim, faz-se necessário um estudo qualitativo de todas as possíveis combinações entre as variáveis de entrada e saída.

Considere inicialmente a situação em que o veículo está na posição de referência ($\Theta = 0º$) e pende para um dos lados (Figura 20, caso 1). Aqui percebe-se que, tanto o erro quanto a variação do erro são positivos e, mais do que isto, estão constantemente aumentando sua magnitude. Desta forma, parece lógico se utilizar de um sinal de controle proporcionalmente alto a fim de se compensar a queda do veículo. Seguindo-se, na Figura 21 é analisado o caso para consequente recuperação do veículo (em direção à sua posição de referência, caso 2). Nota-se que, nesta situação, o erro ainda é grande e positivo (Θ lido longe da referência), mas a

variação deste é negativa (ou seja, o veículo está retornando para sua posição de referência). Assim, faz-se necessária uma ação de controle proporcionalmente menor do que o caso anterior. As duas últimas situações críticas estão na combinação de um erro pequeno com variações de erro grande (caso 3) e pequena (caso 4). Para o caso 3, representado pela Figura 22, é interessante a atuação de um sinal de controle moderado, para que não haja uma grande extrapolação em relação a linha imaginária de referência. Já para o caso 4 (Figura 23), percebe-se que o veículo está muito próximo da situação desejada, por isso o sinal de controle deve ser proporcionalmente pequeno.

Figura 19 - Veículo em queda livre (caso 1)

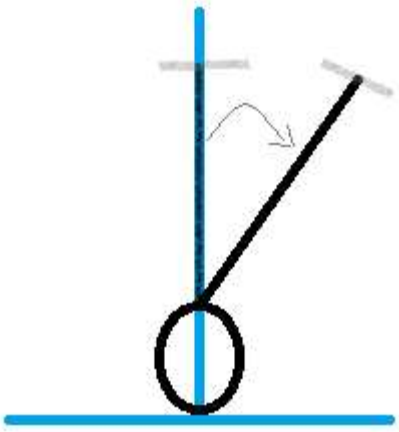


Figura 20 - Veículo com erro grande mas se aproximando da posição de referência (caso 2)

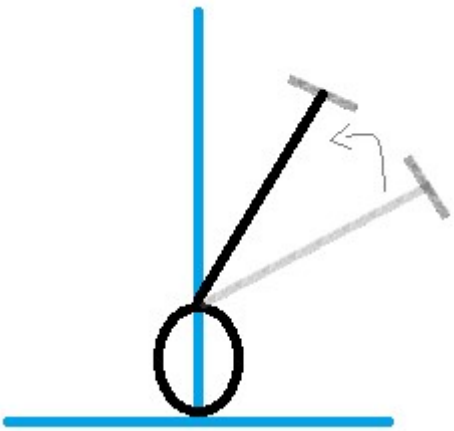


Figura 21 – Veículo com variação
aumentando mas erro pequeno (caso 3)

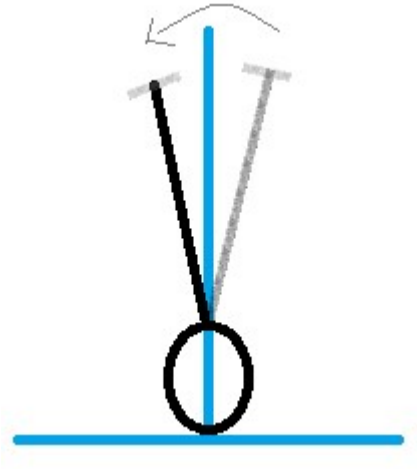
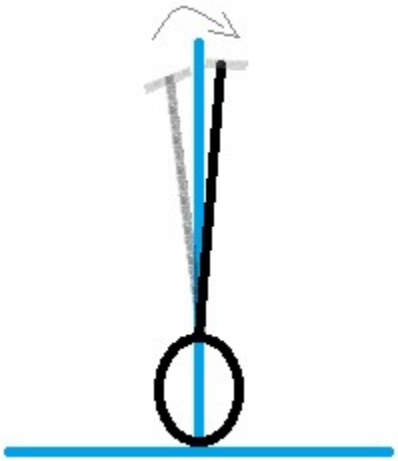


Figura 22 - Veículo com variação
diminuindo e erro pequeno (caso 4)



Feito o estudo das principais situações críticas para a planta, é possível que, de forma empírica, se definam as regras para a lógica *fuzzy* de cada controlador. Aqui, novamente surge a necessidade de se conciliar o número de regras com as limitações do *hardware* disponível. Assim, avaliando-se a capacidade de processamento e de memória RAM e ROM do micro, decidiu-se construir 18 regras para o controlador *fuzzy* com três faixas (todas as possíveis combinações entre as variáveis de entrada foram feitas para este controlador) e 46 regras para o controlador *fuzzy* com 5 faixas. Salienta-se que a construção das regras, bem como a divisão das faixas para cada função de pertinência, foi feita através do *Fuzzy Logic ToolBox*, do MATLAB. As figuras 24 e 25 mostram a construção das regras para cada controlador.

Figura 23 - Construção das regras para o controlador *Fuzzy* com 3 faixas

1. If (Erro is Pequeno) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Pequeno) (1)
2. If (Erro is Pequeno) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Pequeno) (1)
3. If (Erro is Pequeno) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Pequeno) (1)
4. If (Erro is Medio) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Medio) (1)
5. If (Erro is Medio) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Medio) (1)
6. If (Erro is Medio) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Pequeno) (1)
7. If (Erro is Grande) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Grande) (1)
8. If (Erro is Grande) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Grande) (1)
9. If (Erro is Grande) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Medio) (1)

1. If (Erro is Pequeno) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Pequeno) (1)
2. If (Erro is Pequeno) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Pequeno) (1)
3. If (Erro is Pequeno) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Grande) (1)
4. If (Erro is Medio) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Pequeno) (1)
5. If (Erro is Medio) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Medio) (1)
6. If (Erro is Medio) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Grande) (1)
7. If (Erro is Grande) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Grande) (1)
8. If (Erro is Grande) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Grande) (1)
9. If (Erro is Grande) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Grande) (1)

Figura 24 - Construção das regras para o controlador *Fuzzy* com 5 faixas

1. If (Erro is Muito-Pequeno) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Pequeno) (1)
2. If (Erro is Muito-Pequeno) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Pequeno) (1)
3. If (Erro is Muito-Pequeno) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Muito-Pequeno) (1)
4. If (Erro is Muito-Pequeno) and (Variacao-Erro(-) is Muito-Grande) then (Sinal-Controle is Muito-Pequeno) (1)
5. If (Erro is Pequeno) and (Variacao-Erro(-) is Muito-Pequena) then (Sinal-Controle is Pequeno) (1)
6. If (Erro is Pequeno) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Pequeno) (1)
7. If (Erro is Pequeno) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Pequeno) (1)
8. If (Erro is Pequeno) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Pequeno) (1)
9. If (Erro is Pequeno) and (Variacao-Erro(-) is Muito-Grande) then (Sinal-Controle is Pequeno) (1)
10. If (Erro is Medio) and (Variacao-Erro(-) is Muito-Pequena) then (Sinal-Controle is Medio) (1)
11. If (Erro is Medio) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Medio) (1)
12. If (Erro is Medio) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Medio) (1)
13. If (Erro is Medio) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Pequeno) (1)
14. If (Erro is Medio) and (Variacao-Erro(-) is Muito-Grande) then (Sinal-Controle is Pequeno) (1)
15. If (Erro is Grande) and (Variacao-Erro(-) is Muito-Pequena) then (Sinal-Controle is Grande) (1)
16. If (Erro is Grande) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Grande) (1)
17. If (Erro is Grande) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Grande) (1)
18. If (Erro is Grande) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Medio) (1)
19. If (Erro is Grande) and (Variacao-Erro(-) is Muito-Grande) then (Sinal-Controle is Medio) (1)
20. If (Erro is Muito-Grande) and (Variacao-Erro(-) is Muito-Pequena) then (Sinal-Controle is Muito-Grande) (1)
21. If (Erro is Muito-Grande) and (Variacao-Erro(-) is Pequena) then (Sinal-Controle is Muito-Grande) (1)
22. If (Erro is Muito-Grande) and (Variacao-Erro(-) is Media) then (Sinal-Controle is Grande) (1)
23. If (Erro is Muito-Grande) and (Variacao-Erro(-) is Grande) then (Sinal-Controle is Medio) (1)

1. If (Erro is Muito-Pequeno) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Pequeno) (1)
2. If (Erro is Muito-Pequeno) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Pequeno) (1)
3. If (Erro is Muito-Pequeno) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Medio) (1)
4. If (Erro is Muito-Pequeno) and (Variacao-Erro(+) is Muito-Grande) then (Sinal-Controle is Grande) (1)
5. If (Erro is Pequeno) and (Variacao-Erro(+) is Muito-Pequena) then (Sinal-Controle is Pequeno) (1)
6. If (Erro is Pequeno) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Pequeno) (1)
7. If (Erro is Pequeno) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Pequeno) (1)
8. If (Erro is Pequeno) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Medio) (1)
9. If (Erro is Pequeno) and (Variacao-Erro(+) is Muito-Grande) then (Sinal-Controle is Grande) (1)
10. If (Erro is Medio) and (Variacao-Erro(+) is Muito-Pequena) then (Sinal-Controle is Medio) (1)
11. If (Erro is Medio) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Medio) (1)
12. If (Erro is Medio) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Medio) (1)
13. If (Erro is Medio) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Grande) (1)
14. If (Erro is Medio) and (Variacao-Erro(+) is Muito-Grande) then (Sinal-Controle is Grande) (1)
15. If (Erro is Grande) and (Variacao-Erro(+) is Muito-Pequena) then (Sinal-Controle is Grande) (1)
16. If (Erro is Grande) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Grande) (1)
17. If (Erro is Grande) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Grande) (1)
18. If (Erro is Grande) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Muito-Grande) (1)
19. If (Erro is Grande) and (Variacao-Erro(+) is Muito-Grande) then (Sinal-Controle is Muito-Grande) (1)
20. If (Erro is Muito-Grande) and (Variacao-Erro(+) is Muito-Pequena) then (Sinal-Controle is Grande) (1)
21. If (Erro is Muito-Grande) and (Variacao-Erro(+) is Pequena) then (Sinal-Controle is Grande) (1)
22. If (Erro is Muito-Grande) and (Variacao-Erro(+) is Media) then (Sinal-Controle is Muito-Grande) (1)
23. If (Erro is Muito-Grande) and (Variacao-Erro(+) is Grande) then (Sinal-Controle is Muito-Grande) (1)

Para validação da coerência dos modelos dos dois controladores, foram simuladas duas situações críticas (casos 1 e 2). As Figuras 26, 27, 28 e 29 trazem as simulações para cada controlador e situação:

