6 CONTROLE INTELIGENTE

6.1 CONTROLE *NEURO FUZZY*

A estrutura NFN (*Neo Fuzzy Neuron*), primeiramente proposta por (YAMAKAWA, 1992), possui características de transferências sinápticas não-lineares onde a sinapse é realizada por um conjunto de regras *fuzzy* do tipo Sugeno de ordem zero (Figura 64). As funções de pertinência são construídas contemplando toda a faixa das variáveis de entrada correspondentes. As funções de pertinência triangulares são construídas de forma a serem complementares em relação às suas vizinhas. Com isto, para qualquer entrada ativa-se no máximo duas funções de pertinência e a soma destas duas funções é sempre igual a um (equação 23 e equação 24). Portanto, em função desta característica a etapa de defuzzificação baseada no método do centroide não precisa do termo de divisão. Alguns trabalhos apresentam variações desta metodologia, como por exemplo, estruturas com funções de pertinências diferenciadas por região (CHATURVEDI, 2008)

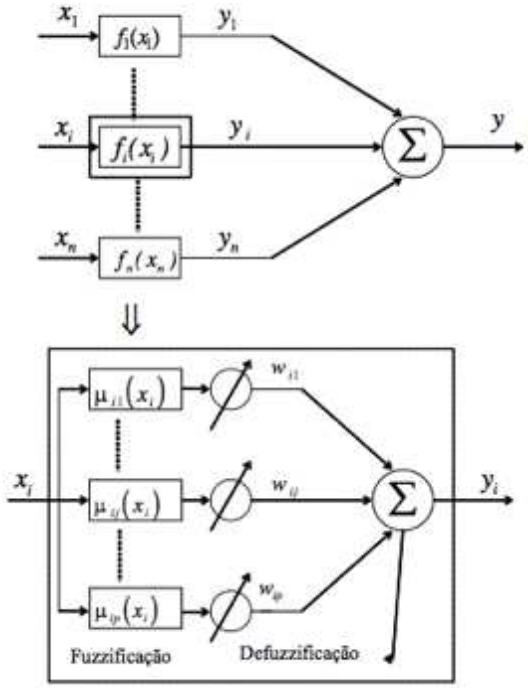


Figura 64 – Representação de uma estrutura NFN.

$$\mu_{\mathrm{ij}} + \mu_{\mathrm{ik}_i+1} = 1 \tag{23}$$

$$f_i(x_i) = \frac{\mu_{\mathrm{ikj}}(\mathrm{x}_i)\mathrm{w}_{ik_i} + \mu_{\mathrm{ik}_i+1}(x_i)w_{ik_i+1}}{\mu_{\mathrm{ikj}}(\mathrm{x}_i) + \mu_{\mathrm{ik}_i+1}(x_i)} = \mu_{\mathrm{ikj}}(\mathrm{x}_i)\mathrm{w}_{ik_i} + \mu_{\mathrm{ik}_i+1}(x_i)w_{ik_i+1} \tag{24}$$

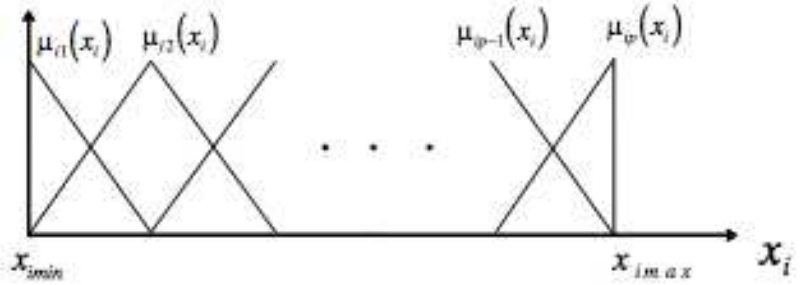


Figura 65 – Representação das funções de pertinência de uma estrutura NFN (CARVALHO, 2010).

No processo de aprendizagem utiliza-se a técnica de retropropagação do erro. Observa-se que apenas as funções ativas são relevantes durante este processo, segundo o método de retropropagação. Assim, só um ou dois pesos correspondentes aos ramos ativos são ajustados durante um passo de treinamento do NFN. Devido a este mecanismo, uma rede NFN exibe um tempo de treinamento muito menor do que uma rede convencional, além de uma simplicidade muito maior (GOUVÊA, 2005). Os ajustes dos pesos são dados pelo método de descida do gradiente, tal que:

$$\mathrm{w}_{\mathrm{ik}_i}^{j+1} = \mathrm{w}_{\mathrm{ik}_i}^{j} - \alpha^j \frac{\partial E_t\left(\mathrm{w}_{\mathrm{ik}_i}^{j}\right)}{\partial(\mathrm{w}_{\mathrm{ik}_i}^{j})} \tag{25}$$

resultando em:

$$\mathrm{w}_{\mathrm{ik}_i}^{j+1} = \mathrm{w}_{\mathrm{ik}_i}^{j} - \alpha^j(\mathrm{y}_t^j - \mathrm{y}_d^j)\mu_{ik_i}(x_{ti}) \tag{26}$$

O termo α^j está relacionado ao tamanho do passo de otimização e pode ser determinado empiricamente ou por algum método de busca.

Gouvêa (2005) propõe um controlador *neuro-fuzzy* de estrutura simples denominado ONFC (*On-line Neuro-fuzzy Controller*). Este controlador de baixa complexidade computacional, baseado em apenas duas regras, apresenta

características adaptativas e já foi implementado com sucesso em aplicações como, por exemplo, no controle de um motor de indução (GOUVÊA, 2005) e num sistema de controle de pH (CARVALHO, 2006).

O controlador ONFC também foi estudado por Pires (PIRES, 2007). Em seu trabalho Pires faz uma análise matemática do controlador, apresentando suas principais características e seus componentes. Carvalho (2010) apresenta o desenvolvimento e a análise de um novo algoritmo de aprendizagem, abordando aspectos relativos à estabilidade, aplicação e critérios para sintonia.

O presente trabalho tem como questão principal a implementação, desenvolvimento e avaliação de desempenho de um controlador adaptativo baseado em sistemas *neuro-fuzzy,* aplicado ao robô auto-equilibrante. Além disto, busca-se um aprofundamento do entendimento do controlador ONFC com uma avaliação da condição de estabilidade. É proposto um método de ajuste dos parâmetros do controlador, bem como o desenvolvimento de técnicas capazes de corrigir limitações observadas em versões anteriores, como a divergência dos pesos das funções de pertinência. Propõe-se então uma otimização dos pesos baseada não somente no erro total do sistema, mas também no erro associado ao aumento desta diferença. A técnica que está sendo usada para corrigir a divergência dos pesos é chamada de regularização e foi proposta em (SPADINI, 2013). Portanto, o controlador ONFC proposto será aplicado em uma planta com alta resposta dinâmica, característica de um robô auto-equilibrante de duas rodas.

No entanto, em geral, o controle ONFC é aplicado para plantas de dinâmica lenta como: o controle vetorial de motor de indução (GOUVÊA, 2005), o controle de pH em processos de neutralização (CARVALHO, 2006), controle da velocidade de um motor CC (PIRES, 2007), o controle de temperatura de uma região da torre fracionadora principal de uma unidade industrial de Coqueamento Retardado (CARVALHO, 2010), e o controle de níveis em planta de tanques acoplados (SPADINI, 2013).

Buscando a implementação de um algoritmo para controle on-line, Spadini (2013) apresenta em seu trabalho o algoritmo ONFC. Baseado na estrutura NFN, o ONFC apresenta uma estrutura bastante simplificada com apenas três funções de pertinência complementares. Sua estrutura simples apresenta um custo computacional baixo, da mesma ordem de grandeza de um controlador PID.

As características citadas são importantes para a aplicação *on-line* deste controlador. O controlador apresenta apenas três funções de pertinência triangulares e complementares. O erro entre o valor desejado e o valor atual é utilizado tanto para determinação da ação de controle, quanto para correção dos seus pesos (parâmetros livres). A figura 66 apresenta um diagrama de blocos de um sistema genérico usando o controlador ONFC. A Figura 67 apresenta a estrutura interna do ONFC.

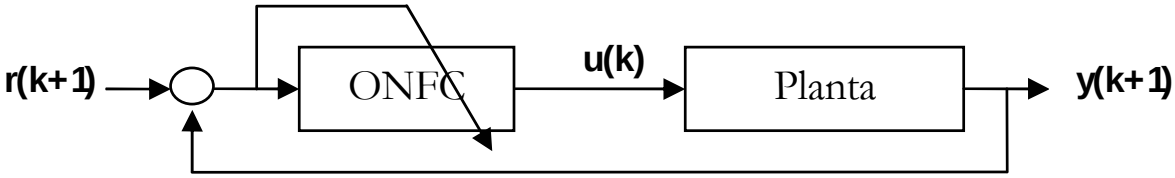


Figura 66 – Diagrama de blocos do controle com o ONFC.

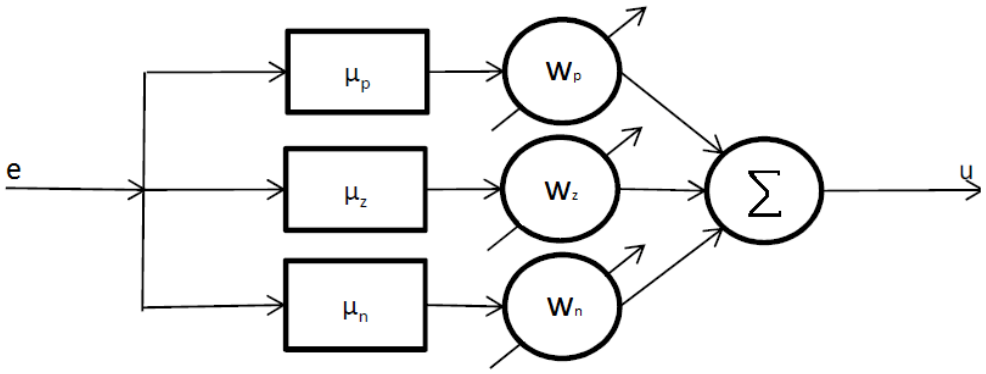


Figura 67 – Estrutura do controlador ONFC com 3 pesos.

O processamento dos sinais pode ser separado em duas etapas: determinação da ação de controle e ajuste dos pesos do controlador.

6.1.1 Determinação da ação de controle

Na etapa de determinação da ação de controle, o sinal do erro é usado para determinar os valores das três funções de pertinência. A estrutura do controlador é definida com três funções de pertinência complementares entre si. Para o universo do discurso, no máximo três funções estarão ativas, tal que:

$$\mu_p + \mu_z + \mu_n = 1 \tag{27}$$

As funções de pertinências positiva μ_p, zero μ_z e negativa μ_n estão representadas na figura 68.

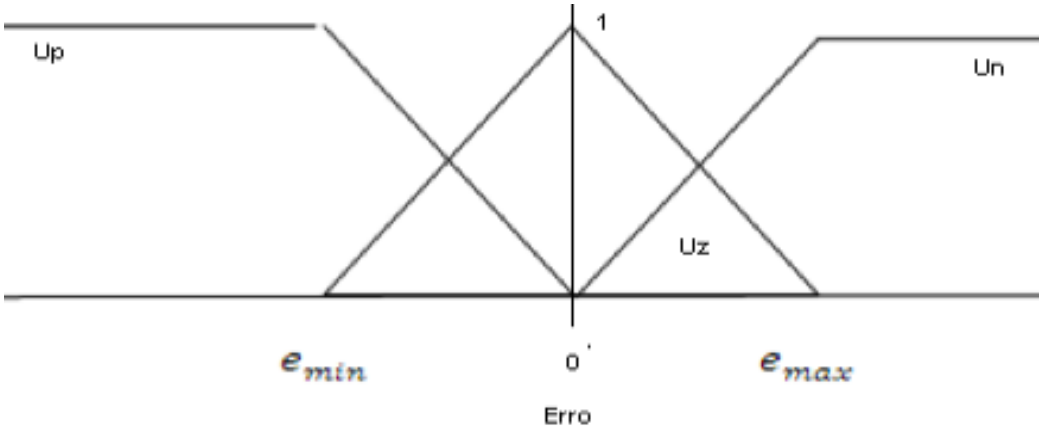


Figura 68 – Funções de pertinência complementares usadas no controlador ONFC.

Os cálculos das funções de pertinências μ_p, μ_z e μ_n são determinados segundo as equações 22, 23 e 24:

$$\mu_p = \begin{cases} \dfrac{-e}{EM} & ; \quad e_{min} < e < 0 \\ 0 & ; \quad e \geq 0 \\ 1 & ; \quad e \leq e_{min} \end{cases} \tag{28}$$

$$\mu_n = \begin{cases} \dfrac{e}{EM} & ; \quad 0 < e < e_{max} \\ 1 & ; \quad e \geq e_{max} \\ 0 & ; \quad e \leq 0 \end{cases} \tag{29}$$

$$\mu_z = \begin{cases} \dfrac{-e}{EM} + 1 & ; \quad 0 < e < e_{max} \\ \dfrac{e}{EM} + 1 & ; \quad e_{min} < e < 0 \\ 0 & ; \quad e \geq e_{max} \\ 0 & ; \quad e \leq e_{min} \end{cases} \tag{30}$$

Onde,

$$EM = \frac{\mathrm{e}_{max} - e_{min}}{2} \tag{31}$$

A entrada e pode ser interpretada com $e(k)$, que significa o erro no instante k.

$$e = y_d - y \tag{32}$$

Os valores de e_{min} e e_{max} são definidos em função do processo. Diferentemente do caso do NFN, estes parâmetros não representam necessariamente todo o universo possível de excursão do erro (e). Estes parâmetros são ajustáveis e normalmente são configurados de forma a representar uma faixa viável de excursão do erro. A configuração pode ser feita com base em estatísticas, usando o desvio padrão da variável erro σ_{erro}. Sugere-se um ajuste conforme apresentado na equação 33.

$$EM = 3\sigma_{erro} \tag{33}$$

É importante salientar que mesmo fora dos limites e_{min} e e_{max}as funções continuam válidas, apresentando valores um ou zero.