Figura 25 - Simulação do caso 2 para o controlador *Fuzzy* com 3 faixas

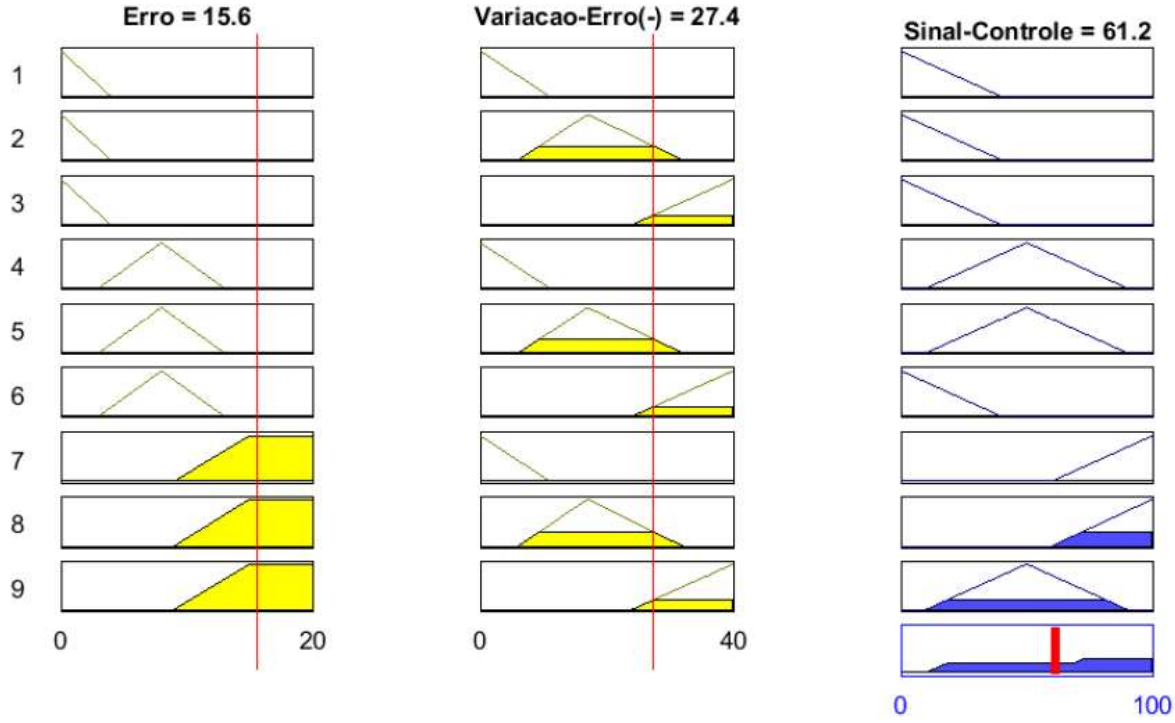


Figura 26 - Simulação do caso 1 para o controlador *Fuzzy* com 3 faixas

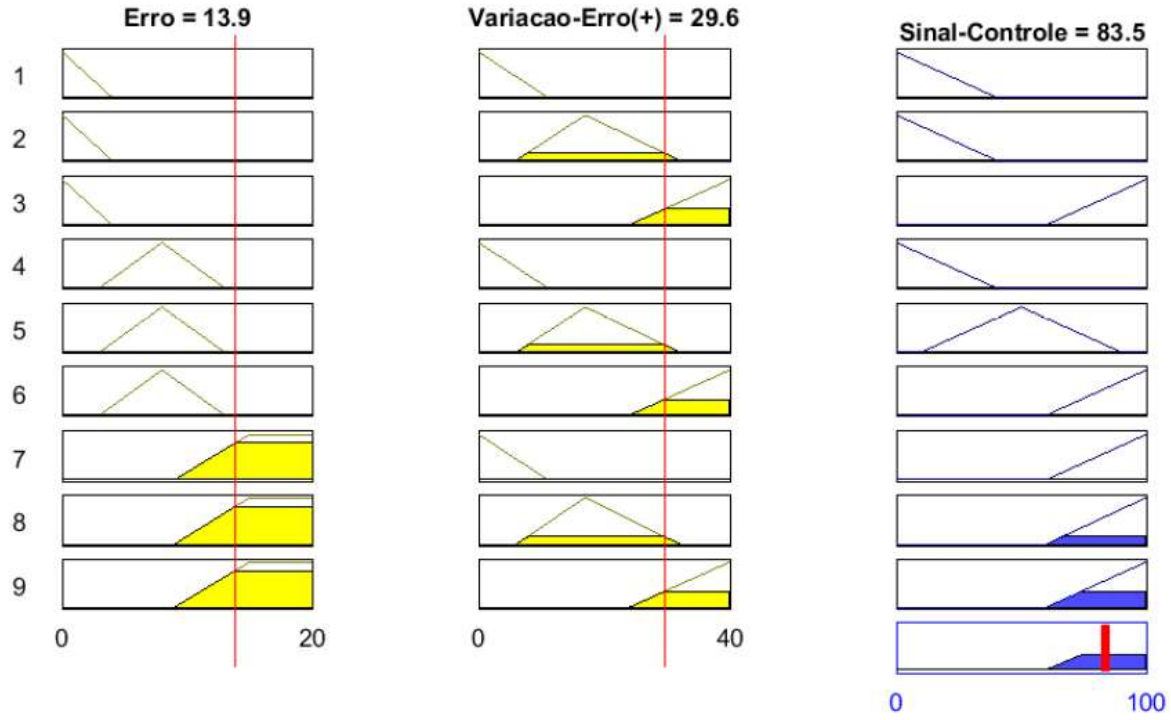


Figura 27 - Simulação do caso 2 para o controlador *Fuzzy* com 5 faixas

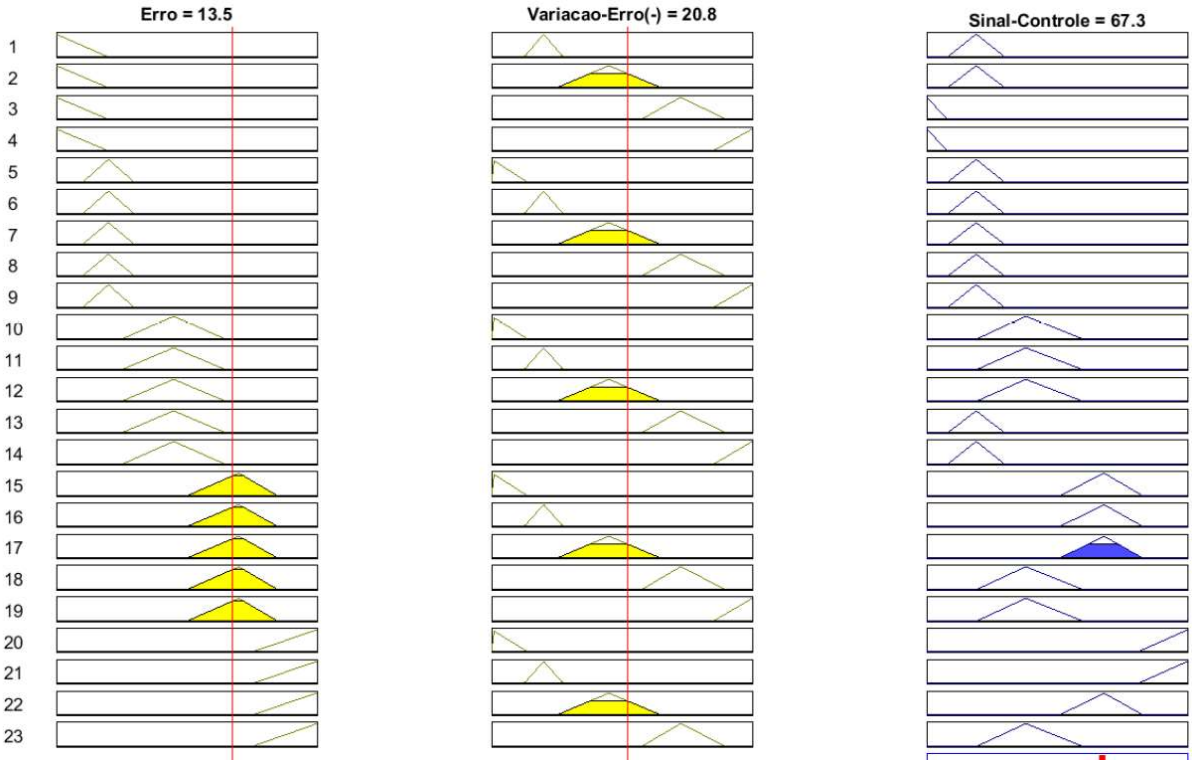
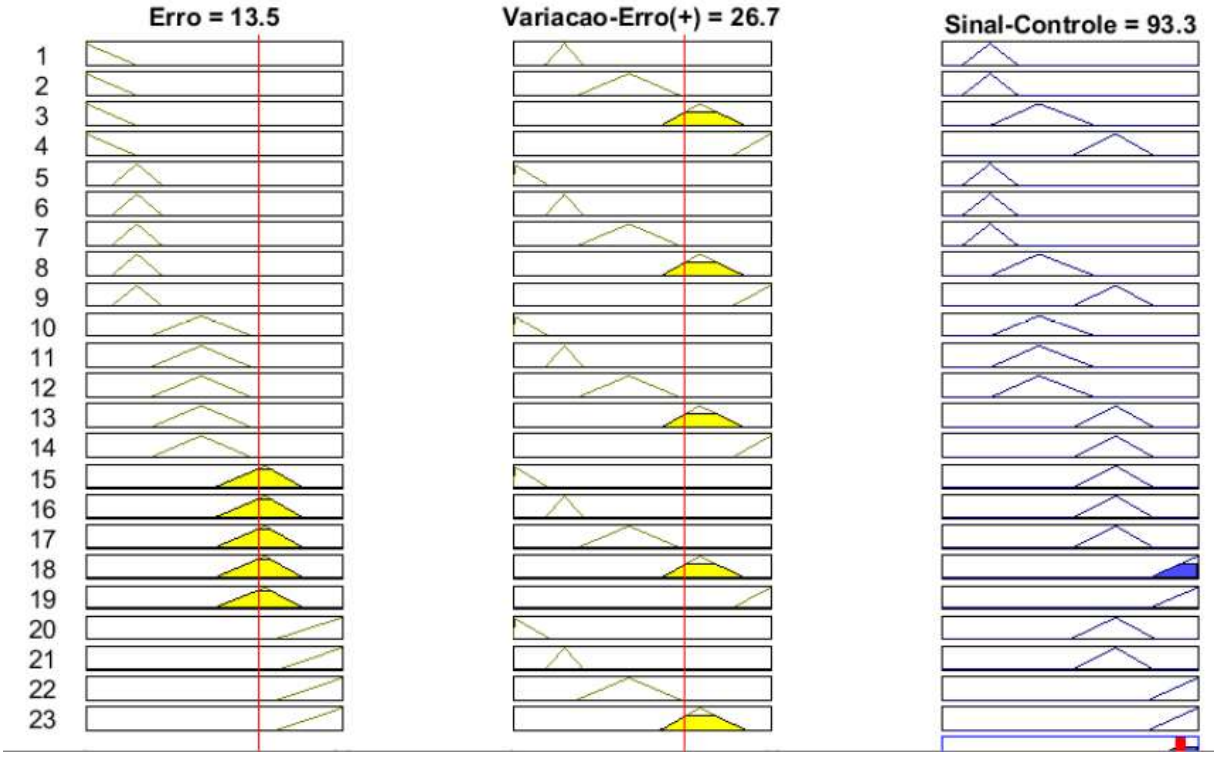


Figura 28 - Simulação do caso 1 para o controlador *Fuzzy* com 5 faixas



Na Figura 26, percebe-se que um erro de 15,6° é considerado grande para a lógica implementada (com 3 faixas). Já uma variação de erro negativa de 27,4° é tratada como uma variação que está transitando da faixa média para a faixa grande (tendo portanto duas "identidades"). Assim, sendo que o operador escolhido para a inferência foi o "*and*", apenas duas das regras tiveram saída não nula, as quais foram utilizadas para o processo de *defuzzyficação*. Seguindo-se na mesma situação, porém para o controlador *Fuzzy* com 5 faixas (Figura 28), percebe-se que o erro de 13,5° foi *fuzzyficado* como grande e a variação de erro negativa de 20,8° foi identificada apenas como uma variação média. Obteve-se, portanto, uma única saída diferente de zero.

Na Figura 27 (caso 1), percebe-se que, assim como no caso da Figura 20, o erro é tratado como grande e a variação deste, agora positiva, como média e grande. Entretanto, visto que neste caso o veículo está em queda livre, obteve-se um sinal de controle maior (83,5) em relação ao do caso 2 (61,2). Finalmente, na Figura 29 (caso 1 para o controlador *Fuzzy* com 5 faixas), o mesmo erro foi tratado como grande e a variação deste (26,7°) apenas como grande.

Por fim, implementou-se em linguagem C os controladores projetados no microcontrolador 328P da ATMega. O código resultante pode ser visualizado no anexo.

3.5 Projeto do controlador *Neurofuzzy*

Para o projeto do controlador inteligente *Neurofuzzy*, fez-se necessário avaliar dois aspectos relevantes à sua implementação: o tipo de treinamento a ser feito juntamente com a fonte de dados a ser utilizada para se treinar a rede e as características da rede implementada.

A realização do treinamento não supervisionado para o sistema foi descartada por dois motivos principais. O primeiro diz respeito a limitações de *hardware*. Para que o veículo atingisse um nível desejado de aprendizado, seriam necessárias inúmeras quedas, fato que poderia acarretar em danos ao veículo. Ainda, seriam necessários rodas maiores e motores com maior potência, a fim de que o veículo conseguisse se reerguer quando alcançasse o chão. Segundo, o entendimento a respeito de algoritmos complexos de inteligência computacional e realidade virtual seriam exigidos para criação do banco de dados via simulação, o que foge ao escopo do trabalho. Decidiu-se, portanto, pela utilização de um sistema de aprendizado supervisionado.