A saída do controlador u é definida segundo um modelo Sugeno de ordem zero. Usa-se a regra do centróide para definição do valor. Como as funções de pertinências são complementares, o cálculo é reduzido para:

$$\mathrm{u(n)} = \mu_{\mathrm{p}}(\mathrm{n})\mathrm{w_p(n)} + \mu_{\mathrm{z}}(\mathrm{n})\mathrm{w_z(n)} + \mu_{\mathrm{n}}(\mathrm{n})\mathrm{w_n(n)} \tag{34}$$

6.1.2 Ajuste de pesos

O ajuste de pesos é feito pela descida do gradiente. Para isso, foi definida uma função de custo que é dada pelo erro médio quadrático.

$$\mathrm{J} = \frac{1}{2}e^2 \tag{35}$$

Derivando a equação anterior em relação aos pesos teremos, respectivamente:

$$\frac{\partial \mathrm{J}}{\partial w_p} = \frac{\partial \mathrm{J}}{\partial e}\frac{\partial \mathrm{e}}{\partial y}\frac{\partial \mathrm{y}}{\partial u}\frac{\partial \mathrm{u}}{\partial w_p} \tag{36}$$

$$\frac{\partial \mathrm{J}}{\partial w_z} = \frac{\partial \mathrm{J}}{\partial e}\frac{\partial \mathrm{e}}{\partial y}\frac{\partial \mathrm{y}}{\partial u}\frac{\partial \mathrm{u}}{\partial w_z} \tag{37}$$

$$\frac{\partial \mathrm{J}}{\partial w_n} = \frac{\partial \mathrm{J}}{\partial e}\frac{\partial \mathrm{e}}{\partial y}\frac{\partial \mathrm{y}}{\partial u}\frac{\partial \mathrm{u}}{\partial w_n} \tag{38}$$

Desenvolvendo as relações apresentadas, chega-se ao cálculo do gradiente da função custo. O desenvolvimento para o segundo e terceiro termo será omitido por ser análogo.

$$\frac{\partial \mathrm{J}}{\partial e} = 2\frac{1}{2}(e) = e \tag{39}$$

$$\frac{\partial \mathrm{e}}{\partial y} = \frac{\partial(\mathrm{y}_d - y)}{\partial y} = -1 \tag{40}$$

$$\frac{\partial \mathrm{u}}{\partial w_p} = \frac{\partial(\mu_p w_p + \mu_z w_z + \mu_n w_n)}{\partial w_p} = \mu_p \tag{41}$$

A derivada $\frac{\partial \mathrm{y}}{\partial u}$ representa a variação da saída da planta em relação à entrada. No trabalho originalmente proposto por Gouvêa (2005), esta derivada é incorporada à taxa de aprendizagem, determinando o sinal da mesma equação (42). Caso a planta apresente ganho estático positivo em relação à variável de saída do controlador, este termo será positivo. Caso contrário, a variável de determinação da taxa de amostragem deverá ter sinal negativo.

$$\alpha = \alpha_0 \frac{\partial y}{\partial u} \tag{42}$$

Obtém-se, ao final, o gradiente da função objetivo que se deseja minimizar. O ajuste dos pesos é então realizado no sentido oposto ao gradiente, resultando na equação 43.

$$\mathrm{w}_p(n+1) = w_p(n) - \alpha\mu_p(-1)e \tag{43}$$

Ou simplesmente:

$$\mathrm{w}_p(n+1) = w_p(n) + \alpha\mu_p e \tag{44}$$

Assim, chega-se a equação (45), que é utilizada para correção dos pesos do ONFC. Esta equação em conjunto com as equações (39), (40), (41) e (43) formam a base para a implementação prática do ONFC.

$$\mathrm{w}_I(n+1) = w_I(n) + \alpha\mu_I e(n) \tag{45}$$

Deve-se salientar que além dos parâmetros livres que são automaticamente corrigidos, apenas a taxa de aprendizagem e os limites do universo de discurso da entrada devem ser ajustados previamente.

6.1.3 Análise da convergência do ONFC

Pode-se perceber que o processo de convergência do ONFC está diretamente ligado ao erro do sistema de controle, ou seja, a convergência se dará quando o sistema atingir o erro nulo. Assumindo-se que a taxa de aprendizado α seja tal que não provoque instabilidade no sistema, a prova de convergência pode ser obtida considerando que nesta condição não há necessidade de correção dos pesos da rede. Assim, a equação (45) se torna:

$$\alpha\mu_p e(n) = w_p(n) - w_p(n-1) = \Delta w_P = 0 \tag{46}$$

$$\alpha\mu_n e(n) = w_n(n) - w_n(n-1) = \Delta w_n = 0 \tag{47}$$

$$\alpha\mu_z e(n) = w_z(n) - w_z(n-1) = \Delta w_z = 0 \tag{48}$$

Considerando-se que a função de pertinência μ_z da equação (30) trata-se de uma função simétrica em relação ao erro zero, e considerando que se o erro for limitado com média zero, o peso w_z se manterá também limitado. Portanto, resta apenas demonstrar a convergência dos demais pesos.

Reescrevendo as equações (46) e (47) em função das equações (28) e (29) e desprezando-se os índices relativos às amostragens, obtém-se:

$$\Delta w_p = \alpha\left[\frac{-e^2}{\Delta_{lim}} + \frac{e}{2}\right] \Rightarrow e(-2e + \Delta_{lim}) = 0 \Rightarrow \begin{cases} e = 0 \\ e = \dfrac{\Delta_{lim}}{2} \end{cases} \tag{49}$$

$$\Delta w_n = \alpha\left[\frac{e^2}{\Delta_{lim}} + \frac{e}{2}\right] \Rightarrow e(-2e + \Delta_{lim}) = 0 \Rightarrow \begin{cases} e = 0 \\ e = \dfrac{-\Delta_{lim}}{2} \end{cases} \tag{50}$$

Tal que,

$$\Delta_{lim} = e_{max} - e_{min} = 2e_{max} = |2e_{min}| \tag{51}$$

Analisando-se soluções obtidas para as equações (43) e (44), conclui-se que a solução que satisfaz simultaneamente as equações (40) e (41), corresponde a e=0, ou seja, erro nulo.

Portanto, baseado na tendência do controlador de atingir um erro nulo verifica-se que o ajuste dos pesos sofre uma interrupção e a ação de controle será igual a 0.

6.1.4 Características de desempenho

O ONFC apresenta características bastante interessantes de desempenho. Em suas aplicações recentes, observa-se bom desempenho e capacidade

adaptativa. Talvez a limitação mais visível seja a tendência de aumento constante da diferença entre os pesos das duas funções de pertinência.

Esta tendência é evidenciada pelo algoritmo de aprendizagem. Observando-se a equação (45) de ajuste dos pesos, nota-se que a correção é diretamente proporcional ao erro e à pertinência. A variação da diferença entre os pesos é definida então como:

$$\Delta w(n) = \Delta w_p(n) - \Delta w_n(n) \tag{52}$$

Lembrando que:

$$\Delta w(n) = \alpha \mu_p e - \alpha \mu_n e \tag{53}$$

Vem que,

$$\Delta w(n) = \alpha e\left(\mu_p - \mu_n\right) \tag{54}$$

Chega-se à equação que determina a evolução de $\Delta w(n)$ como:

$$\Delta w(n) = \begin{cases} \alpha |e|\left(\mu_p - \mu_n\right), & e > 0; \\ \alpha |e|\left(\mu_n - \mu_p\right), & e < 0; \end{cases} \tag{55}$$

Pelas equações 22 e 23 sabe-se que:

$$\begin{cases} \left(\mu_p - \mu_n\right) > 0 \quad se \;\; e > 0; \\ \left(\mu_n - \mu_p\right) > 0 \quad se \;\; e < 0; \end{cases} \tag{56}$$

Desta forma, fica evidente que a diferença entre os pesos é sempre crescente. Este aumento constante da diferença leva a um problema de estabilidade. À medida que este termo aumenta, o controlador tende a se aproximar de uma região instável. Pequenas variações na variável de processo levam a ações bruscas na variável manipulada. Nesta condição é necessária uma ação de ajuste dos pesos. Como sempre ocorrerá um aumento da diferença

dos pesos, chega-se à conclusão de que é necessário um processo de redução desta diferença. Gouvêa (2005) propõe um ajuste baseado na ponderação dos pesos conforme a equação (57).

$$w_p(n) = \frac{u - w_n(n)(1 - \mu_p)}{\mu_p} \tag{57}$$

Esta solução, aparentemente simples e trivial, apresenta dois problemas crônicos. O primeiro é a falta de um critério para definir o momento adequado para efetuar a correção. Esta condição é singular para cada planta. Para sistemas variantes no tempo ou não-lineares esta condição pode, inclusive, ser dependente do ponto de operação do sistema ou condição da planta. Isto torna difícil a definição exata do momento adequado para efetuar a correção.

O segundo problema notado está no próprio método de ajuste, uma vez que o termo μ_p pode apresentar valores entre 0 e 1. Quando o termo de pertinência de um dos pesos tende a zero, a função apresenta uma instabilidade numérica (equação 58).

$$\lim_{\mu_i \to 0} w_i(n) = \lim_{\mu_i \to 0} \frac{u - w_i(n)(1 - \mu_i)}{\mu_i} = \infty \tag{58}$$

Esta condição seria bastante crítica, levando o algoritmo a um erro numérico com consequências sérias. Gouvêa (2005) propõe, como condição para efetuar o ajuste, que $\mu_i \neq 0$.

Mesmo quando $\mu_i > 0$, pode-se ter problemas caso μ_i seja pequeno. O novo valor calculado para $w_i(n)$pode ser inconsistente (por exemplo: $w_1 > w_2$ em uma planta de resposta direta). Esta condição poderia inverter a ação de controle na próxima iteração, o que também poderia ser tratado com a inclusão de mais uma lógica de segurança.

Uma solução encontrada foi realizar uma reinicialização dos pesos (CARVALHO, 2006). Baseando-se no conhecimento da planta é definido um valor limite para os pesos w1 e w2. Quando este limite é atingido, os dois pesos são reiniciados com o valor da saída do controlador.

$$se\ w_i\ ultrapassar\ limite\ \rightarrow \begin{cases} w_p = u(n), \\ w_n = u(n). \end{cases} \quad (59)$$

Esta condição evita os problemas de ordem numérica apresentados anteriormente ou a necessidade de se implementar muitas lógicas de segurança, mas não soluciona a questão da influência negativa de se perder a informação de aprendizagem exatamente no momento em que o erro é significativo. Mesmo se o ajuste não ocorrer durante um transiente forte, as pequenas perturbações levarão a um aumento gradativo da diferença entre os pesos e em algum momento será necessário o ajuste. Nesta condição depara-se com outro problema, pois após o reinício dos pesos, o algoritmo terá que recomeçar o processo de aprendizagem.

Observa-se que a variação da diferença dos pesos é diretamente proporcional ao módulo do erro. Desta forma, se for usada a diferença entre os pesos para definir o momento de se realizar o ajuste do Δw, existe uma grande probabilidade disto ocorrer exatamente em um momento onde existe um erro significativo, aumentando ainda mais o risco envolvido.

6.1.5 Regularização dos pesos

Como mencionado anteriormente, a função de pertinência μ_z é simétrica em relação ao erro zero. Se o erro for limitado e com média zero como esperado, o peso w_z se manterá também limitado naturalmente através da função custo. O mesmo não acontece com os pesos w_ne w_p, que poderão se tornar ilimitados para um sinal de ruído presente no erro ou mesmo para variações intermitentes de referência. Em simulações e experimentos em laboratório este fato não acarreta problemas, pois raramente são feitos testes longos o suficiente para que os valores das variáveis de ponderação aumentarem a ponto de chegarem ao limite computacional do hardware em que o controlador está implementado, deixando o fato menos evidente. No problema descrito por Carvalho (2010), em aplicações reais, especialmente as que requeiram uma mudança constante em sua referência, a limitação torna-se notória.

Através de uma nova função de custo, na qual aparece uma nova parcela composta por w_ne w_p, busca-se não somente a minimização do erro, mas também do comportamento dos pesos:

$$\mathrm{J} = \frac{\mathrm{e(n)}^2 + \gamma \mathrm{w_p(n)}^2 - \gamma \mathrm{w_n(n)}^2}{2} \tag{60}$$

$$\mathrm{J} = \frac{\mathrm{e(n)}^2 + \gamma \mathrm{w_p(n)}^2 - \gamma \mathrm{w_n(n)}^2}{2} \tag{61}$$

Ao observar a expressão (54) nota-se que $w_z$não foi incluída explicitamente nessa função. Para realizar a derivada da nova função custo, segue-se o procedimento utilizando derivadas parciais como já realizado anteriormente, porém ocorre o surgimento de uma nova parcela ao final da expressão de ajuste dos pesos.

Resolvendo para peso $\mathrm{w_p}$,

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w_p}} = \frac{1}{2}\left\{\frac{\partial \mathrm{e(n)}^2}{\partial \mathrm{w_p}} + \frac{\partial\left(\gamma' \mathrm{w_p}(n)^2\right)}{\partial \mathrm{w_p}} - \frac{\partial(\gamma' \mathrm{w_n}(n)^2)}{\partial \mathrm{w_p}}\right\}$$

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w_p}} = \frac{1}{2}\left\{\frac{\partial \mathrm{e(n)}^2}{\partial e}\frac{\partial \mathrm{e}}{\partial y}\frac{\partial \mathrm{y}}{\partial u}\frac{\partial \mathrm{u}}{\partial w_p} + 2\gamma' \mathrm{w_p(n-1)} + 0\right\} \tag{62}$$

sabendo que ,

$$\frac{\partial \mathrm{e}}{\partial y} = \frac{\partial(\mathrm{y}_d - y)}{\partial y} = -1 \tag{63}$$

$$\frac{\partial \mathrm{u}}{\partial w_p} = \frac{\partial(\mu_p w_p + \mu_z w_z + \mu_n w_n)}{\partial w_p} = \mu_p \tag{64}$$

Substituindo as equações acima (63) e (64) em (62):

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w_p}} = -\mathrm{e(n)}\frac{\partial \mathrm{y}}{\partial u}\mu_p(\mathrm{n}) + \gamma' \mathrm{w_p(n-1)} \tag{65}$$

A derivada $\frac{\partial \mathrm{y}}{\partial \mathrm{u}}$ é dependente da planta e desconhecida. Portanto chamaremos esse termo de k.

$$\frac{\partial \mathrm{y}}{\partial u} = k \tag{66}$$

Portanto,

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w_p}} = -\mathrm{ke(n)}\mu_p(\mathrm{n}) + \gamma' \mathrm{w_p(n-1)} \tag{67}$$

Analogamente,

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w_n}} = -\mathrm{ke(n)}\mu_n(\mathrm{n}) - \gamma' \mathrm{w_n(n-1)} \tag{68}$$

Recalculando os pesos:

$$\mathrm{w_p\,(n) = w_p(n-1)} + \alpha\frac{\partial \mathrm{J}}{\partial \mathrm{w_p}}$$

$$\mathrm{w_n\,(n) = w_n(n-1)} - \alpha\frac{\partial \mathrm{J}}{\partial \mathrm{w_n}}$$

$$\begin{cases} \mathrm{w_p\,(n) = w_p(n-1) - \alpha ke(n)}\mu_p(\mathrm{n}) + \alpha\gamma' \mathrm{w_p(n-1)} \\ \mathrm{w_n\,(n) = w_n(n-1) - \alpha ke(n)}\mu_n(\mathrm{n}) - \alpha\gamma' \mathrm{w_n(n-1)} \end{cases} \tag{69}$$

Por simplificação consideraremos $\alpha\gamma' = \gamma$, portanto:

$$\begin{cases} \mathrm{w_p\,(n) = w_p(n-1) - \alpha ke(n)}\mu_p(\mathrm{n}) + \gamma \mathrm{w_p(n-1)} \\ \mathrm{w_n\,(n) = w_n(n-1) - \alpha ke(n)}\mu_n(\mathrm{n}) - \gamma \mathrm{w_n(n-1)} \end{cases}$$

O novo fator γ, compreendido entre zero e um, irá determinar a taxa de decaimento dos pesos e deverá ser ajustado de acordo com a planta utilizada. Devido ao fato de α ser escolhido muito maior do que γ , quando um distúrbio ou uma nova referência atuar sobre o sistema, os valores dos pesos alterarão para compensar o erro e posteriormente w_p e w_n irão gradualmente diminuir. A redução desses valores é possível pelo segundo ramo do controlador que irá compensar na saída o comportamento desses pesos no momento em que for alcançado o regime permanente.

A partir desta nova estratégia de ajuste e otimização de erro pode-se ter um controlador que apresenta uma relação de compromisso entre a necessidade de ajuste dos pesos de suas funções de pertinência e a necessidade de se reduzir a distância entre os pesos.

6.1.6 Avaliação da influência dos parâmetros α e γ no desempenho do sistema

Um dos parâmetros a serem configurados do ONFC é o coeficiente α de ajustes dos pesos w_p , w_n e w_z . Conforme apresentado em Gouvêa (GOUVEA, 2005), este termo agrupa o passo do método do gradiente e a relação entre a variável manipulada e a saída da planta.

Em função de sua importância, o ajuste deste termo é crítico para a estabilidade e desempenho do controlador. O ajuste dos pesos utiliza como base o método da descida do gradiente. Como discutido em seções anteriores, o *α* incorpora tanto a dinâmica da planta, quanto o tamanho do passo do método do gradiente. Neste contexto, verifica-se que a escolha adequada do α está diretamente relacionada com a estabilidade do controlador.

Esta avaliação pode ser feita segundo duas óticas. Observando-se o desenvolvimento sugerido por Pires (2007) pode-se relacionar o termo diretamente com a ação integral do controlador (PIRES, 2007). Este termo seria então inversamente proporcional a um suposto tempo integral. Sob esta ótica, um aumento de α seria equivalente à redução do tempo integral. O tempo integral, para plantas lineares, deve ser ajustado conforme a dinâmica da planta a ser controlada.

Guardando as devidas proporções, verifica-se que o parâmetro α deve ser configurado de forma que sua influência como parcela integradora não leve à instabilidade do sistema, devendo ser menor ou da mesma ordem de grandeza do tempo de resposta da planta. Para o caso específico do ONFC, deve-se levar em consideração o ganho da planta, uma vez que o α também guarda relação com a parte proporcional do controlador.

Outra forma de se avaliar o efeito do α é analisando o método de minimização usado na função objetivo (CARVALHO, 2010). Deseja-se a minimização do erro quadrático entre o valor de referência e o valor da variável de processo. O método utilizado é a descida do gradiente. Este método consiste em buscar a direção contrária ao do gradiente da função objetivo. A aplicação deste método pressupõe o emprego de um passo de ajuste a cada iteração. Este passo pode ser variável ou constante.

O algoritmo ONFC proposto apresenta passo constante. Desta forma, observa-se que a convergência do método pode ser prejudicada com a escolha de um passo inadequado. Caso o α apresente valores elevados pode-se prever um comportamento errático do método da descida do gradiente. Nesta condição o algoritmo pode passar do valor ótimo da função e oscilar em torno dele. Caso este valor esteja excessivamente pequeno, o algoritmo irá demorar a atingir a convergência para o valor ótimo.

O efeito do α no desempenho do controlador pode ser avaliado segundo duas óticas: efeito relacionado a um termo de integração excessivamente baixo e baixa velocidade de aumento da diferença dos pesos w_p e w_n, levando a um termo proporcional baixo. As duas situações levam a uma ação de controle lenta, atrapalhando o desempenho final.

6.1.7 Implementação do controle ONFC

Para avaliação do algoritmo proposto foi configurada uma estrutura no ambiente de simulação computacional Simulink conforme apresentando nas figuras 69, 70 e 71.

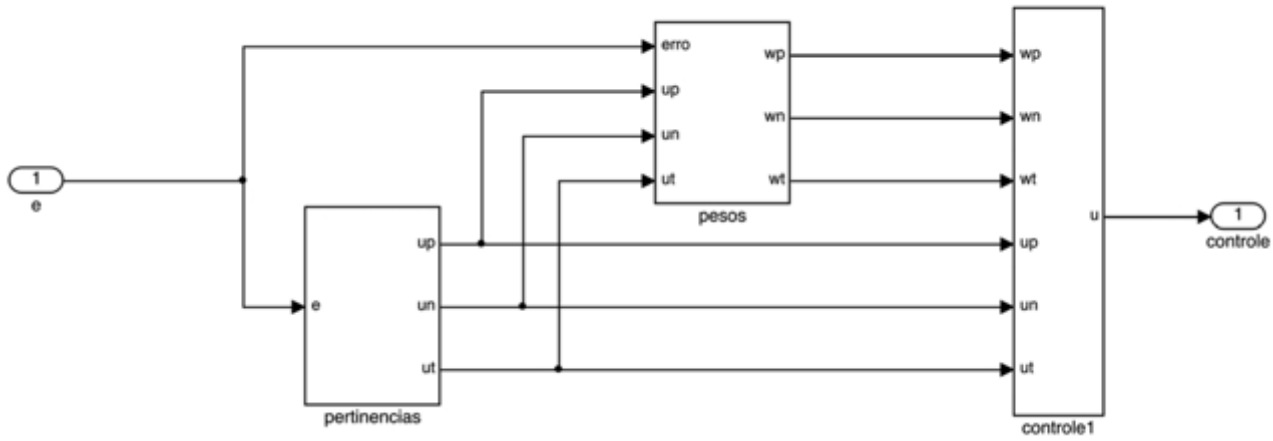


Figura 69 – Implementação da malha de controle do controlador ONFC para avaliação de controle e análise de parâmetros.

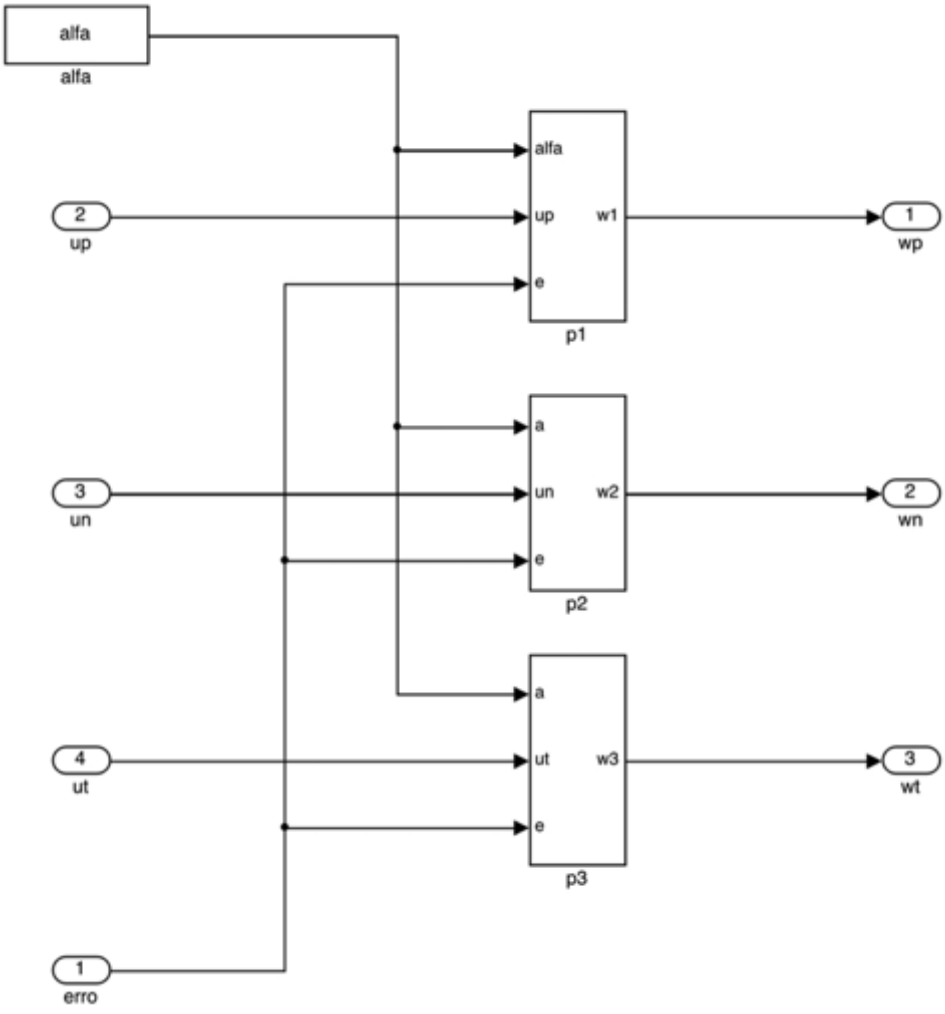


Figura 70 – Detalhe do bloco dos pesos do controlador ONFC.

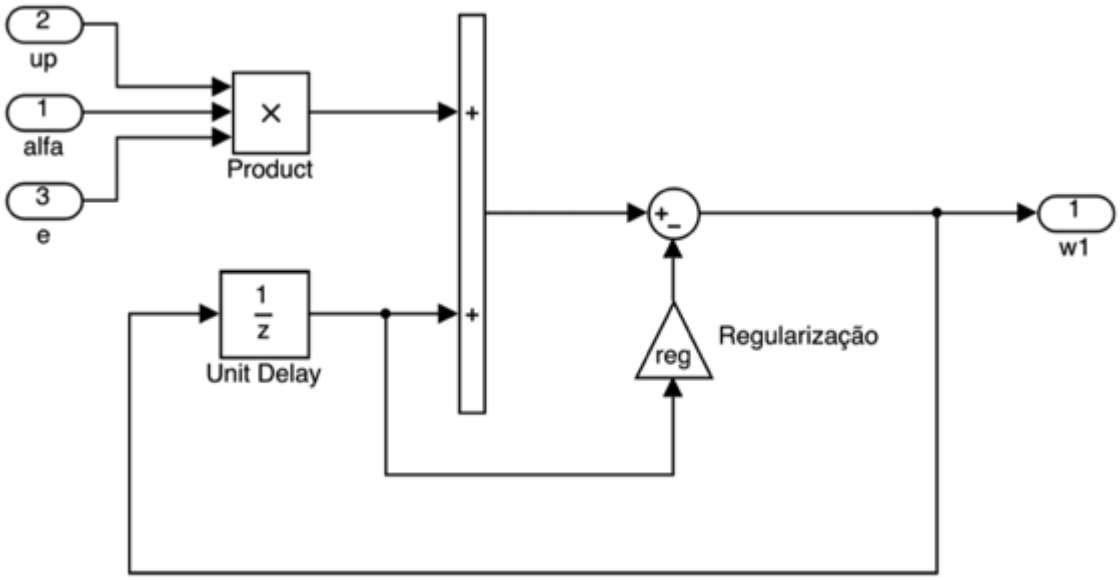
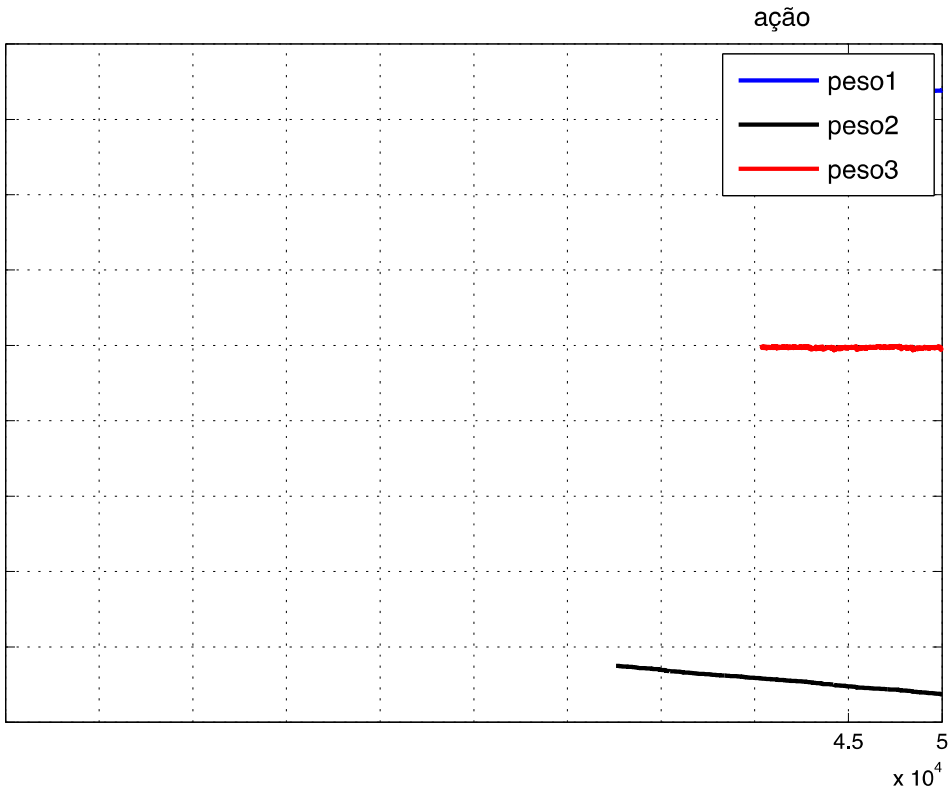


Figura 71 – Detalhe do bloco de regularização do controlador ONFC.