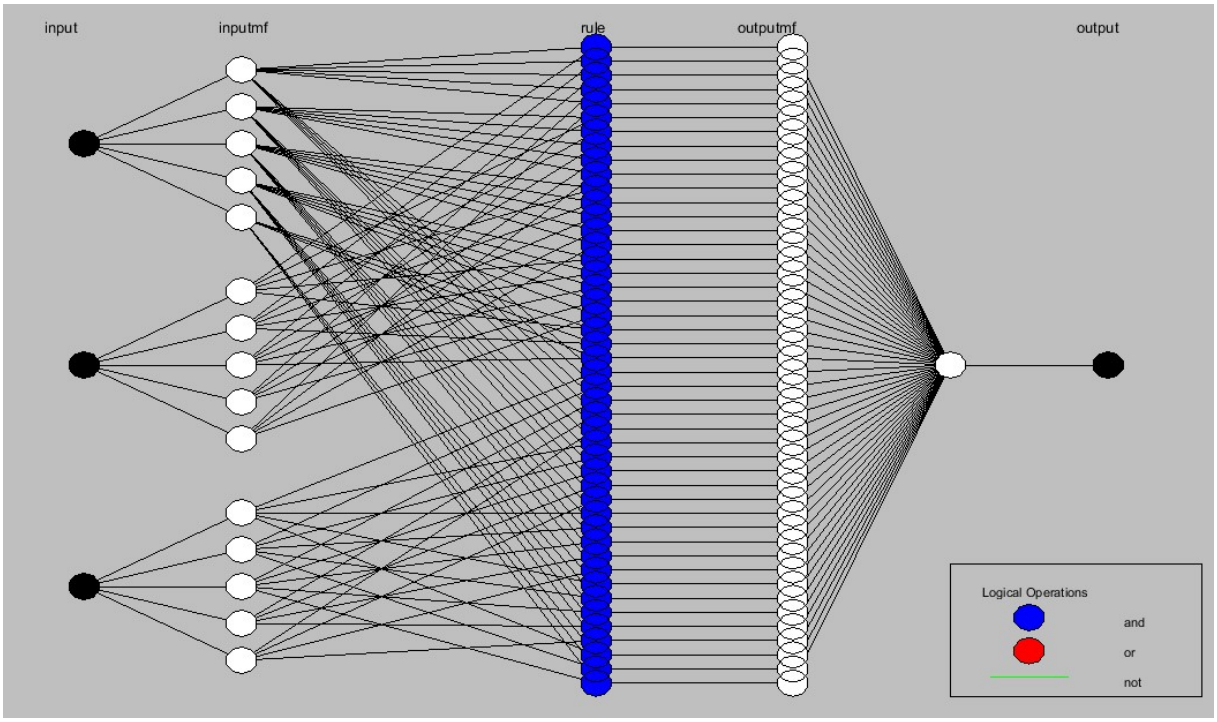
Normalmente, em sistemas de aprendizado supervisionados, banco de dados já existentes são utilizados para a extração de exemplos para a rede. Este, porém, não é o caso do veículo de duas rodas em estudo, visto que o mesmo foi construído sem a utilização de uma plataforma comercial. Assim, foi necessária a criação de um banco de dados a partir de um outro tipo de controlador, técnica esta já utilizada no estado da arte [15]. Partindo-se do princípio que um dos objetivos da implementação do controlador *Neurofuzzy* é a análise das mudanças ocorridas em relação ao controlador *fuzzy* do ponto de vista de adaptação das funções de pertinência, optou-se por utilizar tanto os dados do controlador PID quanto os do *Fuzzy* (com 5 faixas) para o treinamento da rede. Assim, através da captação dos dados de ensaios obtidos com estes controladores, gerou-se um banco de dados para o treinamento da rede. O banco pode ser entendido como uma tabela que contém os dados de entrada (erro e variação do erro) e saída (sinal de controle) do sistema a cada instante amostrado.

A implementação da rede (Figura 30), baseada no modelo ANFIS, pode ser vista como um sistema híbrido, o qual carrega características do sistema *fuzzy* de cinco faixas e de uma rede neural com 5 camadas. Os neurônios da camada 1 representam as funções triangulares das variáveis de entrada do sistema (erro e sua variação), sendo composta, portanto, por 2 neurônios. Os dados *fuzzyficados* dirigem-se então para a camada 2, composta pelas regras da lógica *fuzzy* (aqui cada neurônio representa 1 regra, totalizando, ao final, 46 neurônios). Na camada 3 ocorre a normalização das saídas da camada anterior, dividindo-se cada saída (resultado da regra “n”) pelo somatório das saídas. Esta camada possui o mesmo número de

neurônios da camada anterior. Na camada 4, cada entrada normalizada é multiplicada pelo correspondente polinômio de Sugeno. Por fim, na camada 5, composta pelo neurônio único que representa o sinal de controle de saída do sistema, ocorre o somatório de todas as saídas da camada 4. Ressalta-se que neste modelo não é necessário o processo de *defuzzyficação*.

Implementou-se em linguagem C, inicialmente, o código do controlador *Neurofuzzy* com o algoritmo *Backpropagation* a ser embarcado, conforme pode ser visualizado no Anexo 2. Entretanto, como já era de se esperar, as memórias RAM e ROM do microcontrolador não suportaram o tamanho da implementação. Deste modo, todo o treinamento e implementação *offline* da rede foi realizada pelo *Fuzzy Logic ToolBox*, do MATLAB.

Figura 29 - Modelo do controlador *Neurofuzzy* proposto



4 Resultados e Discussões

Nesta seção serão apresentados os resultados para a resposta individual de cada tipo de controlador (PID modificado, *Fuzzy* e *Neurofuzzy*) para uma perturbação e, por fim, a comparação entre estes. Ao final de cada subseção é feita uma análise qualitativa a respeito do resultado de cada controlador. Na última subseção é realizada a inferência quantitativa.

4.1 Resultados e discussões para os Controladores *Fuzzy*

Logo após a implementação dos controladores no microcontrolador, percebeu-se que seriam necessários ajustes às faixas das funções de pertinência. Sobre a ação do controlador *Fuzzy* com 3 faixas, percebeu-se que os motores do veículo não apresentavam torque suficiente para correção de erros pequenos. Para correção de tal, foi decisivo o encurtamento da faixa relacionada a erros e variações de erro pequenas, ação esta que permitiu aumentar o sinal de controle de saída da lógica. Já para o controlador *Fuzzy* com 5 faixas, foi necessário o deslocamento da faixa Grande para a variável sinal de controle. A mesma foi deslocada para valores de menor magnitude, visto que o veículo apresentou grandes dificuldades para se acomodar diante de erros e variações consideradas altas.

Depois de alguns testes práticos com o veículo, pôde-se alcançar o melhor desempenho possível para os dois controladores. As Figuras 31 e 32 trazem a configuração final das funções de pertinência para as variáveis de entrada e saída dos controladores *Fuzzy* com 3 e 5 faixas, respectivamente.

Figura 30 - Configuração final das funções de pertinência das variáveis de entrada e saída para o controlador *Fuzzy* com 3 faixas

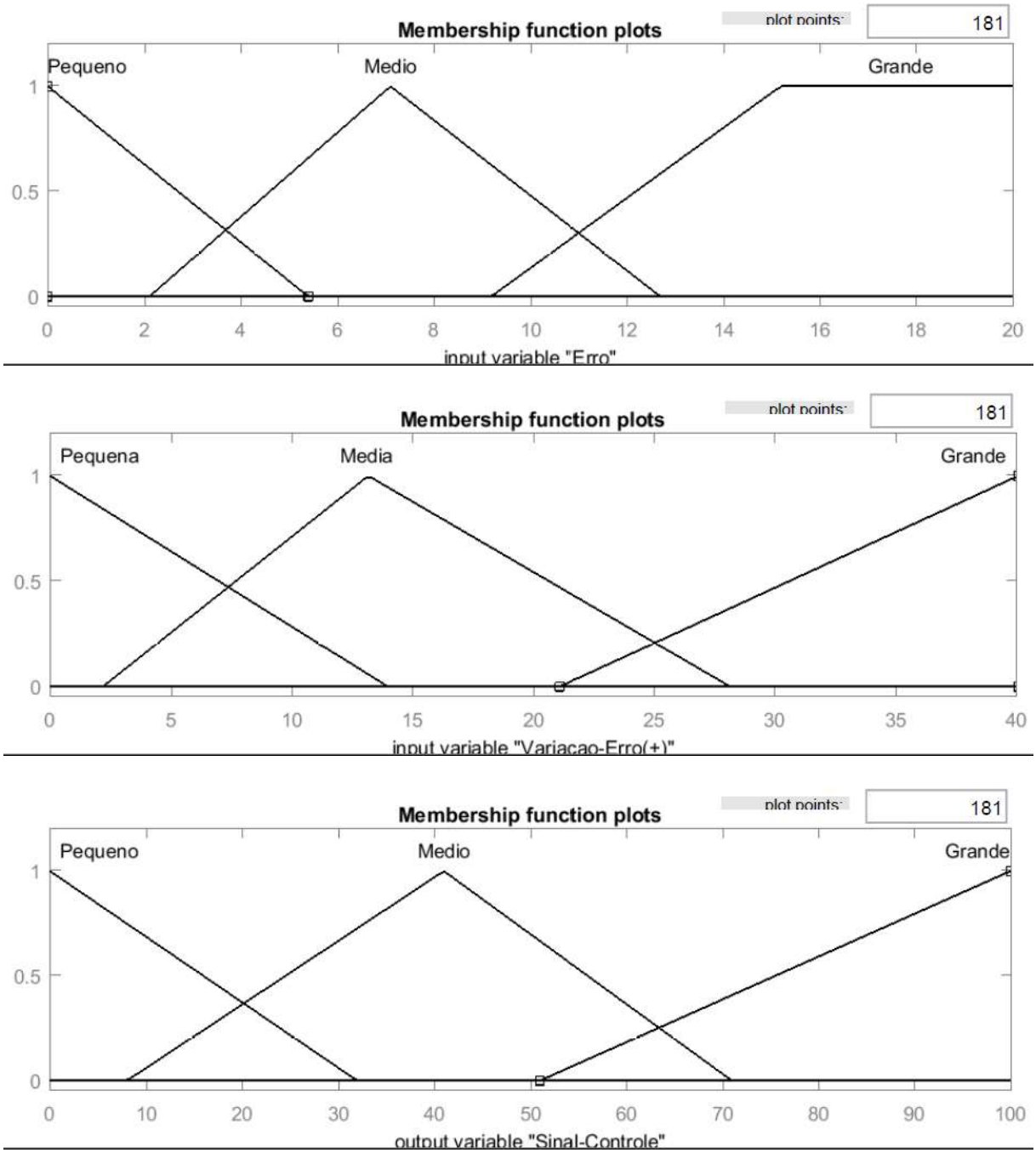
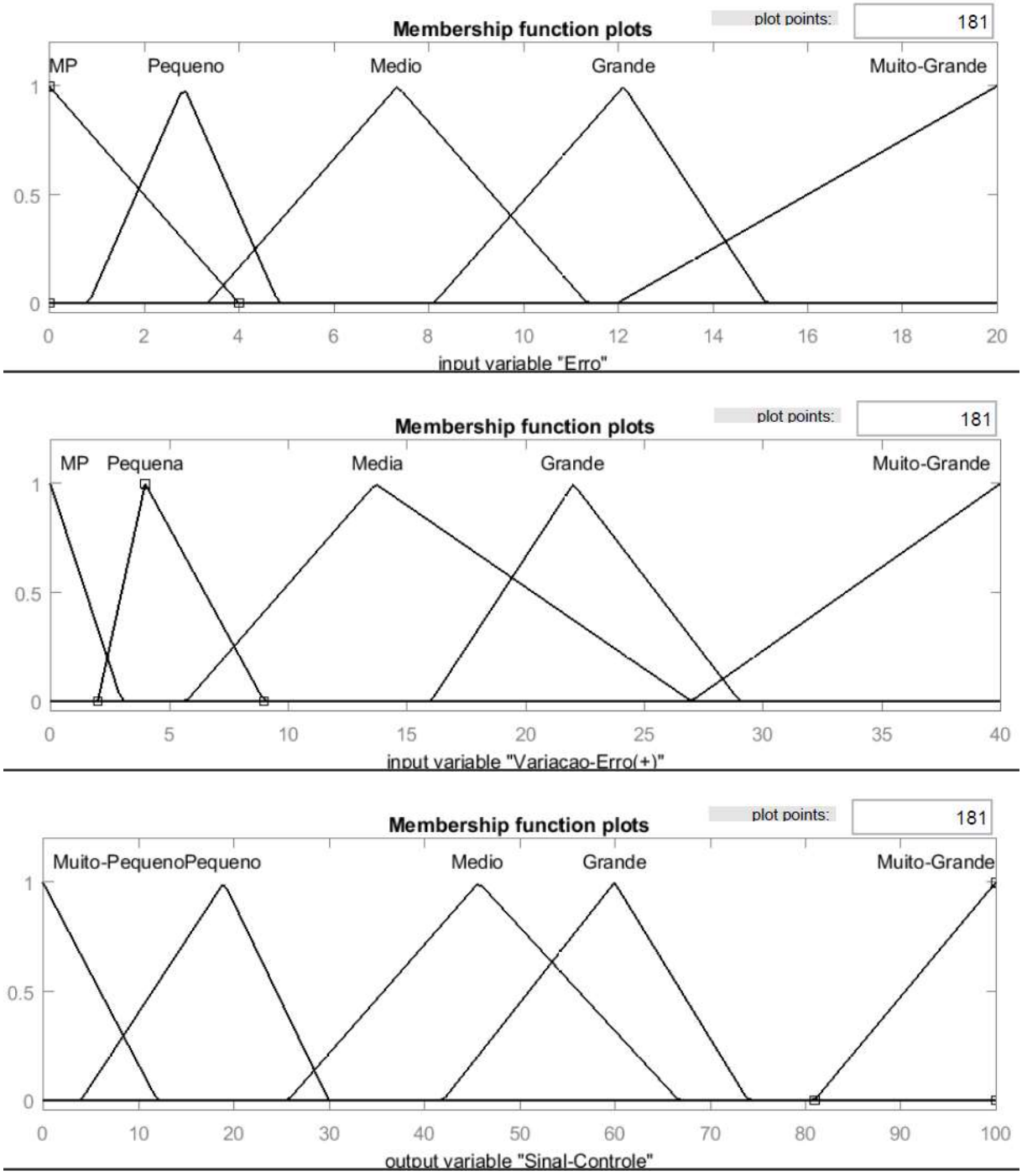


Figura 31 - Configuração final das funções de pertinência das variáveis de entrada e saída para o controlador *Fuzzy* com 5 faixas



Feitos os ajustes, para coleta dos resultados finais dos controladores, optou-se pela realização da média aritmética das medições de cada instante amostrado para 5 ensaios em sequência, excluindo-se a medição com o maior desvio. Ou seja, foram feitos cinco ensaios de 15 segundos para cada controlador, com um total de 150 medições ao final de cada ensaio

(tempo de amostragem = 150 ms). Excluída a medida considerada espúria (maior desvio), realizou-se novamente a média aritmética das 4 medições restantes para auferição final dos resultados. Pôde-se plotar, então, um gráfico da posição angular do veículo (em relação ao eixo vertical) em função do tempo. Salienta-se que a perturbação foi inserida no instante 1,9 s. As Figuras 33 e 34 mostram os resultados para os controladores *Fuzzy* com 3 e 5 faixas, respectivamente.

Figura 32 - Gráfico da posição angular em função do tempo para o veículo sobre ação do controlador *Fuzzy* com 3 faixas

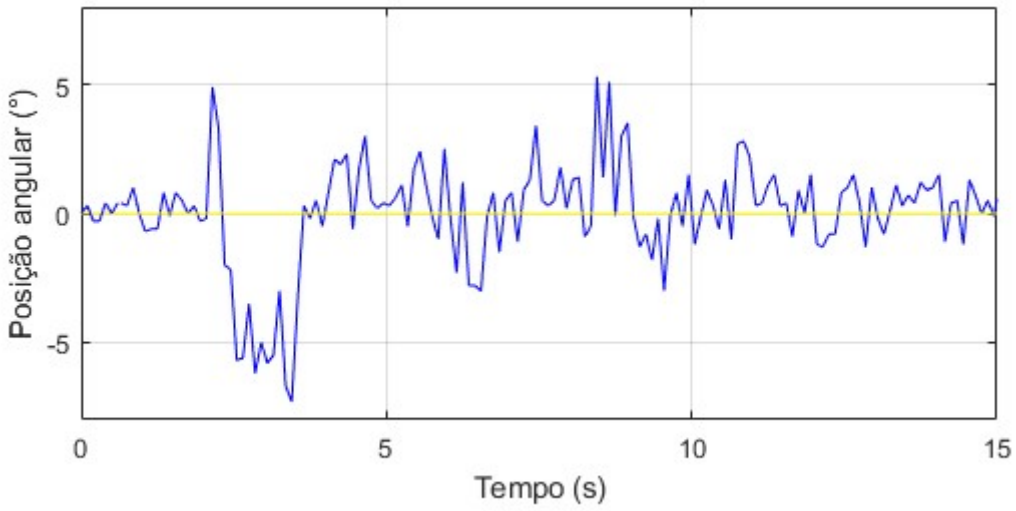
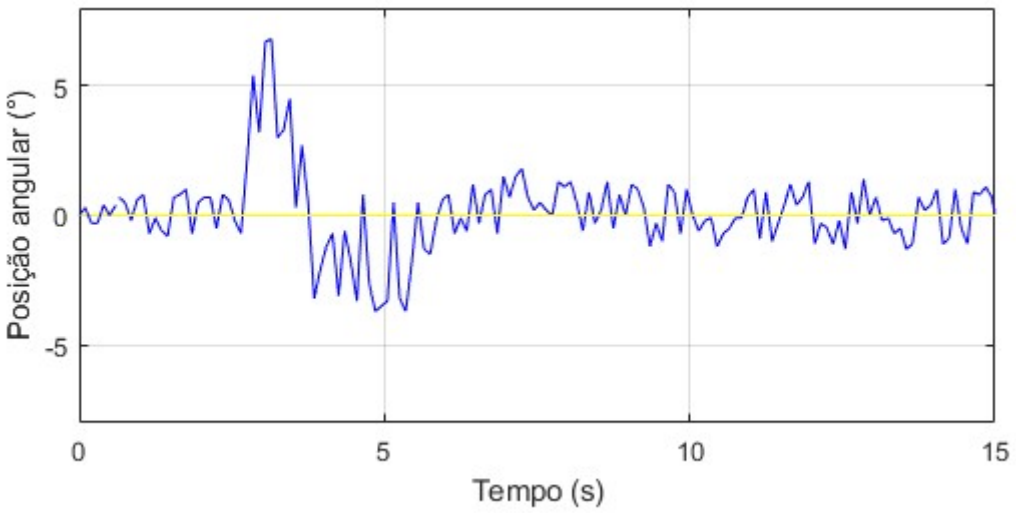


Figura 33 - Gráfico da posição angular em função do tempo para o veículo sobre ação do controlador *Fuzzy* com 5 faixas



Para uma melhor visualização dos resultados, foi realizada uma filtragem digital dos dados coletados, através de um filtro *Butterworth* de ordem 4 e frequência de corte de 50 Hz.

As Figuras 35 e 36 mostram os resultados para a planta sobre a ação dos controladores após a filtragem dos dados.

Figura 34 - Média das medidas para o controlador *Fuzzy* com 3 faixas após a filtragem

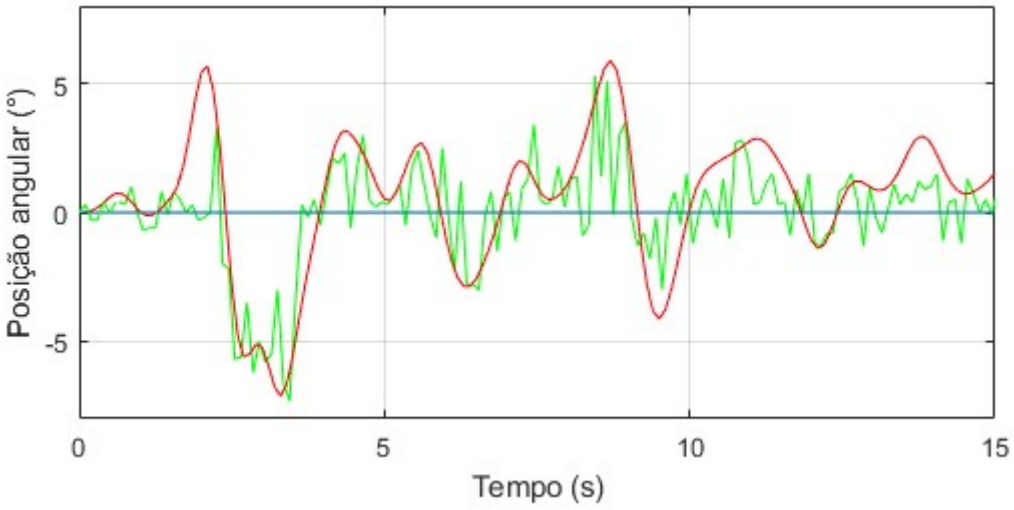
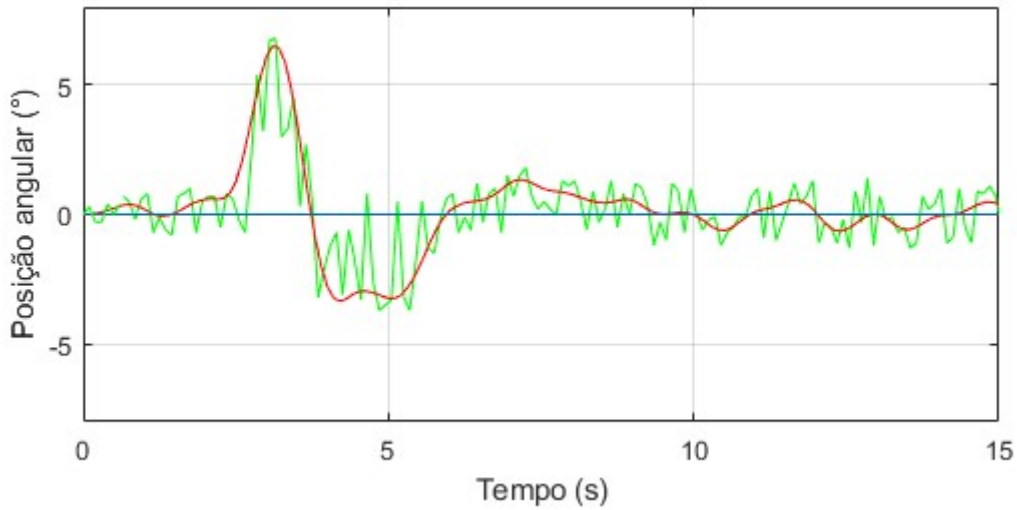


Figura 35 - Média das medidas para o controlador *Fuzzy* com 5 faixas após a filtragem

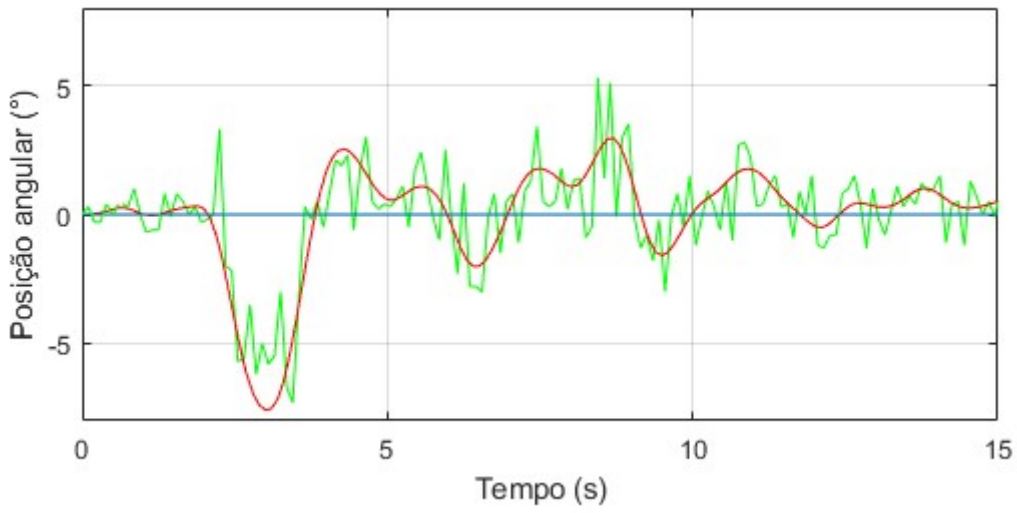


Analisando-se os resultados, percebe-se que o aumento do número de faixas para as variáveis da lógica *fuzzy* e consequente aumento do número de regras ajudou a melhorar o desempenho do sistema em termos de *overshoot*, tempo de acomodação e erro em regime. Nota-se ainda que o controlador *fuzzy* com 3 faixas teve oscilações após ter entrado em regime, o que pode ser explicado pelo fato de o controlador possuir apenas uma faixa para erro e variações de erro pequenas.

4.2 Resultados e discussões para o Controlador PID modificado

Após a implementação do controlador projetado na subseção 3.2 no microcontrolador, foram definidas as diretrizes para os ensaios, da mesma maneira que para os controladores *fuzzy* da última subseção. Ou seja, foram realizados 5 ensaios de 15 segundos (150 medições de ângulo) retirando-se a medida com maior desvio entre elas. Após foi feita a média aritmética entre as 4 restantes, a fim de se graficar os resultados de posição angular do veículo em função do tempo amostrado, como segue na Figura 37. Os resultados são mostrados com os dados já filtrados, para melhor visualização.

Figura 36 - Posição angular do veículo sob a ação do controlador PID em função do tempo amostrado



Analisando-se a Figura 37, percebe-se que o controlador PID projetado foi capaz de estabilizar a planta em estudo, obtendo resultados satisfatórios para o tempo de acomodação e erro em regime, apesar de apresentar um *overshoot* considerável.

4.3 Resultados e discussões para o Controlador *Neurofuzzy*

Realizado o treinamento da rede do controlador híbrido, pôde-se perceber mudanças em relação às delimitações das faixas das funções de pertinência das variáveis de entrada do controlador *Fuzzy* com cinco faixas (as mudanças para os dois bancos de dados foram muito similares, portanto optou-se pela análise do treinamento feito à base dos dados do PID). O ajuste "inteligente" pode ser observado pela comparação das funções de pertinência dos controladores mostrada na Figura 38 (variável erro) e 39 (variável variação do erro).