(A)



(B)

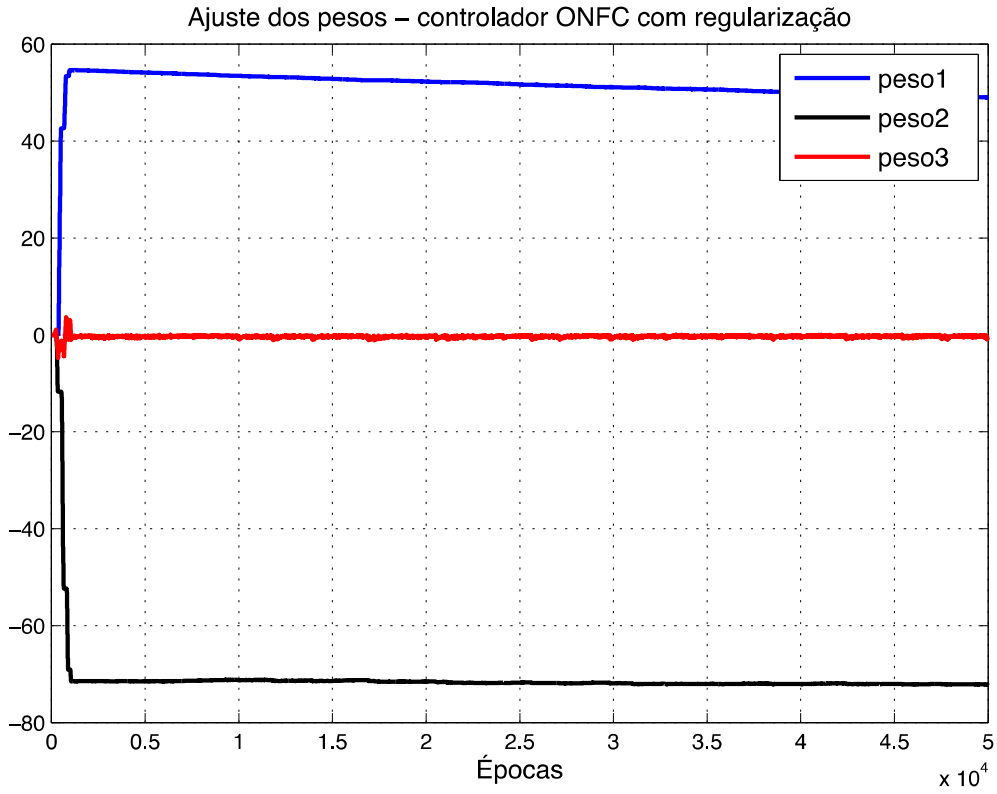


Figura 72 – Ajuste dos pesos controlador ONFC. (A) Sem regularização; (B) Com regularização.

Conforme figura 72, a adição da regularização prova apresentar desempenho adequado. A diferença dos pesos sem regularização tende a crescer em longos períodos. Enquanto que a diferença dos pesos com regularização tendeu a diminuir quando foi alcançado o regime permanente.

Diversos testes reais no auto-equilibrante NXT foram realizados para avaliar o efeito da variação dos termos utilizados para determinação de sintonia do ONFC: taxa de aprendizado (α), a taxa de regularização (γ) e o universo de discurso (e) das funções de pertinência, conforme tabela 7.

Tabela 7 – Análise de robustez da sintonia do $\boldsymbol{\gamma}$ em relação aos parâmetros $\boldsymbol{\alpha}$ e o universo de discurso.

Taxa de Aprendizado (α)	Taxa de Regularização (γ)	Universo Discurso	Controle
0,05	0,001	+100/ -100	Instável
0,05	0,003	+100/ -100	Instável
0,05	0,005	+100/ -100	Estável
0,05	0,007	+100/ -100	Instável
0,05	0,009	+100/ -100	Instável

Tabela 8 – Análise de robustez da sintonia do $\boldsymbol{\alpha}$ em relação aos parâmetros $\boldsymbol{\gamma}$ e o universo de discurso.

Taxa de Aprendizado (α)	Taxa de Regularização (γ)	Universo Discurso	Controle
0,01	0,005	+100/ -100	Instável
0,03	0,005	+100/ -100	Instável
0,05	0,005	+100/ -100	Estável
0,07	0,005	+100/ -100	Instável
0,09	0,005	+100/ -100	Instável

Tabela 9 – Análise do universo de discurso em relação aos parâmetros $\boldsymbol{\alpha}$ e $\boldsymbol{\gamma}$.

Taxa de Aprendizado (α)	Taxa de Regularização (γ)	Universo Discurso	Controle
0,05	0,005	+0,1/ -0,1	Instável
0,05	0,005	+0,5/ -0,5	Instável
0,05	0,005	+10/ -10	Instável
0,05	0,005	+50/ -50	Instável
0,05	0,005	+100/ -100	Estável

O melhor caso obtido ocorreu para uma taxa de aprendizado com valor de $\alpha = 0{,}05$ e taxa de regularização de $\gamma = 0{,}005$. O melhor universo de discurso das funções de pertinência foi de -100 a +100. A amplitude desse universo deve-se diretamente aos *spikes* verificados na saída do sensor gyro. Na figura 74 é

possível observar o ajuste dos pesos ao longo do tempo para essa condição paramétrica. Nas figuras 75 e 76 observa-se o deslocamento angular e linear, respectivamente, obtidos durante a operação do NXT na condição paramétrica de melhor resultado.

Verifica-se, nas implementações realizadas, que valores muito baixos da taxa de aprendizado impedem uma progressão rápida do ajuste dos pesos, fazendo com que a diferença entre eles se torne muito pequena. Portanto, a instabilidade é observada devido a uma resposta lenta dos atuadores sobre o auto-equilibrante. Por outro lado, valores muito altos da taxa de aprendizado, apesar de permitirem a resposta rápida dos atuadores, resulta em elevados *overshoots* ao sistema que podem levar a inclinações verticais grandes o suficiente para ultrapassar as regiões de controlabilidade.

Da mesma forma, valores muito altos de regularização provocam uma variação abrupta nos valores dos pesos, podendo levar ao reinício dos pesos. Como consequência é observada a redução de desempenho no controle de equilíbrio do auto-equilibrante. Já valores muito baixos de regularização apresentam a mesma resposta dinâmica do controle ONFC sem regularização. Ou seja, ao longo do tempo a diferença entre os pesos pode ser grande o suficiente para impedir o auto-equilíbrio.

Explorando um pouco mais a região em torno do valor da taxa de aprendizado de $\alpha = 0{,}05$ que garantiu a estabilidade do robô, verificou-se experimentalmente que o sistema permaneceu estável para uma tolerância de $\alpha = \pm 0{,}005$ (Tabela 10 e figura 73).

Tabela 10 – Análise de robustez da sintonia do $\boldsymbol{\alpha}$ em relação aos parâmetros $\boldsymbol{\gamma}$ e o universo de discurso.

Taxa de Aprendizado (α)	Taxa de Regularização (γ)	Universo Discurso	Controle
0,045	0,005	+100/ -100	Estável
0,050	0,005	+100/ -100	Estável
0,055	0,005	+100/ -100	Estável

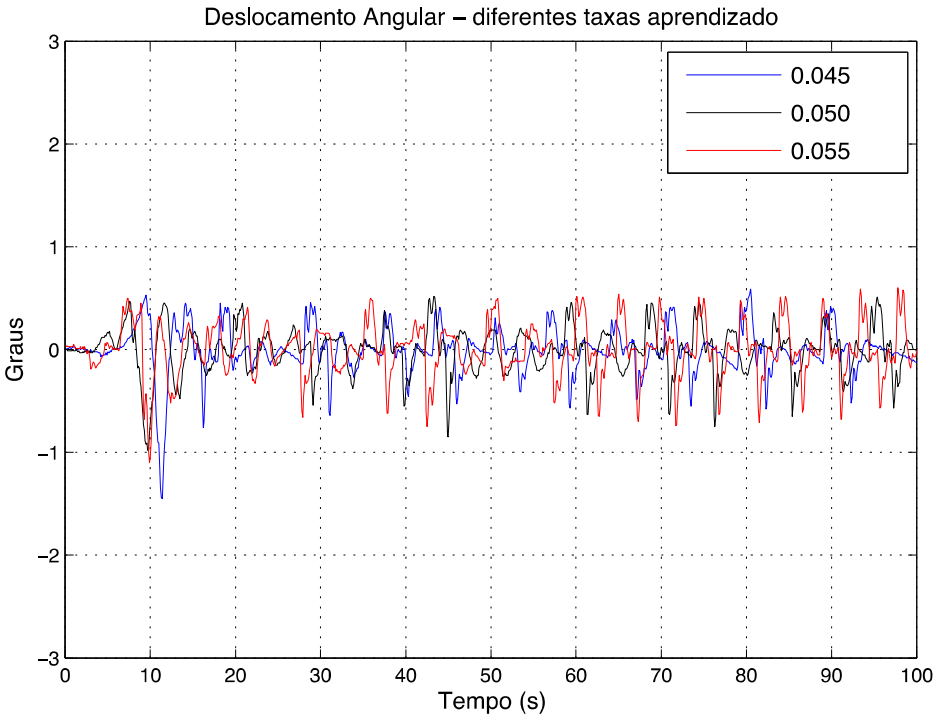


Figura 73 – Resposta do deslocamento angular para diferentes taxas de aprendizado.

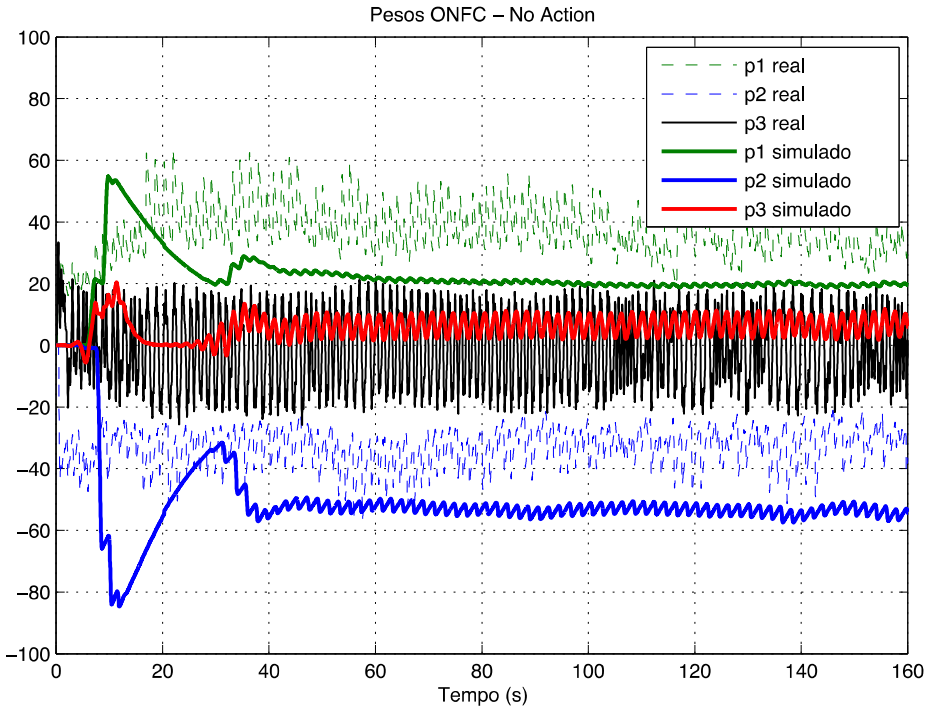


Figura 74 – Ajuste dos pesos do controlador ONFC.

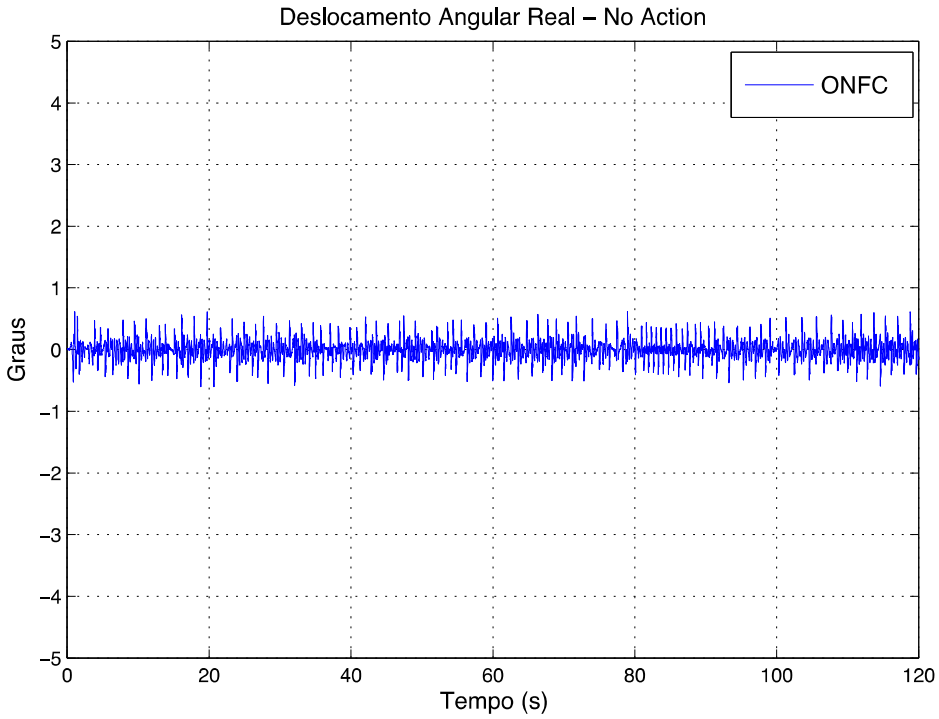


Figura 75 – Deslocamento angular para o controle ONFC estacionário.

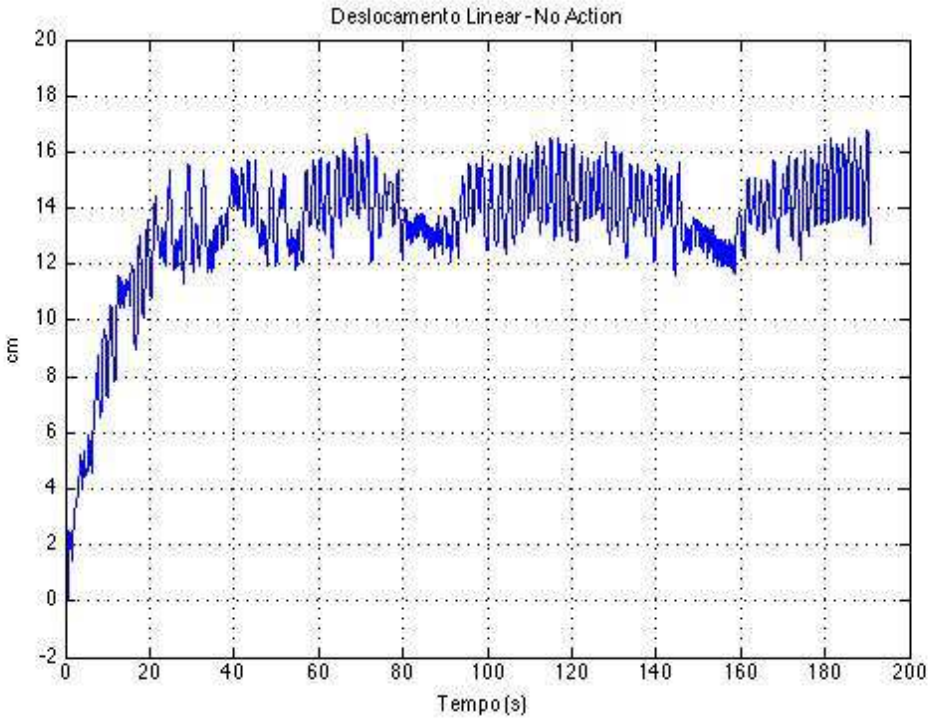


Figura 76 – Deslocamento linear para o controle ONFC estacionário.

- **Filtragem dos pesos**

O filtro de média móvel apresenta bom desempenho em muitas aplicações e ótimo desempenho na redução de ruído branco, ao mesmo tempo que preserva a resposta ao degrau. Para reduzir a oscilação frequente na atualização do pesos do robô foi implementado um filtro de média móvel na equação de atualização dos pesos. Conforme resultados experimentais (figura 77), a melhor ordem do filtro foi para M=2. Ordens superiores a M=2 provocam instabilidade ao sistema, pois torna a resposta dinâmica mais lenta. Na figura 78-B, observa-se a redução da oscilação dos pesos w_p e w_n indicando a redução de oscilação durante o equilíbrio estacionário.

(A) (B)

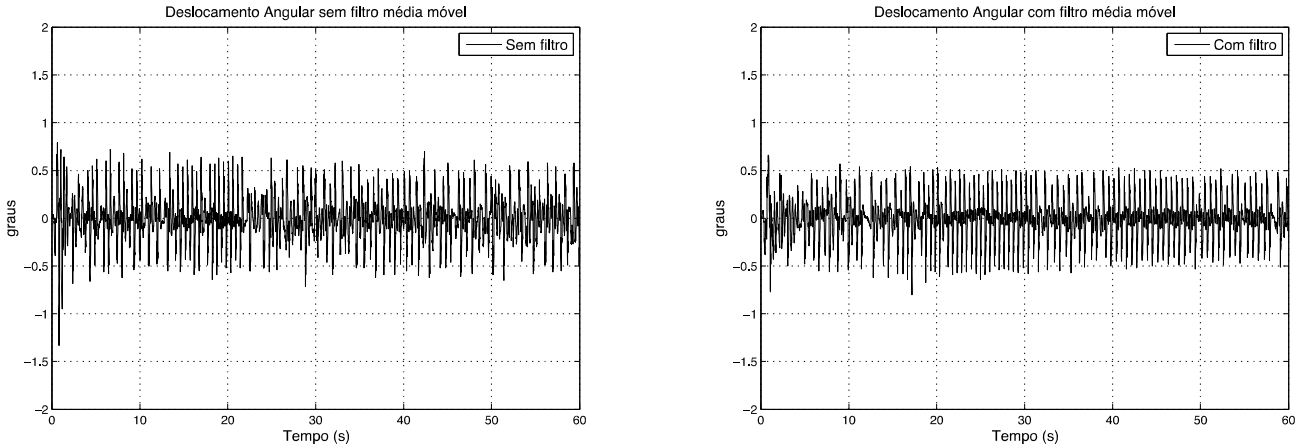


Figura 77 – Implementação do filtro de média móvel: (A) teta sem filtro; (B) teta com filtro.

(A) (B)

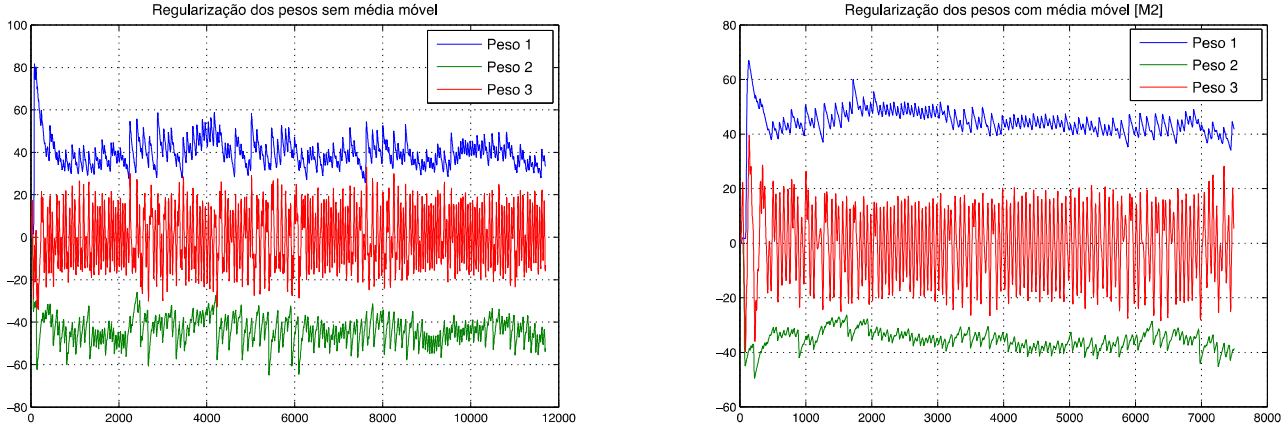


Figura 78 – Implementação do filtro de média móvel: (A) pesos sem filtro; (B) pesos com filtro.

6.2 CONTROLE NEURO ARTIFICIAL

As redes neurais artificiais são sistemas computacionais de processamento paralelo e distribuído baseado no modelo biológico do neurônio ou sistema nervoso, possuindo a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso. O poder computacional da rede neural depende de sua estrutura paralelamente distribuída e sua habilidade de aprendizagem. Essas capacidades tornam as redes neurais capazes de resolver problemas complexos.
Ela se assemelha ao cérebro em dois aspectos:

- Conhecimento adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem.
- Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar conhecimento adquirido.

Uma representação de rede neural pode ser vista na figura 79.

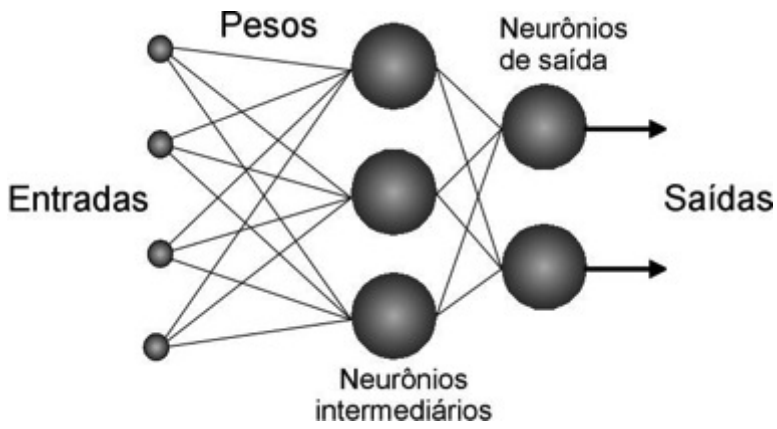


Figura 79 – Representação da rede neural artificial multicamadas.

A partir de um algoritmo de aprendizado é efetuado a aprendizagem da rede neural, onde os pesos sinápticos são atualizados durante a execução da mesma com a finalidade de se alcançar um resultado desejado (HAYKIN, 2003).

Dentre as principais propriedades das redes neurais pode se destacar:

- Aprendizado: a rede é treinada a partir de exemplos.
- Não linearidade: um neurônio artificial pode ser linear ou não-linear, sua aplicação é adequada para sistemas físicos que geralmente são não lineares.
- Capacidade de adaptação: apresenta a propriedade de se adaptar a nova informação.
- Capacidade de generalização: uma rede neural possui uma capacidade de generalização do conhecimento adquirido a partir da existência de dados incompletos ou imprecisos, uma vez que pode preencher lacunas sem sofrer degradação, em uma referência muito próxima à interpolação e extrapolação no campo da estatística, mas com uma execução muito diferente.
- Informação Contextual: O conhecimento adquirido é representado pela estrutura neural e o estado de ativação dos neurônios da rede. Como cada neurônio é afetado de todos os neurônios da rede a informação contextual é tratada
- Tolerância a Falhas: é capaz de realizar computação robusta sob condições de operação adversas, onde seu desempenho se degrada suavemente por estar relacionado ao processamento paralelo e distribuído da rede.

O controlador PIDNN (*Proportional-Integral-Derivative Neural-Networks*) (FERRARI, 2011) é baseado no modelo da equação discreta da função de transferência do controlador PID. A equação genérica da função do controlador PID no domínio s é dada:

$$C(s) = \frac{\mathrm{u}(s)}{e(s)} = \mathrm{K}_p + \frac{K_i}{s} + K_d s \tag{70}$$

Onde K_p, K_i e K_d correspondem respectivamente aos ganhos proporcional, integral e derivativo. Aproximando a parte integral pelo método forward e a parte derivativa usando backward, resulta em:

$$\frac{\mathrm{u(z)}}{e(z)} = \mathrm{K}_p + K_i\left(\frac{\Delta t}{z-1}\right) + K_d\left(\frac{z-1}{z\Delta t}\right) \tag{71}$$

Desenvolvendo a equação do sinal de controle de tempo discreto $\mathrm{u(z)}$, vem que:

$$\mathrm{u(z)} = \mathrm{K}_p e(z) + K_i\left(\frac{\Delta t}{z-1}\right)e(z) + K_d\left(\frac{z-1}{z\Delta t}\right)e(z) \tag{72}$$

Para simplificar a equação (72) podemos considerar os parâmetros $v(z)$ e $w(z)$, tais que:

$$v(z) = \left(\frac{\Delta t}{z-1}\right)e(z)$$

$$w(z) = \left(\frac{z-1}{z\Delta t}\right)e(z)$$

A equação em tempo discreto para o termo integral *v(n)* e o termo derivativo *w(n)* fica:

$$\mathrm{v(n)} = \mathrm{v(n-1)} + \Delta\mathrm{t.\,e(n)} \tag{73}$$

$$\mathrm{w(n)} = \frac{1}{\Delta t}\left(e(n) - e(n-1)\right) \tag{74}$$

Resultando na equação de controle em tempo discreto:

$$\mathrm{u(n)} = \mathrm{K}_p e(n) + K_i v(n) + K_d w(n) \tag{75}$$

Ferrari (2011) mostra que a partir da combinação das equações do controlador PID de tempo discreto obtém-se uma rede neural recorrente (figura 80). Tal que a entrada da rede é o sinal de erro e(n) e a saída é o sinal de controle u(n). A função de ativação dos neurônios é linear.

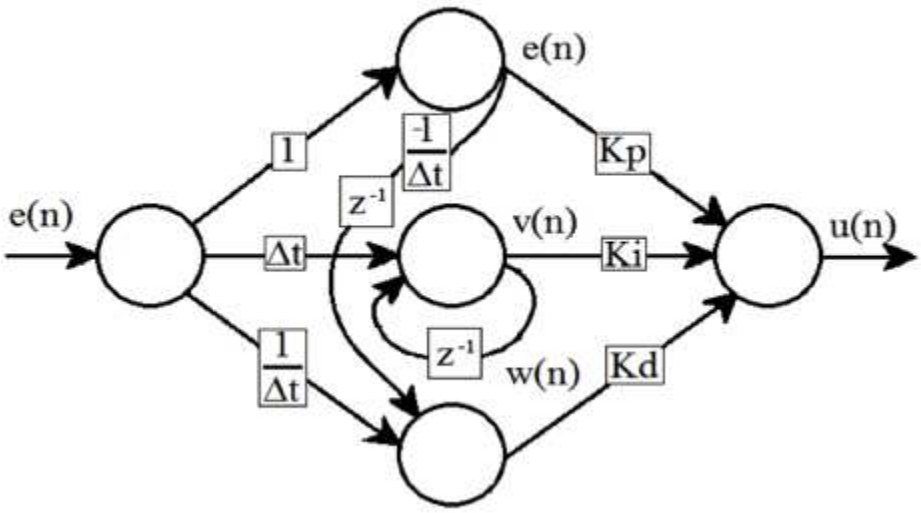


Figura 80 – Estrutura do controlador PIDNN

Os ganhos *Kp, Ki* e *Kd* da estrutura neural do controlador PID (Figura 41) serão os pesos que serão sintonizados e o restante dos pesos possuem valores fixos. Considerando $O_1(n)$, $O_2(n)$ e $O_3(n)$ as saídas dos neurônios da camada intermediária, a saída da rede pode ser computada seguindo as seguintes equações:

$$e(n) = O_1(n) \tag{76}$$

$$v(n) = O_2(n) = O_2(n-1) + \Delta t.\, e(n) \tag{77}$$

$$w(n) = O_3(n) = \frac{1}{\Delta t}(e(n) - e(n-1)) \tag{78}$$

O algoritmo de treinamento do controlador PIDNN é baseado no método de descida do gradiente e utiliza o sinal de erro $\varepsilon(n)$ que é a diferença entre o sinal desejado *d(n)* e a resposta de saída da rede neural.

$$\varepsilon(n) = d(n) - u(n) \tag{79}$$

6.2.1 Equações de ajuste dos pesos do PIDNN

Ferrari (2011) apresenta duas formas de sintonia dos pesos do controlador PIDNN: através do treinamento off-line e da sintonia online.

Primeiramente o PIDNN é treinado off-line, e os dados são gerados de um sistema PID simulado e usado para treinar a rede neural. Portanto, a aprendizagem é supervisionada pela regra por correção do erro. O desempenho desejado é minimizado pela soma dos erros E_{sq} do treinamento dos dados.

$$\mathrm{E}_{sq}(n) = \frac{1}{2}\sum_{n=1}^{N} \varepsilon(n)^2 \tag{80}$$

De acordo com método da descida do gradiente, a atualização dos pesos segue as seguintes derivações.

$$\mathrm{K}_p(n+1) = K_p(n) + \alpha\varepsilon(n)O_1(n) \tag{81}$$

$$\mathrm{K}_i(n+1) = K_i(n) + \alpha\varepsilon(n)O_2(n) \tag{82}$$

$$\mathrm{K}_d(n+1) = K_d(n) + \alpha\varepsilon(n)O_3(n) \tag{83}$$

Posteriormente ao treinamento off-line, o PIDNN é colocado em série com o sistema para sintonia online (figura 81). Neste modo de aprendizado a atualização dos pesos é realizada após a apresentação de cada exemplo de treinamento. A diferença da resposta da saída da planta em relação ao sinal de referência gera o erro e_1 que será o sinal de entrada do controlador. O erro e_2 é gerado pela diferença da resposta de saída da planta em relação à resposta do modelo de referência. Este sinal está diretamente relacionado com a atualização dos pesos online.