Figura 37 - Comparação do ajuste empírico (acima) com o ajuste inteligente (abaixo) para a variável erro

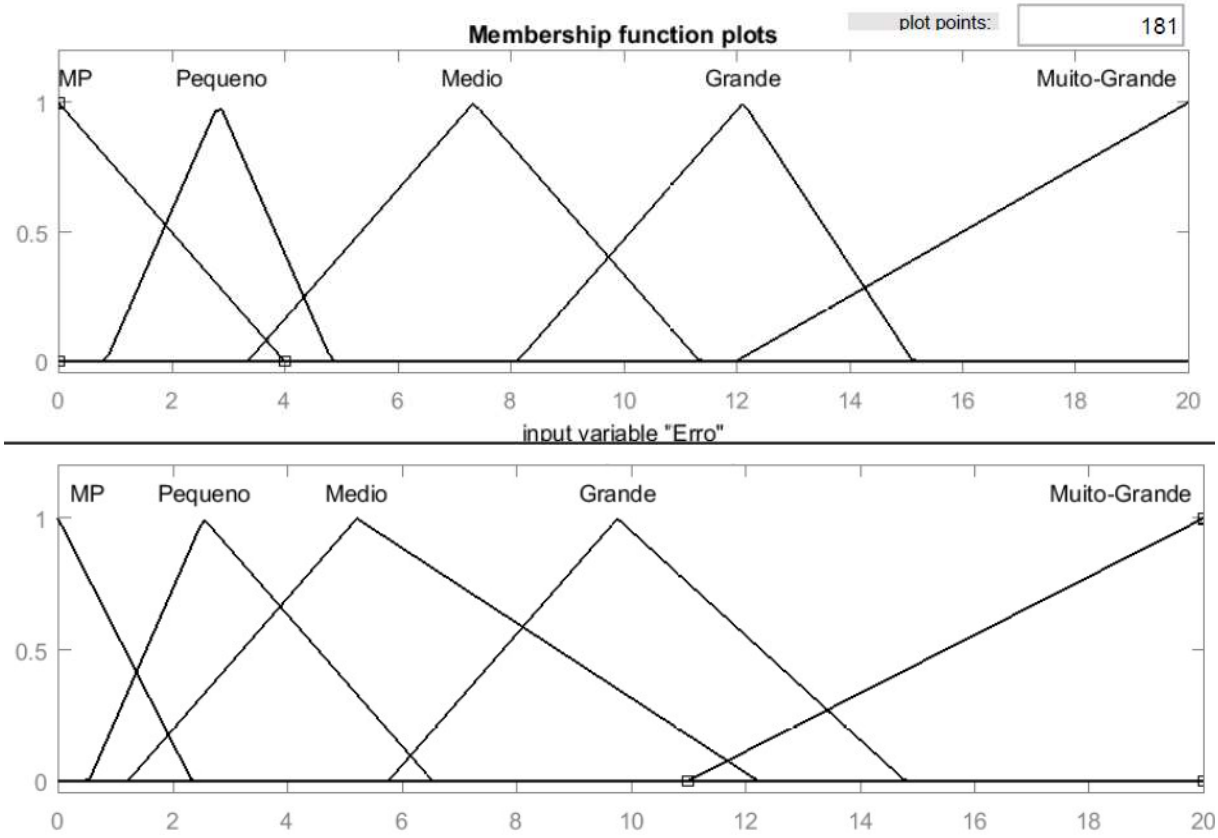
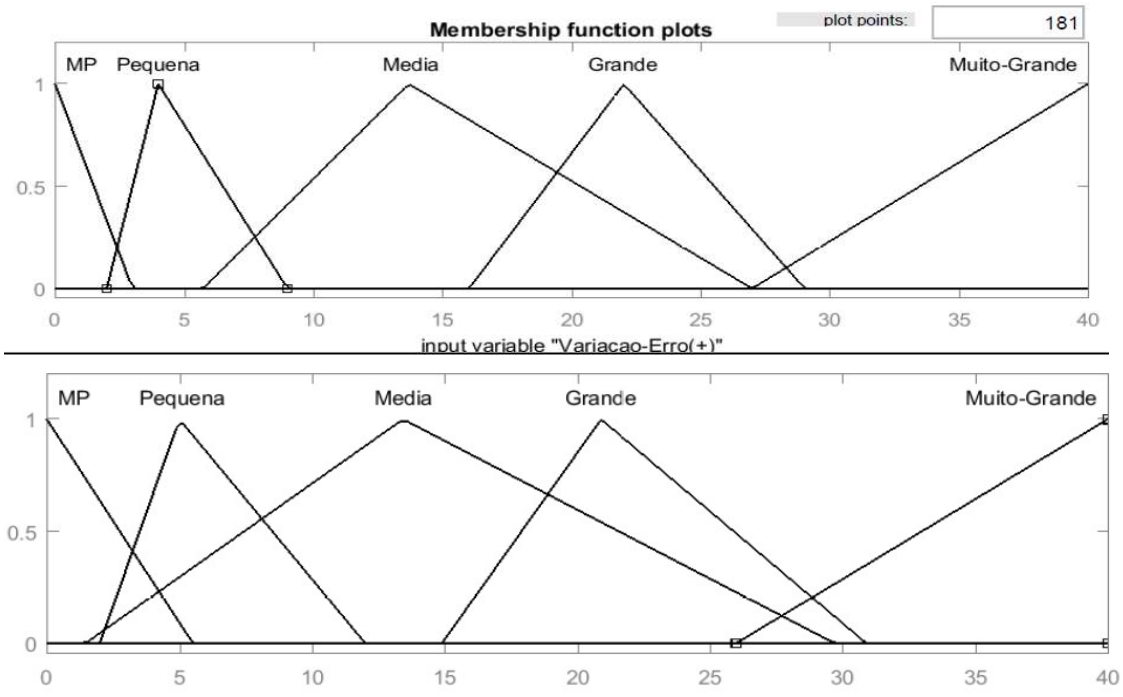


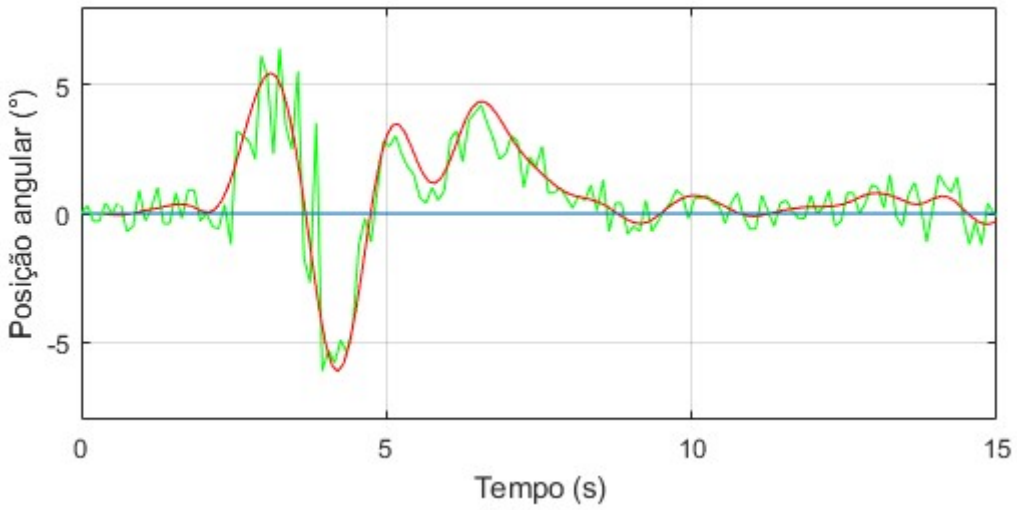
Figura 38 - Comparação do ajuste empírico (acima) com o ajuste inteligente (abaixo) para a variável variação do erro



Analisando-se a comparação dos ajustes, percebe-se que houve duas mudanças significativas em ambas as funções de pertinência (erro e variação do erro). A primeira delas foi em relação a faixa central, a qual foi “alargada”, de forma a cobrir um maior número de ângulos. Nota-se também que há um maior número de cruzamentos entre as regiões, fato este já estudado e inferido como positivo em trabalhos anteriores [9].

O procedimento para os ensaios do controlador *Neurofuzzy* foram os mesmos feitos para os outros controlares. Os resultados (com o banco de dados do PID), seguidos de filtragem, são mostrados na Figura 40.

Figura 39 - Posição angular em função do tempo para o veículo sob atuação do controlador *Neurofuzzy*



Percebe-se, analisando a Figura 40, que o controlador *Neurofuzzy* foi capaz de estabilizar a planta com sucesso. Os resultados quantitativos para este controlador serão discutidos na próxima subseção, juntamente com a comparação com os demais controladores implementados.

4.4 Comparação dos resultados dos controladores

A Figura 41 mostra a comparação dos resultados para os controladores *Fuzzy* e PID. A Tabela 3 traz os resultados individuais para cada controlador, tomando-se como parâmetros o tempo de acomodação, o máximo *overshoot* e o erro em regime.

Figura 40 - Comparação dos controladores *Fuzzy* e PID após filtragem

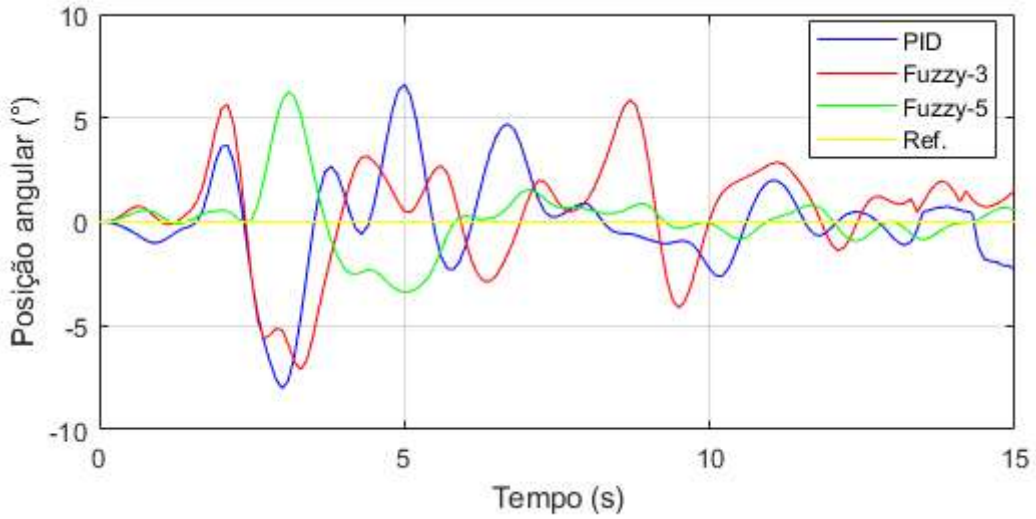


Tabela 7 - Resultados quantitativos para os controladores *Fuzzy* e PID

Controlador	Tempo de acomodação	Máximo *overshoot*	Erro em regime
Fuzzy com 3 faixas	9,82 segundos	7,11 graus	3,09 graus
Fuzzy com 5 faixas	7,21 segundos	6,27 graus	1,33 graus
PID	7,36 segundos	8,02 graus	2,16 graus

Analisando o gráfico da Figura 41 e os dados da Tabela 3, pode-se inferir que o controlador *Fuzzy* com 5 faixas apresenta um melhor desempenho referente aos três parâmetros em análise (tempo de acomodação, máximo *overshoot* e erro em regime), quando comparado aos outros dois controladores. Já o controlador *Fuzzy* com três faixas demonstrou menos eficácia em relação ao controlador PID nos quesitos tempo de acomodação e erro em regime, tendo um melhor desempenho apenas em relação ao máximo *overshoot*.

Seguindo-se com as comparações, preferiu-se confrontar os resultados do controlador *Neurofuzzy* apenas com os do controlador *Fuzzy* com 5 faixas, haja vista que este último demonstrou ser o melhor em relação aos outros dois controladores. A Figura 42 mostra os resultados para a comparação entre os controladores *Neurofuzzy* e *Fuzzy* com 5 faixas. Já a Tabela 4 traz os parâmetros para cada um dos dois controladores.

Figura 41 - Comparação entre os controladores *Neurofuzzy* e *Fuzzy* com 5 faixas

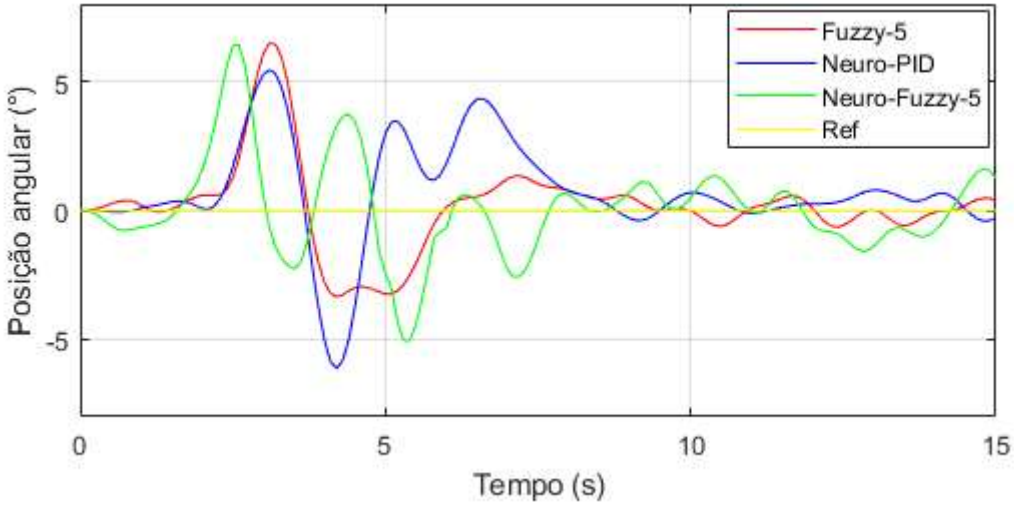


Tabela 8 - Resultados quantitativos para os controladores *Neurofuzzy* e *Fuzzy* com 5 faixas

Controlador	Tempo de acomodação	Máximo *overshoot*	Erro em regime
Neurofuzzy (PID)	8,02 segundos	6,10 graus	1,04 graus
Neurofuzzy (*Fuzzy*)	7,44 segundos	6,22 graus	1,68 graus
Fuzzy com 5 faixas	7,21 segundos	6,27 graus	1,33 graus

Analisando-se os dados da Figura 42 e os dados da Tabela 4, percebe-se que o desempenho de todos os controladores é muito similar nos três parâmetros em análise. Tanto o controlador *Fuzzy* com 5 faixas, quanto os controladores inteligente apresentaram uma estabilidade satisfatória para o equilíbrio frontal do veículo.