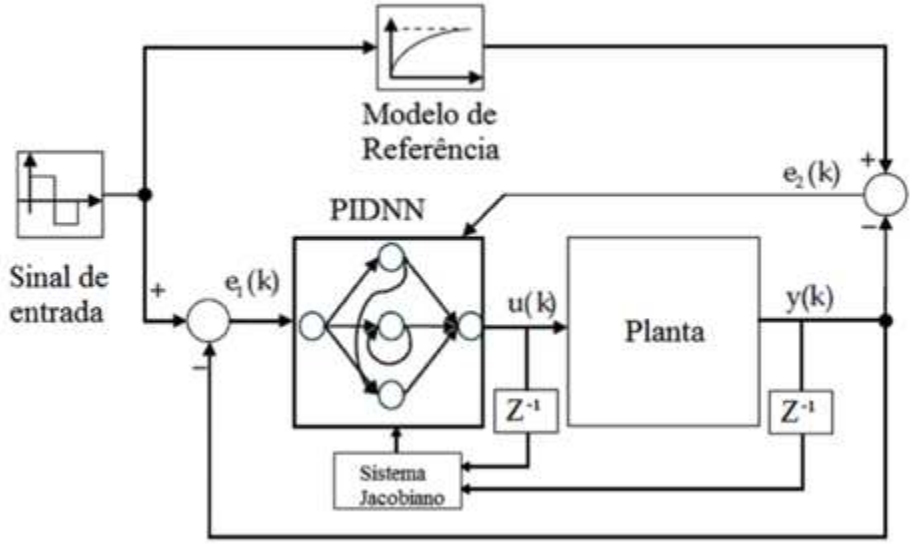


Figura 81 – Malha de controle da sintonia *on-line* (FERRARI, 2011).

A adaptação dos pesos ocorre pela minimização da função de custo $\mathrm{I(n)}$:

$$\mathrm{I(n)} = \frac{1}{2} e_2(n)^2 \tag{84}$$

$$\mathrm{K}_p(n+1) = K_p(n) + \alpha[e_2(n).J(n)]O_1(n) \tag{85}$$

$$\mathrm{K}_i(n+1) = K_i(n) + \alpha[e_2(n).J(n)]O_2(n) \tag{86}$$

$$\mathrm{K}_d(n+1) = K_d(n) + \alpha[e_2(n).J(n)]O_3(n) \tag{87}$$

Tal que $J(n)$ é o sistema jacobiano, que para um sistema MIMO, com *n* entradas e *m* saídas pode ser definido pela matriz de equação (88):

$$J = \frac{\partial y}{\partial u} = \begin{bmatrix} \frac{\partial y_1}{\partial u_1} & \cdots & \frac{\partial y_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial u_1} & \cdots & \frac{\partial y_m}{\partial u_n} \end{bmatrix} \tag{88}$$

Tal que $y = [y_1 y_2 \dots y_n]^T$ é o vetor de saída e $u = [u_1 u_2 \dots u_n]^T$ é o vetor de entrada para o sistema.

Para o sistema estudado, com 1 entrada e 4 saídas o jacobiano é dado por:

$$J = \frac{\partial y}{\partial u} = \begin{bmatrix} \frac{\partial y_1}{\partial u_1} \\ \frac{\partial y_2}{\partial u_2} \\ \frac{\partial y_3}{\partial u_3} \\ \frac{\partial y_4}{\partial u_4} \end{bmatrix} \tag{89}$$

Considerando uma tempo de amostragem pequeno, o jacobino da planta pode ser aproximado pela aproximação da derivada:

$$J = \frac{\partial y}{\partial u} \approx \frac{y(n) - y(n-1)}{u(n) - u(n-1)} \tag{90}$$

Tal que o jacobiano pode ser recalculado para:

$$J = \frac{\partial y}{\partial u} = \begin{bmatrix} \frac{y_1(n) - y_1(n-1)}{u_1(n) - u_1(n-1)} \\ \frac{y_2(n) - y_2(n-1)}{u_2(n) - u_2(n-1)} \\ \frac{y_3(n) - y_3(n-1)}{u_3(n) - u_3(n-1)} \\ \frac{y_4(n) - y_4(n-1)}{u_4(n) - u_4(n-1)} \end{bmatrix} \tag{91}$$

O modelo de referência é gerador *on-line* com o algoritmos de controle no sistema em simulação. Ferrari (2011) considera o modelo de referência desejado como um sistema em malha fechada de terceira ordem, criticamente amortecido possuindo dois pólos com o mesmo valor. O valor do parâmetro que determina o valor do terceiro pólo é escolhido de tal forma que seu efeito determine o tempo de acomodação (OGATA, 1982).

6.2.2 Implementação do controle PIDNN

Para avaliação do algoritmo proposto foi configurada uma estrutura no ambiente de simulação computacional Simulink conforme apresentando na Figura 82.

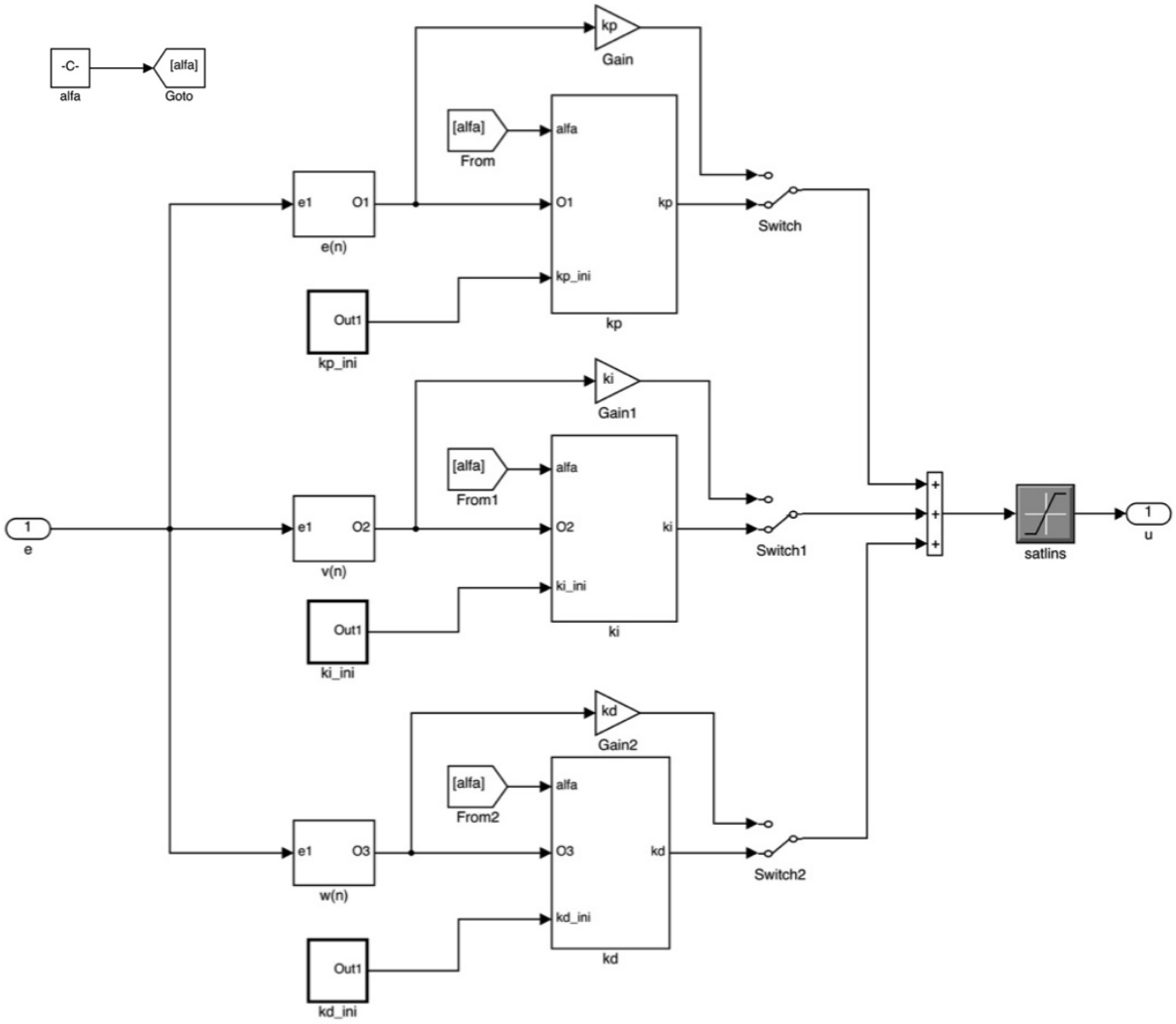


Figura 82 – Implementação no Simulink do controlador PIDNN para treinamento *off-line.*

Na plataforma de simulação da figura 82, é possível simular a atualização dos pesos, através do treinamento off-line quando a chave de saída da última camada de blocos está na posição 2 (chave fechada para baixo). Após o treinamento, as chaves de saída do controlador podem ser colocadas na posição 1 (chave fechada para cima) para teste do controlador sintonizado.

Para um valor de taxa de aprendizado α, fixo em $\alpha = 1 * 10^{-7}$, a atualização dos pesos é apresentada conforme figura 83. Em testes, valores superiores de α levaram à instabilidade do NXT.

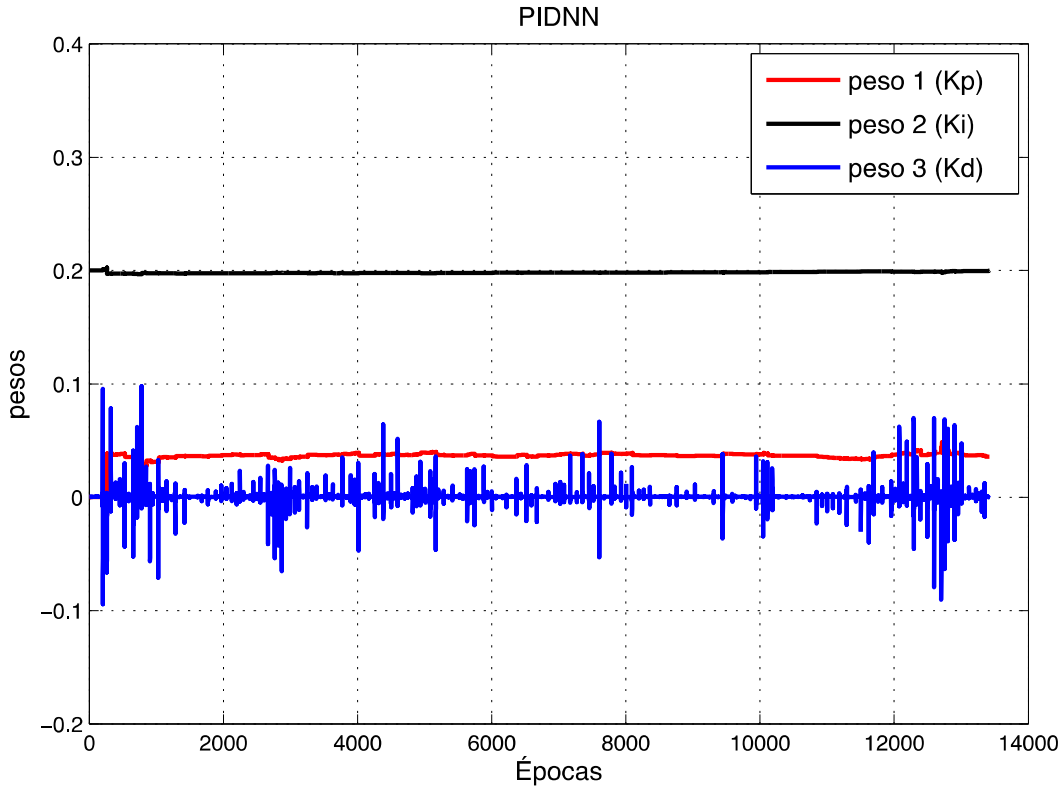


Figura 83 – Atualização dos pesos off-line na simulação.

O peso médio dos ganhos do controlador podem ser definidos conforme tabela 11.

Tabela 11 – Pesos sintonizados pelo treinamento off-line.

Ganhos do controlador	Peso médio sintonizado
k_p	0.0364
K_i	0.1998
k_d	0.0150

Com os pesos médios sintonizados foi possível fazer o teste real no NXT. Comparando na figura 84 o deslocamento angular do controlador PIDNN com o controlador PID foi possível verificar que os ajuste dos ganhos do controlador

ocorreu de forma satisfatória. Observa-se que a estabilidade do NXT foi alcançada.

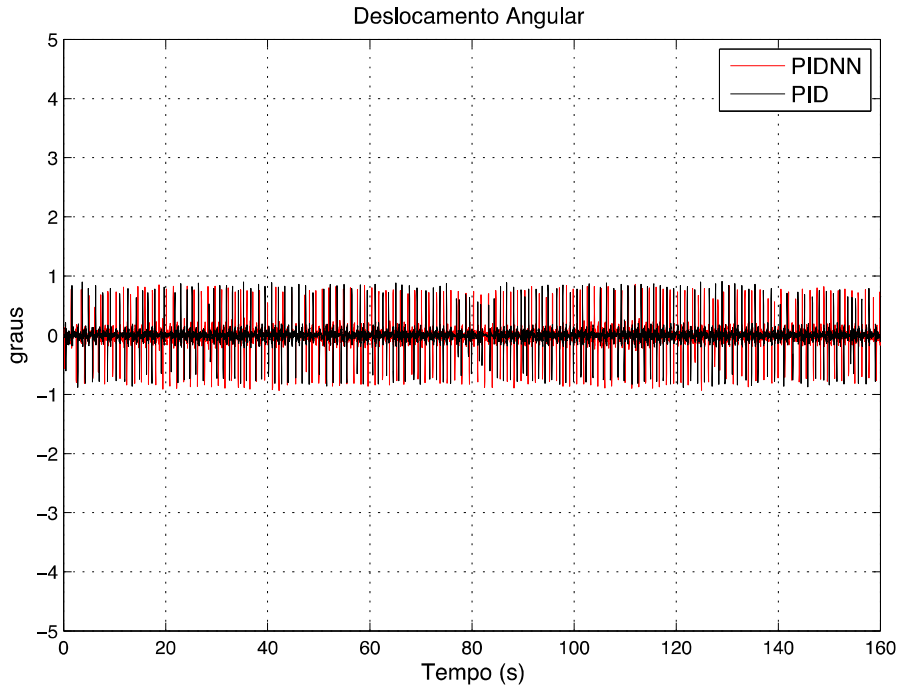


Figura 84 – Comparativo entre o deslocamento angular real do controle PIDNN e do PID.

Para implementação do treinamento *on-line* foi desenvolvida uma estrutura no ambiente de simulação computacional Simulink conforme apresentando na Figuras 85 e 86.

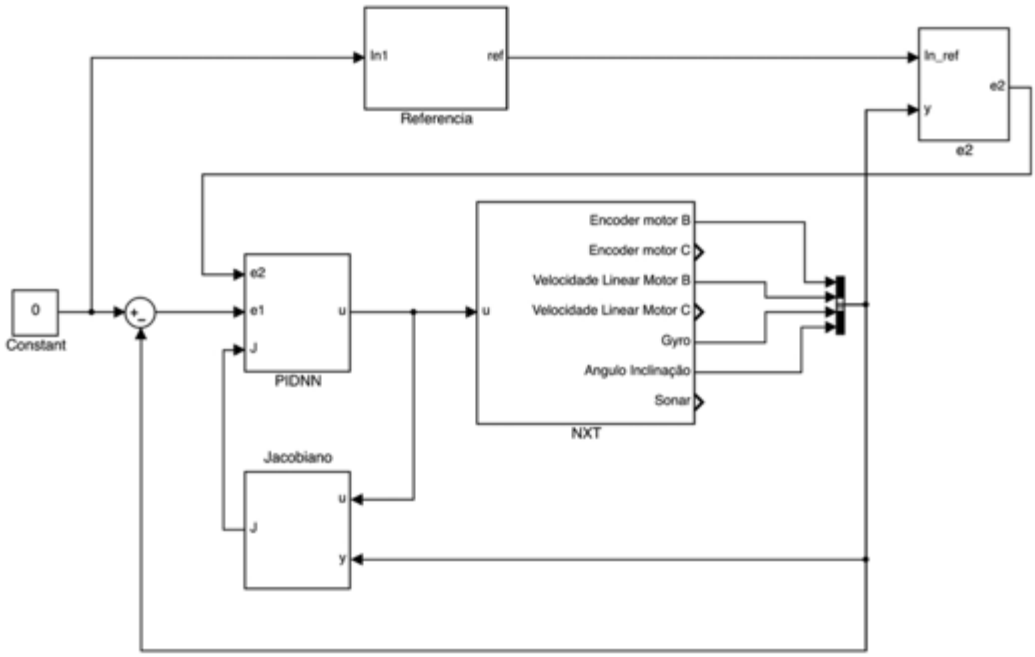


Figura 85 – Malha de controle para o treinamento online.

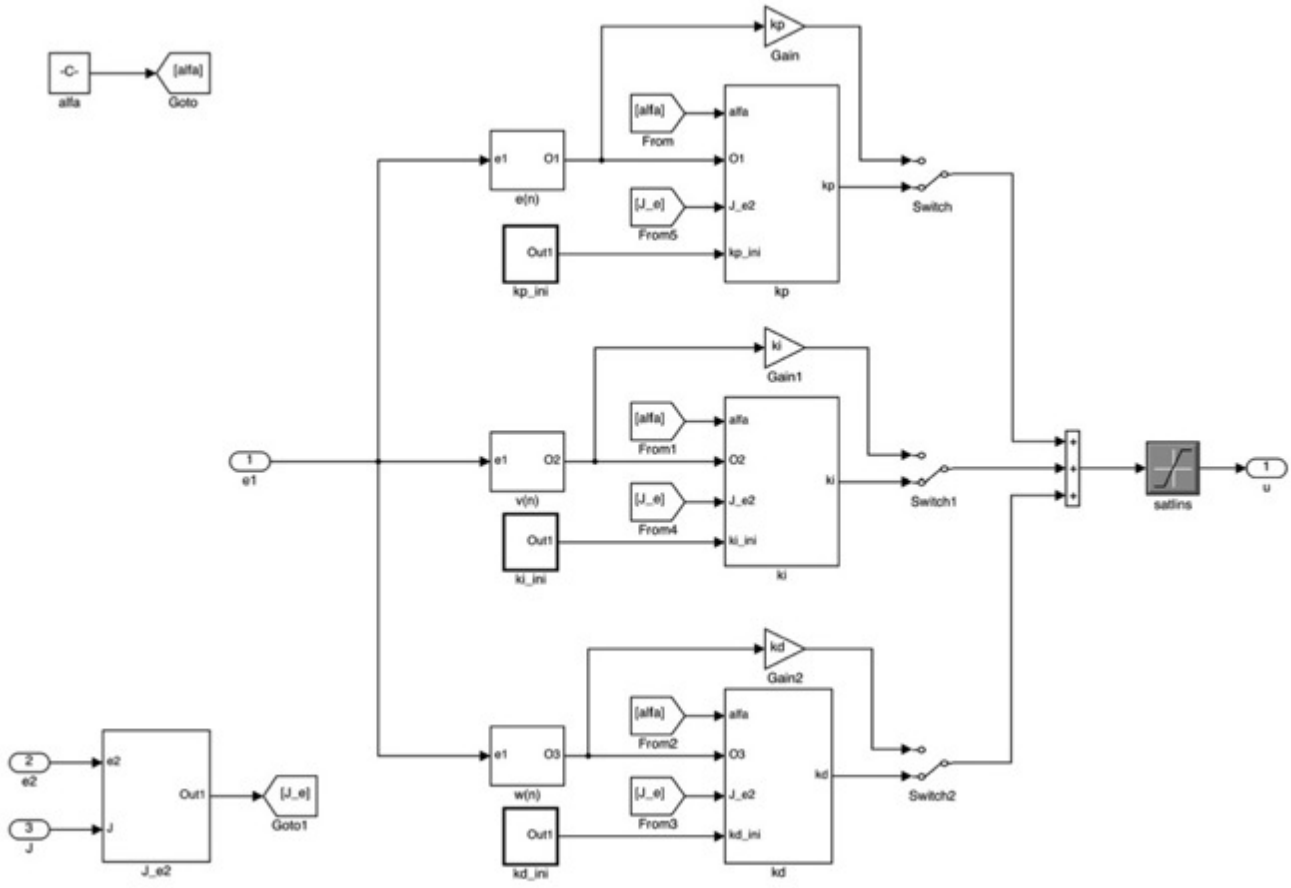


Figura 86 – Implementação no Simulink do controlador PIDNN para treinamento online.

Na plataforma de simulação da figura 86, é possível simular a atualização dos pesos, através do treinamento *on-line* quando a chave de saída da última camada de blocos está na posição 2 (chave fechada para baixo). Após treinamento, a as chaves de saída do controlador podem ser colocadas na posição 1 (chave fechada para cima) para teste do controlador sintonizado.

Para um valor de taxa de aprendizado α, fixo em $\alpha = 0.0001$, a atualização dos pesos é apresentada conforme figura 87.

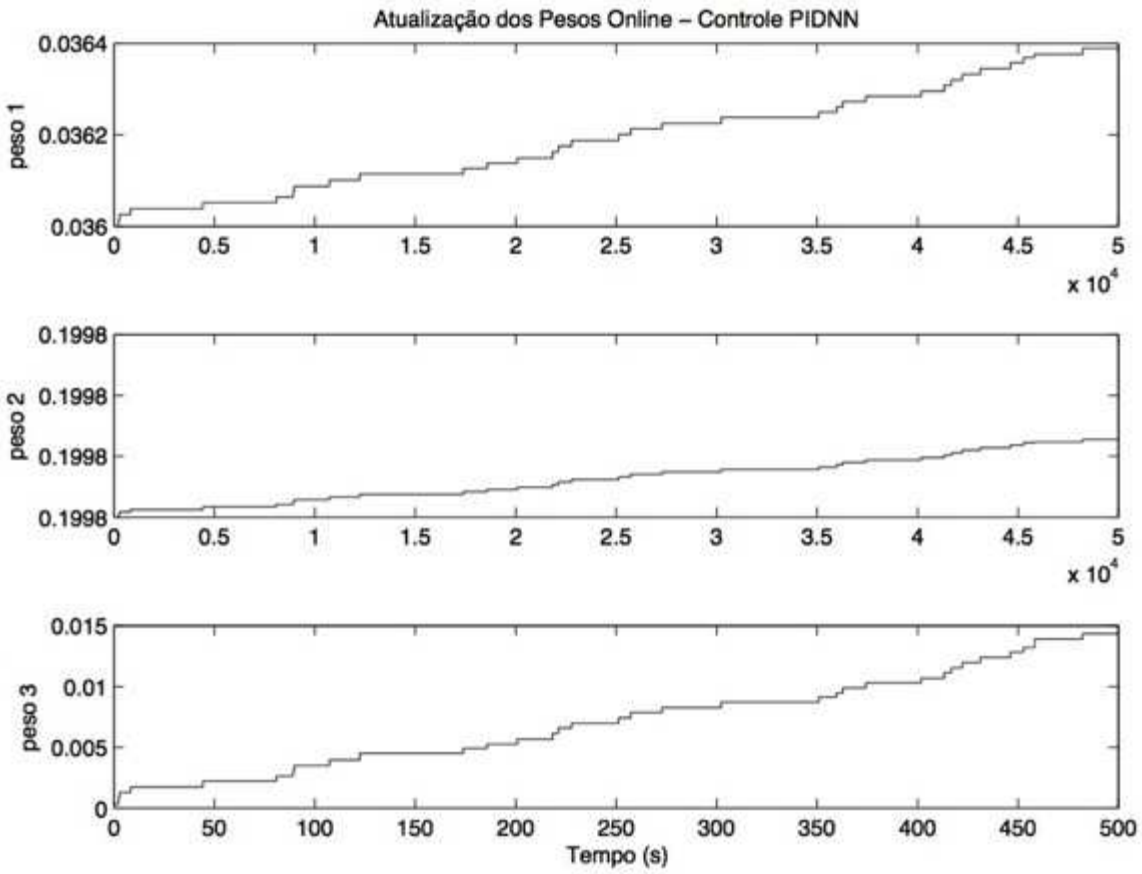


Figura 87 – Variação dos ganhos K_p, K_i e K_d na simulação.

O peso médio dos ganhos do controlador podem ser definidos conforme tabela 12.

Tabela 12 – Pesos sintonizados pelo treinamento online.

Ganhos do controlador	Peso médio sintonizado
k_p	0,0360
K_i	0,1998
k_d	0,000358

Com os pesos médios sintonizados foi possível fazer o teste em simulação do NXT. Comparando o deslocamento angular do controlador PIDNN com o controlador PID foi possível verificar que os ajuste dos ganhos do controlador ocorreu de forma satisfatória. Observa-se que a estabilidade do NXT foi alcançada.

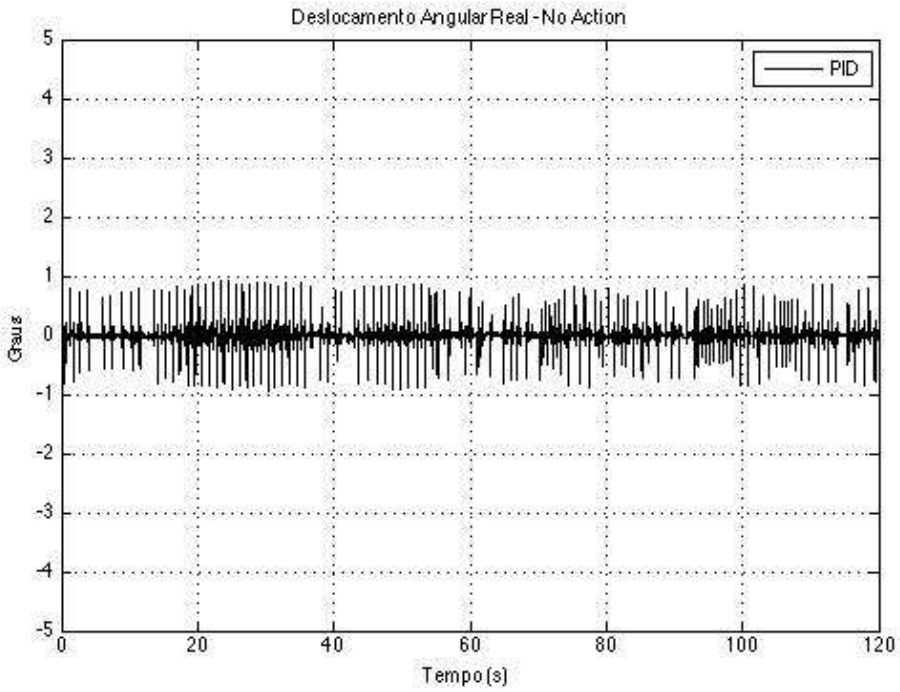
6.3 Estudo de caso 1

Os desempenhos obtidos em testes reais com o robô auto-equilibrante foram analisados entre os controladores estudados: PID, ONFC e PIDNN. A resposta do controle PID serviu apenas como referência para comparar o desempenho dos controladores inteligentes com o controlador clássico. Dois casos foram estudados para essa análise: controle em terreno plano e controle em terreno com obstáculos.