5 Conclusões

O trabalho aqui apresentado permitiu avaliar o desempenho de três diferentes tipos de controladores (PID modificado, *Fuzzy* e *Neurofuzzy*) para o problema do equilíbrio frontal de um veículo de duas rodas. Comparando-se os resultados dos controladores *Fuzzy* com 3 e 5 faixas para o problema em questão, pôde-se concluir que com o aumento do número de faixas para cada variável de entrada e saída da lógica (e consequente aumento no número de regras), houve um melhor desempenho no equilíbrio do veículo diante de perturbações.

Outro fato estudado foi o efeito do treinamento para um controlador híbrido. O controlador inteligente *Neurofuzzy* demonstrou eficácia similar à do controlador *Fuzzy* com 5 faixas, quando treinado com pares exemplos dos controladores PID modificado e *Fuzzy* (5 faixas). Pôde-se observar a diferença entre um ajuste empírico e o resultante do *Neurofuzzy* dos parâmetros das funções de pertinência das variáveis de entrada da lógica. Entretanto, analisando-se os resultados de comparação entre os controladores *Neurofuzzy* e *Fuzzy* (5 faixas), percebe-se que não se justifica o uso do controlador *neurofuzzy* para o veículo de duas rodas, visto que o controlador *Fuzzy* obteve melhores resultados. Isso pode ser explicado por dois pontos principais. Primeiro, os ajustes feitos às faixas durante os ensaios tornaram o controlador *Fuzzy* o melhor possível. E segundo, a faixa da variável de entrada ângulo era restrita. Para aplicações comerciais, possivelmente seria justificável o uso do controlador *Neurofuzzy* para a planta em estudo.

Por fim, constatou-se que o controlador PID modificado obteve um desempenho intermediário entre os controladores *Fuzzy*. Em relação ao de 3 faixas, este controlador obteve melhores resultados para o tempo de acomodação e erro em regime. Já quando comparado ao de 5 faixas, percebeu-se que seu desempenho ficou um pouco abaixo, em relação aos três parâmetros avaliados. Entretanto, haja vista que o custo computacional de implementação do controlador PID é menor em relação aos controladores *Fuzzy*, este ainda demonstra ser uma boa opção para solução do problema do equilíbrio frontal de veículos de duas rodas.

Para trabalhos futuros, seria interessante a investigação de outras técnicas para melhoria de desempenho da planta diante de perturbações. Controladores como o LQR, o híbrido PID-*Fuzzy* e mesmo o *Neurofuzzy* com treinamento não supervisionado seriam boas apostas para tentativas de melhoria de desempenho da planta. Outra possível investigação seria a análise da retirada de uma das rodas do veículo, transformando o mesmo em um monociclo. Isso expandiria o campo de estudo, visto que outro atuador seria necessário para o equilíbrio lateral.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] BASNAYAKE, I.; MADHUSHANI, T.; *Intrinsic PID Controller for a Segway type model robot.* ICII, 2014, Austrália.

[2] DESERNO, T.; PAPLINSKI, A.; *Self organising map based region of interest labelling for automated defect identification in large sewer pipe image collections*. International Joint Conference on Neural Networks (IJCNN), 2012

[3] PAVITHRA, S.; SIVA, A.; *7TH sense-a multiporpose robot for milatary*. International Conference on Information Communication and Embeeded Systems (ICICES), 2013.

[4] HOLK, M.; *Interactive three dimensional presentation of Segway laboratory model*. International Conference on Emerging eLearning Technologies and applications (ICETA), 2016.

[5] JIANG, H.; BRAZIS, P.; *Safety considerations of wireless charger for electric vehicles*. Symposium on product compliance engineering proceedings, IEEE, 2012.

[6] JUANG, H.-S., & Lum, K.-Y. (2013). *Design and Control of a Two-Wheel Self-Balancing Robot using the Arduino Microcontroller Board*. 10th IEEE International Conference on Control and Automation (ICCA). Hangzhou, China

[7] PRAKASH, K., & Thomas, K. (2016). *Study of Controllers for a Two Wheeled Self-balancing Robot*. International Conference on Next Generation Intelligent Systems (ICNGIS). Kollam, India.

[8] PALIWAL, S.; CHOPRA, V. *Stabilization of Mobile Inverted Pendulum Using Fuzzy PID Controllers*. 10th IEEE International Conference on Control and Automation (ICCA). Hangzhou, China.

[9] AKMAL, M.; *Fuzzy Logic Controller for Two Wheleed EV3 lego robot*. Conference on Systems, Process and Control, IEEE, 2017

[10] Modelagem retirada do site: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlPID

[11] BAZANELLA, A.S.; GOMES, J.M. *Sistemas de Controle: Princípios e Métodos de Projetos*. UFRGS, 2003. Citado nas páginas 97 e 140

[12] OGATA, K.; *Modern control engineering*. [S.l.]: Prentice Hall, 2001.

[13] HAJEC, P.; *Fuzzy Logic: Voice of Stanford Encyclopedia of Philosophy*, 2006.

[14] NAUCK, D.; KRUSE, R.; *Neurofuzzy control based on the NEFCON Model: Recent developments,*soft computing 2, páginas 167 a 181, 1999.

[15] RODRIGUES, M.C; *Técnicas Inteligentes Híbridas para o Controle de Sistemas Não Lineares*.2006. Tese de mestrado, Programa de Pós-Graduação em Engenharia Elétrica, UFRN. Citado na página 46

[16] REZENDE, J.; MAITELLI, A.; *Um estudo Comparativo entre diferentes técnicas de Otimização do treinamento de Neurocontroladores;* Congresso Brasileiro de Redes Neurais, São José dos Campos, SP, 1999.

ANEXO A Código da implementação do controlador *Fuzzy*

```
#include <stdio.h>
#include <stdlib.h>
#include "fuzzy_defuzzy.h"

int main()
{
//Declaração das variáveis
float erro_atual, Var_erro, output, Wx1[3], Wx2[3], Wx3[3], WR[9];
int i;
float erro_anterior = 0;

 //Três triângulos para o erro, sendo o range total deste de 0 a 30
graus
float a1[3] = {0, 4, 12};    //erro pequeno
float b1[3] = {4, 8, 20};  //erro médio
float c1[3] = {8, 15, 30}; //erro grande

//Três triângulos para a Var_erro positiva, sendo o range total para
este de 0 a 30 graus
float a2[3] = {0, 4, 12};
float b2[3] = {4, 8, 20};
float c2[3] = {8, 15, 30};

//Três triângulos para a Var_erro negativa, sendo o range total para
este de 0 a 30 graus
float a3[3] = {-8, -15, -30};
float b3[3] = {-4, -8, -20};
float c3[3] = {0, -4, -12};

while(1){
//Captação do erro e da Var_erro
printf("\n---Range da variavel erro: 0 a 30 graus---\n");
printf("Digite o erro:\n");
scanf("%f", &erro_atual);
//printf("Digite a variacao do erro:\n");
//scanf("%f", &Var_erro);

Var_erro = -1*(erro_atual - erro_anterior);
if(Var_erro == 0) Var_erro = 1;
        //Fuzzyficação dos pesos através da chamada da função, contida
no outro header
        Wx1[0] = fuzzyficacao(0, erro_atual, a1, b1, c1);
        Wx1[1] = fuzzyficacao(1, erro_atual, a1, b1, c1);
        Wx1[2] = fuzzyficacao(2, erro_atual, a1, b1, c1);
        Wx2[0] = fuzzyficacao(0, Var_erro, a2, b2, c2);
        Wx2[1] = fuzzyficacao(1, Var_erro, a2, b2, c2);
        Wx2[2] = fuzzyficacao(2, Var_erro, a2, b2, c2);
        Wx3[0] = fuzzyficacao(0, Var_erro, a3, b3, c3);
        Wx3[1] = fuzzyficacao(1, Var_erro, a3, b3, c3);
        Wx3[2] = fuzzyficacao(2, Var_erro, a3, b3, c3);

        //Definição das regras (causalidade) e inferência pela
multiplicacão dos pesos
        if(Var_erro < 0){
```

```
                for(i=0;i<3;i++){
                    if((Wx1[0] >= Wx3[i]) && ((Wx1[0]*Wx3[i]) != 0))
WR[i] = Wx1[0];
                    else if((Wx1[0] < Wx3[i]) && ((Wx1[0]*Wx3[i]) !=
0)) WR[i] = Wx3[i];
                    else WR[i] = 0;
                }

                for(i=3;i<6;i++){
                    if((Wx1[1] >= Wx3[i-3]) && ((Wx1[1]*Wx3[i-3]) !=
0)) WR[i] = Wx1[1];
                    else if((Wx1[1] < Wx3[i-3]) && ((Wx1[1]*Wx3[i-3])
!= 0)) WR[i] = Wx3[i-3];
                    else WR[i] = 0;
                }

                for(i=6;i<9;i++){
                    if((Wx1[2] >= Wx3[i-6])  && ((Wx1[2]*Wx3[i-6]) !=
0)) WR[i] = Wx1[2];
                    else if((Wx1[2] < Wx3[i-6]) && ((Wx1[2]*Wx3[i-6])
!= 0)) WR[i] = Wx3[i-6];
                    else WR[i] = 0;
                }

        }

        else{

        for(i=0;i<3;i++){
                    if((Wx1[0] >= Wx2[i]) && ((Wx1[0]*Wx2[i]) != 0))
WR[i] = Wx1[0];
                    else if((Wx1[0] < Wx2[i]) && ((Wx1[0]*Wx2[i]) !=
0)) WR[i] = Wx2[i];
                    else WR[i] = 0;
                }

                for(i=3;i<6;i++){
                    if((Wx1[1] >= Wx2[i-3]) && ((Wx1[1]*Wx2[i-3]) !=
0)) WR[i] = Wx1[1];
                    else if((Wx1[1] < Wx2[i-3]) && ((Wx1[1]*Wx2[i-3])
!= 0)) WR[i] = Wx2[i-3];
                    else WR[i] = 0;
                }

                for(i=6;i<9;i++){
                    if((Wx1[2] >= Wx2[i-6])  && ((Wx1[2]*Wx2[i-6]) !=
0)) WR[i] = Wx1[2];
                    else if((Wx1[2] < Wx2[i-6]) && ((Wx1[2]*Wx2[i-6])
!= 0)) WR[i] = Wx2[i-6];
                    else WR[i] = 0;
                }
        }
        //Processo de defuzzyficação = geração do sinal de saída
        output = defuzzyficacao(WR, Var_erro);

        printf("\n");
```

```
        for(i=0;i<9;i++){
            printf("WR[%d] = %f\n", i, WR[i]);
        }

        printf("\nErro = %f\n\n", erro_atual);
        printf("\nVariacao do Erro = %f\n\n", Var_erro);
        printf("\nPWM = %f\n\n", output);
        erro_anterior = erro_atual;
}
}
float fuzzyficacao(int k, float input, float *a,float *b, float *c)
{
    float dom;
    if ( input > a[k] && input < b[k]) // Região da reta inicial do
triângulo
    {
        dom = (input - a[k])/(b[k] - a[k]);
    }
    else
        if (input > b[k] && input < c[k]) // Região da reta final do
triângulo
        {
            dom = (c[k] - input)/(c[k] - b[k]);
        }
    else
        if (input == b[k]) // Aqui há a coinscidência da variável cair
no bico do triângulo!
        {
            dom = 1.0;
        }
    else
    {
        dom = 0; // Aqui a variável encontra-se fora da faixa
    }
    return dom; //Retorna o grau da função associada (DoM)
}
float defuzzyficacao(float *WR, float Var_erro)
{
    float Centroide = 0;
    int i;
    float Area_total = 0;
    float Numerador = 0;
    float a[7] = {0,5,10,15,35,55,65};
    float b[7] = {0,10,15,25,45,65,80};
    float c[7] = {0,15,20,35,55,75,95};
    //Faixas para o sinal de controle

    float A[9] = {0,0,0,0,0,0,0,0,0};

    if(Var_erro < 0){

    //o centroide de cada faixa é o próprio b[i]
    //cálculo da área da faixa: A = h *(B+b) / 2
    A[0] = WR[0] * (c[0] - a[0] + ((c[0] - a[0]) - 2*WR[0]*(b[0]-
a[0])));
    Numerador = Numerador + A[0]*b[0];
```

```
    A[1] = WR[1] * (c[1] - a[1] + ((c[1] - a[1]) - 2*WR[1]*(b[1]-
a[1])));
    Numerador = Numerador + A[1]*b[1];
    A[2] = WR[2] * (c[4] - a[4] + ((c[4] - a[4]) - 2*WR[2]*(b[4]-
a[4])));
    Numerador = Numerador + A[2]*b[4];
    A[3] = WR[3] * (c[2] - a[2] + ((c[2] - a[2]) - 2*WR[3]*(b[2]-
a[2])));
    Numerador = Numerador + A[3]*b[2];
    A[4] = WR[4] * (c[3] - a[3] + ((c[3] - a[3]) - 2*WR[4]*(b[3]-
a[3])));
    Numerador = Numerador + A[4]*b[3];
    A[5] = WR[5] * (c[4] - a[4] + ((c[4] - a[4]) - 2*WR[5]*(b[4]-
a[4])));
    Numerador = Numerador + A[5]*b[4];
    A[6] = WR[6] * (c[5] - a[5] + ((c[5] - a[5]) - 2*WR[6]*(b[5]-
a[5])));
    Numerador = Numerador + A[6]*b[5];
    A[7] = WR[7] * (c[6] - a[6] + ((c[6] - a[6]) - 2*WR[7]*(b[6]-
a[6])));
    Numerador = Numerador + A[7]*b[6];
    A[8] = WR[8] * (c[6] - a[6] + ((c[6] - a[6]) - 2*WR[8]*(b[6]-
a[6])));
    Numerador = Numerador + A[8]*b[6];

    for(i=0;i<9;i++){
    Area_total = Area_total + A[i];
    }

    if(Area_total == 0)
        Centroide = 0;
    else
    //cálculo do centroide no final
    Centroide = Numerador/Area_total;

    }
    else{
    A[0] = WR[0] * (c[0] - a[0] + ((c[0] - a[0]) - 2*WR[0]*(b[0]-
a[0])));
    Numerador = Numerador + A[0]*b[0];
    A[1] = WR[1] * (c[0] - a[0] + ((c[0] - a[0]) - 2*WR[1]*(b[0]-
a[0])));
    Numerador = Numerador + A[1]*b[0];
    A[2] = WR[2] * (c[0] - a[0] + ((c[0] - a[0]) - 2*WR[2]*(b[0]-
a[0])));
    Numerador = Numerador + A[2]*b[0];
    A[3] = WR[3] * (c[1] - a[1] + ((c[1] - a[1]) - 2*WR[3]*(b[1]-
a[1])));
    Numerador = Numerador + A[3]*b[1];
    A[4] = WR[4] * (c[2] - a[2] + ((c[2] - a[2]) - 2*WR[4]*(b[2]-
a[2])));
    Numerador = Numerador + A[4]*b[2];
    A[5] = WR[5] * (c[1] - a[1] + ((c[1] - a[1]) - 2*WR[5]*(b[1]-
a[1])));
    Numerador = Numerador + A[5]*b[1];
    A[6] = WR[6] * (c[4] - a[4] + ((c[4] - a[4]) - 2*WR[6]*(b[4]-
a[4])));
```

```
    Numerador = Numerador + A[6]*b[4];
    A[7] = WR[7] * (c[4] - a[4] + ((c[4] - a[4]) - 2*WR[7]*(b[4]-
a[4])));
    Numerador = Numerador + A[7]*b[4];
    A[8] = WR[8] * (c[3] - a[3] + ((c[3] - a[3]) - 2*WR[8]*(b[3]-
a[3])));
    Numerador = Numerador + A[8]*b[3];

        for(i=0;i<9;i++){
    Area_total = Area_total + A[i];
    }

    if(Area_total == 0)
        Centroide = 0;
    else
    //cálculo do centroide no final
    Centroide = Numerador/Area_total;


    }

 return (Centroide);

}
```

ANEXO B Código da implementação do controlador *Neurofuzzy*

```
//Segue o controlador NeuroFuzzy com o algorítmo de aprendizagem Backpropagation
//Serão um total de 6 camadas, sendo uma de entrada, uma de saída e 4 intermediárias
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/*
Faixas de erro e variação de erro(variáveis de entrada da lógica fuzzy)-definição lógica/qualitativa INICIAL

erro pequeno          0 a 20
erro médio            0 a 20
erro grande           10 a 23
Var_erro pequeno      20 a 26
Var_erro médio        23 a 30
Var_erro grande       0 a 20
*/

//OBS:não é necessária a definição pras faixas da saída, visto que se utilizarão os polinômios de Sugeno para interpolação/Defuzzyficação

//Função de Fuzzyficação
// A função recebe como parâmetros a entrada, os pontos da faixa em questão e valor da faixa
// Os "if(s)" conferem em qual região a variável se encontra dentro das regiões possíveis do triângulo
// Após calcula-se o quanto porcento ela representa esta faixa (tamanho da reta hor. definido pela variável sobre tamanho total da reta)
// Ressalta-se que tal operação é possível haja vista a linearidade da função triangular!

float fuzzify(int k, float input, float *a,float *b, float *c)
{
    float dom;
```

```
          if ( input > a[k] && input < b[k]) // Região da reta inicial do
triângulo
          {
              dom = (input - a[k])/(b[k] - a[k]);
          }
          else
              if (input > b[k] && input < c[k]) // Região da reta final
do triângulo
              {
                  dom = (c[k] - input)/(c[k] - b[k]);
              }
          else
              if (input == b[k]) // Aqui há a coinscidência da variável
cair no bico do triângulo!
              {
                  dom = 1.0;
              }
          else
          {
              dom = 0; // Aqui a variável encontra-se fora da faixa
          }
          return dom; //Retorna o grau da função associada (DoM)
      }


      int main()
      {

      //Definição das variáveis do código
      int i = 0, z = 0;
      float   erro,   Var_erro,   Wx1[3],   Wx2[3],WR[9],Wn[9],   WnT,
Polinomio_Sugeno[9], ref, saida, desejado;
      float   Error5,   Error4[9],   Error3[9],   Error2[9],   Error1[6],
DER_POL_p[9], DER_POL_q[9], DER_POL_r[9];
      float     DER_TRIe_a1[3],     DER_TRIe_b1[3],     DER_TRIe_c1[3],
DER_TRIV_a2[3], DER_TRIV_b2[3], DER_TRIV_c2[3], erro_aux1, erro_aux2;
      float  VAR_a1[3],  VAR_b1[3],  VAR_c1[3],  VAR_a2[3],  VAR_b2[3],
VAR_c2[3], VAR_p[9], VAR_q[9], VAR_r[9];
```

```
//Inicialização de variáveis vetores que devem ter um valor inicial
float n = 0.1;
float Resto[9] = {0,0,0,0,0,0,0,0,0};
float Soma_VAR_a1[3] = {0, 0, 0};
float Soma_VAR_b1[3] = {0, 0, 0};
float Soma_VAR_c1[3] = {0, 0, 0};
float Soma_VAR_a2[3] = {0, 0, 0};
float Soma_VAR_b2[3] = {0, 0, 0};
float Soma_VAR_c2[3] = {0, 0, 0};
float Soma_VAR_p[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
float Soma_VAR_q[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
float Soma_VAR_r[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
//Definição das faixas "iniciais"(antes do aprendizado) de entrada

//Três triângulos para o erro
float a1[3] = {0, 3, 6};
float b1[3] = {3, 6, 10};
float c1[3] = {6, 10, 20};

//Três triângulos para a Var_erro
float a2[3] = {0, 3, 6};
float b2[3] = {3, 6, 10};
float c2[3] = {6, 10, 20};

//Atribuindo um valor inicial aos polinômios de Sugeno
float p[9] = {1,2,3,4,5,6,7,8,9};
float q[9] = {1,2,3,4,5,6,7,8,9};
float r[9] = {1,2,3,4,5,6,7,8,9};

//INÍCIO DO PROCESSO DE CRIAÇÃO DO MODELO ANFIS
/*
Camada 0 == 2 neurônios --> Leitura das entradas;
Camada 1 == 6 neurônios --> Fuzzyficação das variáveis (6 funcções
de pertinência no total);
Camada 2 == 9 neurônios --> Definição das 9 regras e dos pesos
equivalentes de cada regra (inferência probabilística);
```

```
      Camada 3 == 9 neurônios --> Normalização dos pesos associados a cada
regra;
      Camada 4 == 9 neurônios --> Associação dos polinômios de Sugeno a
cada regra (total de 9 polinômios);
      Camada 5 == 1 neurônio -->  Soma dos polinômios de Sugeno, resultando
na saída única do sistema;
      */

      // 'DO WHILE 2', referente a parada ou não do treinamento
      //do{ IMPORTANTE DESCOMENTAR ESTA LINHA PRO CÓDIGO FUNCIONAR!

      //"DO WHILE 1", referente ao treinamento da época x para que todos
os pares de exemplo treinem a rede;
      do{

      //Na etapa 1 (camada 0 == 2 neurônios) são lidas (pelo sensor) as
entradas erro e variação de erro

      //Aqui são fornecidos os pares "erro,saida_desejada(Ref)" e
"Var_erro, saida_desejada" para posterior treinamento da rede
      /*
      printf("\nDigite a o erro: ");
      scanf("%f", &erro);
      printf("\nDigite a variação do erro: ");
      scanf("%f", &Var_erro);
      printf("\nDigite a saida referencia para tal erro: ");
      scanf("%f", &ref);
      */
      if(z=0)
      {
      erro = 7.897;
      Var_erro = 5.647;
      ref = 30.87;
      }
      if(z=1)
      {
      erro = 1.788;
      Var_erro = 9.11;
```

```
ref = 15.65;
}
if(z=2)
{
erro = 15.78;
Var_erro = 19;
ref = 88.96;
}
if(z=3)
{
erro = 10.76;
Var_erro = 1.1;
ref = 17.56;
}
if(z=4)
{
erro = 5.67;
Var_erro = 15.876;
ref = 58.099;
}
if(z=5)
{
erro = 0.676;
Var_erro = 3.8;
ref = 9.54;
}
if(z=6)
{
erro = 18.96;
Var_erro = 5.647;
ref = 78.96;
}
if(z=7)
{
erro = 16.84;
Var_erro = 14.76;
ref = 90.66;
}
```

```
if(z=8)
{
erro = 2.76;
Var_erro = 10.90;
ref = 18.76;
}
if(z=9)
{
erro = 19.65;
Var_erro = 5.647;
ref = 77.201;
}
```

//Na etapa 2 (camada 1 == 6 neurônios) ocorre o processo de fuzzyficação das variáveis de entrada (3 faixas triangulares para cada uma)

//OBS: Para erro: "pequeno", "medio" e "grande"

```
Wx1[0] = fuzzify(0, erro, a1, b1, c1);
Wx1[1] = fuzzify(1, erro, a1, b1, c1);
Wx1[2] = fuzzify(2, erro, a1, b1, c1);
Wx2[0] = fuzzify(0, Var_erro, a2, b2, c2);
Wx2[1] = fuzzify(1, Var_erro, a2, b2, c2);
Wx2[2] = fuzzify(2, Var_erro, a2, b2, c2);
```

//Na etapa 3 (camada 2 == 9 neurônios) são definidas as regras (empiricamente), sendo um total de 3x3 = 9 regras

//OBS: o cálculo do peso equivalente para cada regra é feito pela multiplicação dos pesos associados a mesma

```
/*
if (erro == pequeno && Var_erro == pequeno)     {   WR[0] =
Wx1[0]*Wx2[0];}
else if (erro == pequeno && Var_erro == medio) {   WR[0] =
Wx1[0]*Wx2[1];}
else if (erro == pequeno && Var_erro == grande){   WR[2] =
Wx1[0]*Wx2[2];}
else if (erro == medio && Var_erro == pequeno) {   WR[3] =
Wx1[1]*Wx2[0];}
```

```
          else if (erro == medio && Var_erro == medio)    {   WR[4] =
Wx1[1]*Wx2[1];}
          else if (erro == medio && Var_erro == grande)   {   WR[5] =
Wx1[1]*Wx2[2];}
          else if (erro == grande && Var_erro == pequeno){   WR[6] =
Wx1[2]*Wx2[0];}
          else if (erro == grande && Var_erro == medio)   {   WR[7] =
Wx1[2]*Wx2[1];}
          else if (erro == grande && Var_erro == grande) {   WR[8] =
Wx1[2]*Wx2[2];}
          */

          WR[0] = Wx1[0]*Wx2[0];
          WR[1] = Wx1[0]*Wx2[1];
          WR[2] = Wx1[0]*Wx2[2];
          WR[3] = Wx1[1]*Wx2[0];
          WR[4] = Wx1[1]*Wx2[1];
          WR[5] = Wx1[1]*Wx2[2];
          WR[6] = Wx1[2]*Wx2[0];
          WR[7] = Wx1[2]*Wx2[1];
          WR[8] = Wx1[2]*Wx2[2];

     //Na etapa 4 (camada 3 == 9 neurônios) os pesos resultantes das
regras são normalizados para que sejam preparados para o método Sugeno

     for(i=0;i<9;i++){
     Wn[i]                                                          =
WR[i]/(WR[0]+WR[1]+WR[2]+WR[3]+WR[4]+WR[5]+WR[6]+WR[7]+WR[8]);
     }

     //Na etapa 5 (camada 4 == 9 neurônios) é definida a função associada
aos neurônios desta camada, segundo o modelo Sugeno
     //OBS: nota-se que haverá 9 funções associadas e não 7, pois nesse
caso estas estão ligadas às regras, e não à variável de saída em si

     for(i=0;i<9;i++){
```

```
      Polinomio_Sugeno[i]   =   Wn[i]*(p[i]*erro   +   q[i]*Var_erro   +
r[i]);//"p[i], q[i] e r[i]" são os parâmetros de ajuste dos polinômios de
Sugeno
      }

      //Na etapa 6 (camada 5 == 1 neurônio) é feita a soma do resultado
dos polinômios de Sugeno, que é a saída em si
      saida = 0;
      for(i=0;i<9;i++){
      saida = saida + Polinomio_Sugeno[i];
      }


      //INICIO DO ALGORÍTMO DE APRENDIZAGEM BACKPROPAGATION
      /*
      --> Cálculo da derivada do erro quadrático médio em cada camada da
rede, para cada neurônio, em relação à saída da camada em questão. Tudo
isso feito em relação a cada ponto;
      --> Cálculo da derivada da função de ativação em relação a cada
parâmetro.Isso tudo para cada camada e para cada ponto de treinamento;
      --> Cálculo da variação de cada parâmetro, utilizando-se para tal
as derivadas anteriormente calculadas;
      --> Cálculo da soma da variação de cada parâmetro;
      --> Atualização dos parâmetros das camadas 1 e 4 (Funções
triangulares e Polinômios de Sugeno) após o final da última época (fim do
treinamento) == APRENDIZAGEM!!!
      */

      //Na etapa 7 calcula-se a derivada do erro quadrático médio em
relação a referência (camada 5), a fim de "retropropagá-lo" para os
neurônios das outras camadas

       Error5 = -2*(ref - saida);

      //Na etapa 8, são calculadas as derivadasdos erros para os neurônios
da camada "4", os quais levarão o "Error5" da última camada

       Error4[0] = Error5;//Erros dos neurônios da camada 4
```

```
 Error4[1] = Error5;
 Error4[2] = Error5;
 Error4[3] = Error5;
 Error4[4] = Error5;
 Error4[5] = Error5;
 Error4[6] = Error5;
 Error4[7] = Error5;
 Error4[8] = Error5;
//Na etapa 9, são calculados as derivadas dos erros para os neurônios da camada "3", os quais levarão o "Error" da última camada (Error[4][i])

for(i=0;i<9;i++){
Error3[i] = Error4[i]*(p[i]*erro + q[i]*Var_erro + r[i]);
}

//Na etapa 10, são calculadas as derivadas dos erros para os neurônios da camada "2", os quais levarão o "Error" da última camada (Error[3][i])
//OBS: "Error[i][j]" refere-se à derivada do erro da camada "i" do neurônio "j"
//Obs2: antes é calculado o somatório total dos pesos normalizados, a fim de uso posterior da mesma

for(i=0;i<9;i++){
WnT = WnT + Wn[i];//Somatório dos pesos das regras normalizados
}

//Agora é feito parte do cálculo da derivada do erro da camada 2 (é necessário ler a dissertação do cara pra entender essa parte)
//Inicializando inicialmente os valores do vetor pra entrar no for


for(i=0;i<9;i++)
{
if(i != 0)
Resto[0] = Resto[0] + Error3[i] * Wn[i]/(WnT*WnT);

if(i != 1)
```

```
      Resto[1] = Resto[1] + Error3[i] * Wn[i]/(WnT*WnT);

      if(i != 2)
      Resto[2] = Resto[2] + Error3[i] * Wn[i]/(WnT*WnT);

      if(i != 3)
      Resto[3] = Resto[3] + Error3[i] * Wn[i]/(WnT*WnT);

      if(i != 4)
      Resto[4] = Resto[4] + Error3[i] * Wn[i]/(WnT*WnT);

      if(i != 5)
      Resto[5] = Resto[5] + Error3[i] * Wn[i]/(WnT*WnT);

      if(i != 6)
      Resto[6] = Resto[6] + Error3[i] * Wn[i]/(WnT*WnT);

      if(i != 7)
      Resto[7] = Resto[7] + Error3[i] * Wn[i]/(WnT*WnT);

      if(i != 8)
      Resto[8] = Resto[8] + Error3[i] * Wn[i]/(WnT*WnT);
      }

      //Agora sim definindo finalmente a derivada do erro de cada neurônio
da camada 2, que leva o erro da camada 3

      for(i=0;i<9;i++){
      Error2[i] = Error3[i]*((WnT-Wn[i])/(WnT*WnT)) - Resto[i];
      }

      //Na etapa 11, são calculados os erros para os neurônios da camada
"1"(um total de 6 neurônios), os quais levarão o "Error" da última camada

      Error1[0] = Error2[0]*Wx2[0] + Error2[1]*Wx2[1] + Error2[2]*Wx2[2];
      Error1[1] = Error2[3]*Wx2[0] + Error2[4]*Wx2[1] + Error2[5]*Wx2[2];
      Error1[2] = Error2[6]*Wx2[0] + Error2[7]*Wx2[1] + Error2[8]*Wx2[2];
      Error1[3] = Error2[0]*Wx1[0] + Error2[3]*Wx1[1] + Error2[6]*Wx1[2];
```

```
Error1[4] = Error2[1]*Wx1[0] + Error2[4]*Wx1[1] + Error2[7]*Wx1[2];
Error1[5] = Error2[2]*Wx1[0] + Error2[5]*Wx1[1] + Error2[8]*Wx1[2];

//Na etapa 12, cálculo das derivadas das funções dos neurônios em relação aos parâmetros de ajuste, para posterior uso
//OBS: as camadas que necessitam ajustes são as camadas "4" e "1", devido as mesmas possuírem funções de pertinência!

//Derivada das funções da camada 4 (polinômios de Sugeno), sendo 27 no total

for(i=0;i<9;i++){
DER_POL_p[i] = erro*Wn[i];//derivada parcial do polinômio de Sugeno em relação ao parâmetro "p"
DER_POL_q[i] = Var_erro*Wn[i];//derivada parcial do polinômio de Sugeno em relação ao parâmetro "q"
DER_POL_r[i] = Wn[i];//derivada parcial do polinômio de Sugeno em relação ao parâmetro "r"
}

//Derivada das funções da camada 1, sendo um total de 18
//OBS: aqui "a[i]", "b[i]" e "c[i]" representam os pontos inicial, médio e final(respectivamente) de cada triângulo (num total de 3 pra cada variável)
//Interessante se ressaltar que os "if" separam as regiões do triângulo (reta crescente e decrescente)

for(i=0;i<3;i++){
if(a1[i] < erro < b1[i])
DER_TRIe_a1[i] = erro/((b1[i]-a1[i])*(b1[i]-a1[i]));//derivada parcial do triângulo do erro em relação ao parâmetro "a"
else DER_TRIe_a1[i] = 0;

if(a1[i] < erro < b1[i])
DER_TRIe_b1[i] = -1*(erro)/((b1[i]-a1[i])*(b1[i]-a1[i]));//derivada parcial do triângulo do erro em relação ao parâmetro "b"
else DER_TRIe_b1[i] = -erro/((c1[i]-b1[i])*(c1[i]-b1[i]));
```

```
      if(b1[i] < erro < c1[i])
      DER_TRIe_c1[i]   =   erro/((c1[i]-b1[i])*(c1[i]-b1[i]));//derivada
parcial do triângulo do erro em relação ao parâmetro "c"
      else DER_TRIe_c1[i] = 0;

      if(a2[i] < Var_erro < b2[i])
      DER_TRIV_a2[i] = Var_erro/((b2[i]-a2[i])*(b2[i]-a2[i]));//derivada
parcial do triângulo da variação do erro em relação ao parâmetro "a"
      else DER_TRIV_a2[i] = 0;

      if(a2[i] < Var_erro < b2[i])
      DER_TRIV_b2[i]          =          -Var_erro/((b2[i]-a2[i])*(b2[i]-
a2[i]));//derivada parcial do triângulo da variação do erro em relação ao
parâmetro "b"
      else DER_TRIV_b2[i] = -Var_erro/((c2[i]-b2[i])*(c2[i]-b2[i]));

      if(b2[i] < Var_erro < c2[i])
      DER_TRIV_c2[i] = Var_erro/((c2[i]-b2[i])*(c2[i]-b2[i]));//derivada
parcial do triângulo da variação do erro em relação ao parâmetro "c"
      else DER_TRIV_c2[i] = 0;
      }

      //Na etapa 13, cálculo do coeficiente de aprendizado, totalmente
baseado na dissertação do cara

      erro_aux1 = 0;
      erro_aux2 = (ref - saida)*(ref - saida);
      if(erro_aux2 > 1,04 * erro_aux1) n = n * 0,7;
      else if (erro_aux2 <= erro_aux1) n = n * 1,5;
      else erro_aux2 = erro_aux1;

      //Na etapa 14, cálculo da variação dos parâmetros
      //Ao todo serão calculadas 6x3 + 9x3 = 45 variações de parâmetros
(Para as funções triangulares de entrada (total de 6): "a", "b", "c";
para o polinômio de saída(9): "p", "q","r")
      //Para o cálculo serão utilizados as derivadas dos erros e dos
parâmetros anteriormente calculadas
```

```
      //O SOMATÓRIO DA VARIAÇÃO DOS PARÂMETROS VAI SENDO REALIZADO A CADA
PONTO DE TREINAMENTO

      for(i=0;i<3;i++){

      //Cálculo da variação dos parâmetros dos triângulos da variável erro
e variação do erro

      VAR_a1[i] = -1*n * Error1[i] * DER_TRIe_a1[i];
      VAR_b1[i] = -1*n * Error1[i] * DER_TRIe_b1[i];
      VAR_c1[i] = -1*n * Error1[i] * DER_TRIe_c1[i];
      VAR_a2[i] = -1*n * Error1[i+3] * DER_TRIV_a2[i];
      VAR_b2[i] = -1*n * Error1[i+3] * DER_TRIV_b2[i];
      VAR_c2[i] = -1*n * Error1[i+3] * DER_TRIV_c2[i];

      //Somatório das variações dos parâmetros das funções de pertinência
das entradas

      Soma_VAR_a1[i] = Soma_VAR_a1[i] + VAR_a1[i];
      Soma_VAR_b1[i] = Soma_VAR_b1[i] + VAR_b1[i];
      Soma_VAR_c1[i] = Soma_VAR_c1[i] + VAR_c1[i];
      Soma_VAR_a2[i] = Soma_VAR_a2[i] + VAR_a2[i];
      Soma_VAR_b2[i] = Soma_VAR_b2[i] + VAR_b2[i];
      Soma_VAR_c2[i] = Soma_VAR_c2[i] + VAR_c2[i];
      }

      for(i=0;i<9;i++){

      //Cálculo da variação dos parâmetro dos polinômios de Sugeno

      VAR_p[i] = -1*n * Error4[i] * DER_POL_p[i];
      VAR_q[i] = -1*n * Error4[i] * DER_POL_q[i];
      VAR_r[i] = -1*n * Error4[i] * DER_POL_r[i];

      //Somatório das variações dos parâmetros dos polinômios de Sugeno

      Soma_VAR_p[i] = Soma_VAR_p[i] + VAR_p[i];
      Soma_VAR_q[i] = Soma_VAR_q[i] + VAR_q[i];
```

```
Soma_VAR_r[i] = Soma_VAR_r[i] + VAR_r[i];
}

z = z + 1;

//FIM DO "DO WHILE 1" == FIM DOS PONTOS DE EXEMPLO == FINAL DA ÉPOCA X DO TREINAMENTO

}
while(z<9); // z equivale ao número de pares exemplos já apresentados à rede

//Na etapa 15, atualização dos parâmetros ==>ISSO É INTELIGÊNCIA/APRENDIZAGEM!!!!!!!!!!!!!!!!

//Atualização dos parâmetros dos polinômios de Sugeno

for(i=0;i<9;i++){
p[i] = p[i] + Soma_VAR_p[i];
q[i] = q[i] + Soma_VAR_q[i];
r[i] = r[i] + Soma_VAR_r[i];
}

//Atualização dos parâmetros das funções triangulares

for(i=0;i<3;i++){
a1[i] = a1[i] + Soma_VAR_a1[i];
b1[i] = b1[i] + Soma_VAR_b1[i];
c1[i] = c1[i] + Soma_VAR_c1[i];
a2[i] = a2[i] + Soma_VAR_a2[i];
b2[i] = b2[i] + Soma_VAR_b2[i];
c2[i] = c2[i] + Soma_VAR_c2[i];
}

//FIM DO "DO WHILE 2" == Finda-se o treinamento se o erro está dentro do limite/faixa aceitável

//}
```

```
//while(erro <= desejado);//IMPORTANTE: DESCOMENTAR ESTA LINHA PRO CÓDIGO FUNCIONAR

//Printagem dos valores dos parâmetros referentes ao resultado do treinamento

for(i=0;i<3;i++){

printf("a1[%d] = %f \n", i, a1[i]);
printf("b1[%d] = %f \n", i, b1[i]);
printf("c1[%d] = %f \n", i, c1[i]);
printf("a2[%d] = %f \n", i, a2[i]);
printf("b2[%d] = %f \n", i, b2[i]);
printf("c2[%d] = %f \n", i, c2[i]);
}

for(i=0;i<9;i++){
printf("p[%d] = %f \n", i, p[i]);
printf("q[%d] = %f \n", i, q[i]);
printf("r[%d] = %f \n", i, r[i]);
}

}
```