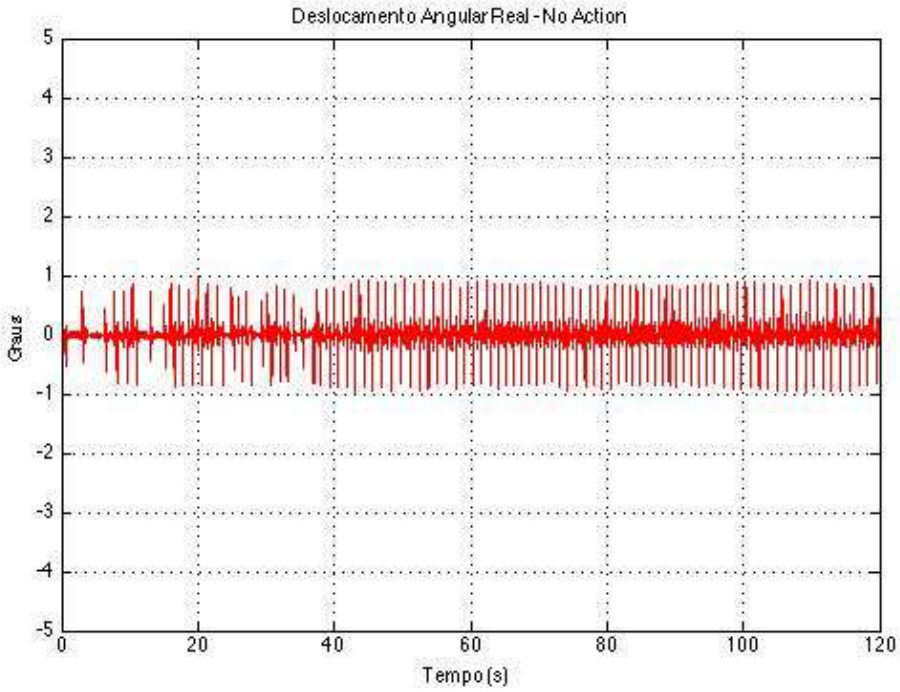
6.3.1 Controle de Equilíbrio em terreno plano

Observa-se que a resposta do deslocamento angular (figura 88) dos três controladores estudados PID, PIDNN e ONFC, na operação de controle de equilíbrio em terreno plano, está limitada na faixa de $\pm 1^o$ grau. Esse resultado é satisfatório pois nessa faixa de trabalho, o robô auto-equilibrante consegue manter a estabilidade de seu equilíbrio. É importante mencionar que essa faixa de operação está associada à faixa de tolerância do erro do sensor gyro ($\pm 1^o$ grau).

(A)


Deslocamento Angular Real - No Action
PID
Graus
Tempo (s)
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
20
40
60
80
100
120

(B)


Deslocamento Angular Real - No Action
Graus
Tempo (s)
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
20
40
60
80
100
120

(C)

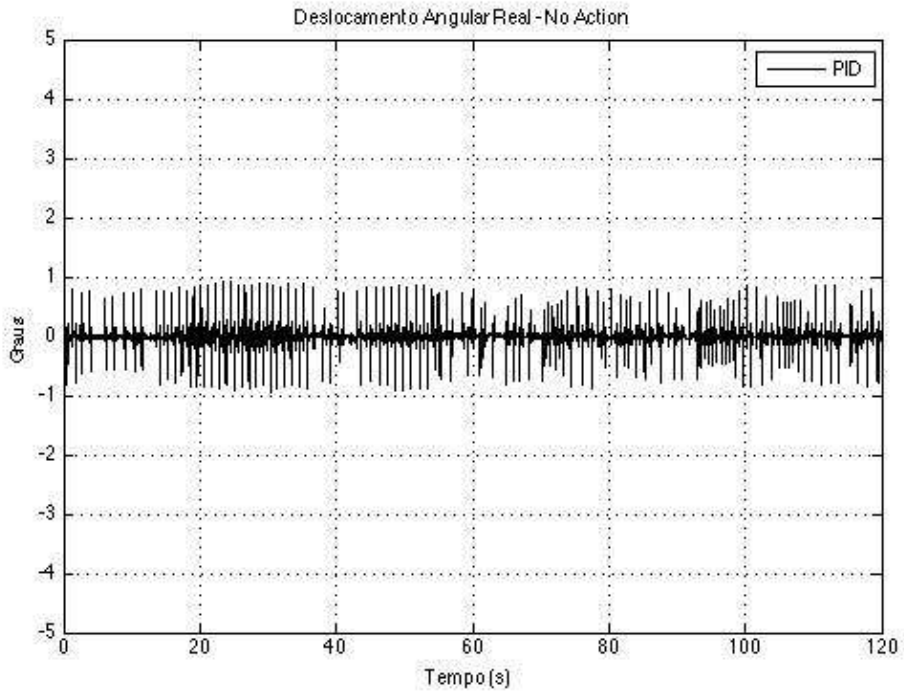


Figura 88 – Controle de Equilíbrio dos controladores: (A) ONFC; (B) PIDNN;(C) PID: deslocamento angular.

Com relação ao deslocamento linear, observa-se na figura 89, que todos os controladores estudados, na operação de controle de equilíbrio em terreno plano, apresentaram resposta com deslocamento linear superior a 10 cm. Em regime estacionário, nota-se um deslocamento máximo de até 16 cm para o controladores PIDNN e ONFC. O controle PID teve um deslocamento máximo de 14 cm.

Esse deslocamento observado em todos os controladores indica o efeito do problema de *bias* do sensor gyro. O ângulo calculado pode continuamente aumentar ou diminuir, mesmo quando o auto-equilibrante estiver estacionário na posição de equilíbrio. Este escorregamento angular também causa o deslocamento linear, pois para compensar o erro angular o robô afasta-se da posição de referência criando um erro de posição.

Outra razão para o deslocamento linear do robô na operação estacionária pode estar associada à condição inicial. Como a iniciação do auto-equilibrante depende do posicionamento manual na vertical, o ponto de equilíbrio perfeito (vetor nulo das variáveis) não é garantido, sempre observará um deslocamento

linear até ser alcançado a correção no sistema. O tempo observado de correção e estabilização na posição estacionária é de aproximadamente 20 segundos.

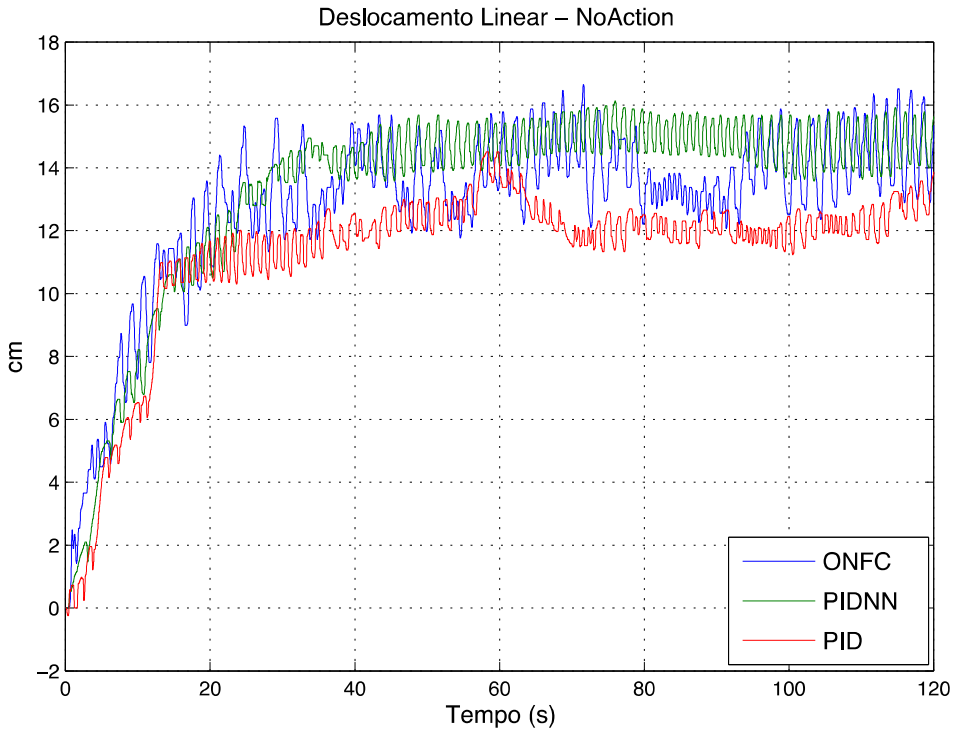


Figura 89 – Controle de Equilíbrio dos controladores ONFC, PIDNN e PID: deslocamento linear.

A figura 90 mostra a superioridade do controlador ONFC para a rejeição a pequena perturbação externa. A ação de controle durante o momento da perturbação é inferior relativamente ao controlador PID. Assim a variação da saída, especificamente o deslocamento angular, mostrou-se inferior em relação ao controle PID. O controlador PID-NN foi omitido dessa análise por ter uma resposta nessa condição muito similar ao PID.

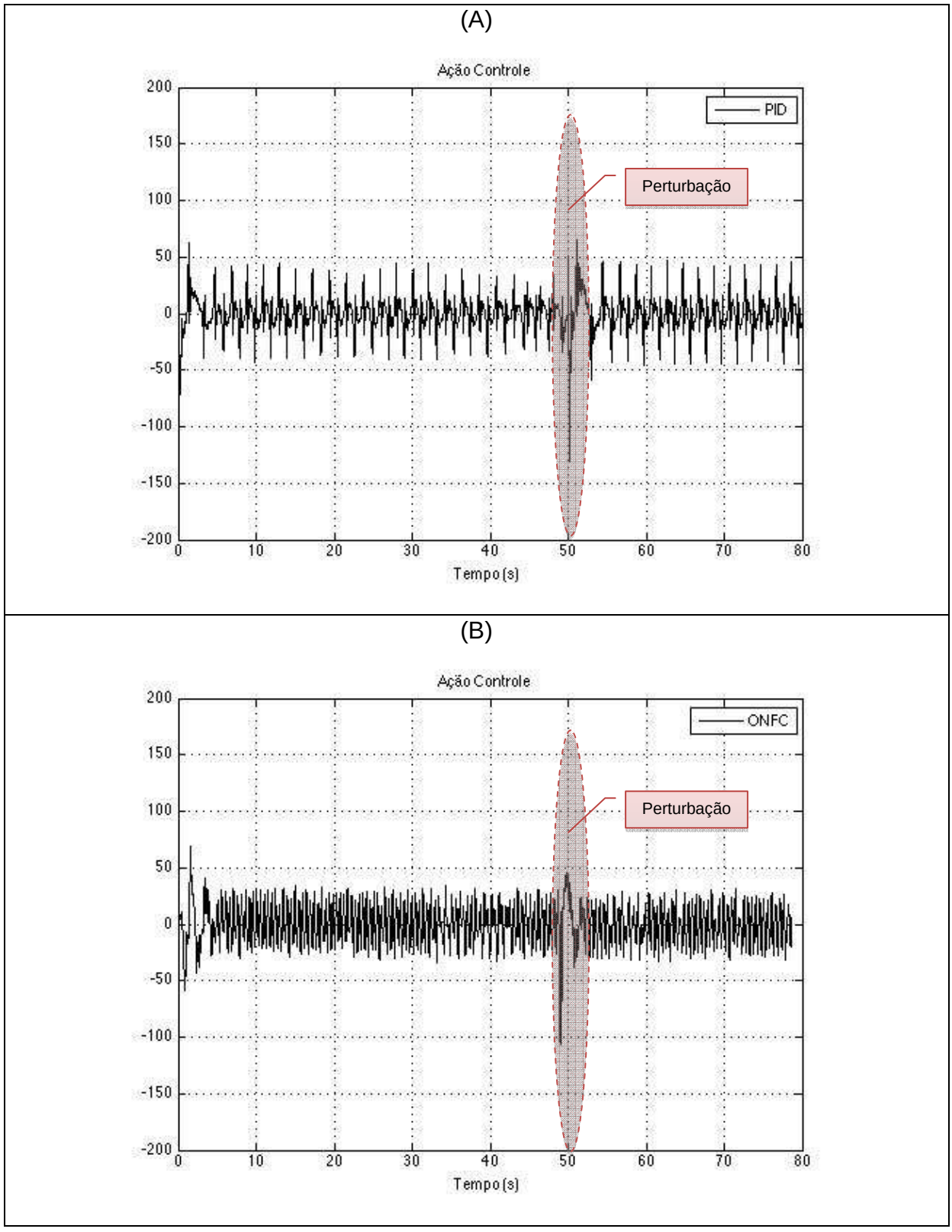


Figura 90 – Comparação entre o desempenho do PID e do ONFC para rejeição à perturbação externa: (A) Resposta do sistema com controlador PID; (B) Resposta do sistema com controlador ONFC.

A figura 91 mostra o momento de atualização dos pesos do controlador ONFC para rejeição à perturbação externa ao auto-equilibrante, demonstrando a vantagem adaptativa do algoritmo.

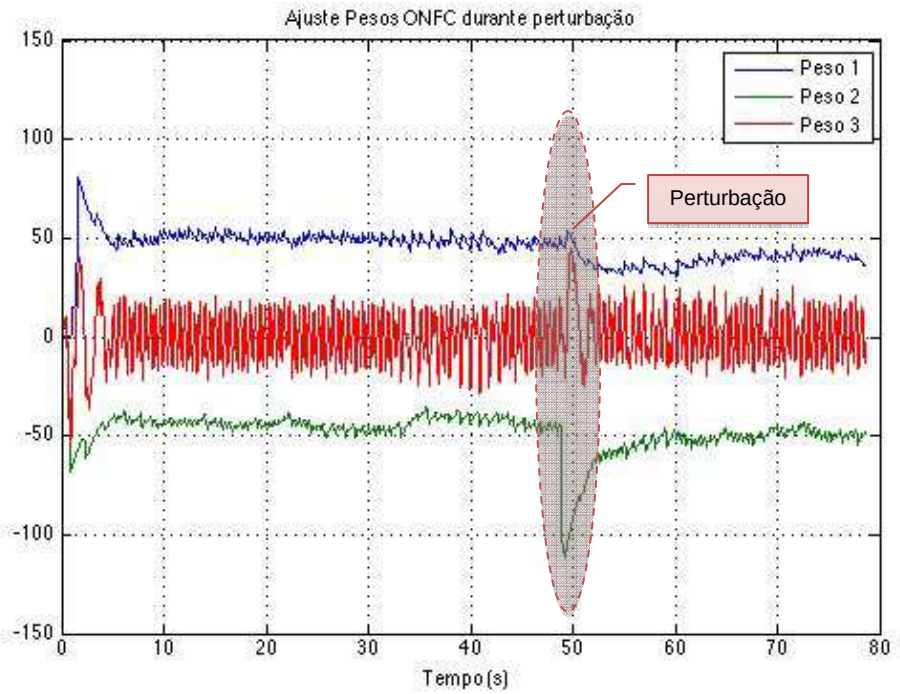


Figura 91 – Atualização em tempo real dos pesos do ONFC durante rejeição à pequena perturbação externa.

6.3.2 Controle de Posição em terreno plano

Na figura 92 observa-se a resposta do deslocamento linear dos três controladores estudados PID, PIDNN e ONFC, na operação de controle de posição em terreno plano. Todos os controladores alcançaram a posição desejada de 50 cm, no teste implementado. O tempo para alcançar o regime estacionário foi de aproximadamente 30 segundos, em todos controladores. Em relação aos demais controladores, o controlador ONFC exibiu inicialmente maior amplitude de excursão em torno do ponto de referência desejado durante operação em regime estacionário. No entanto, conforme os pesos foram sendo ajustados ao longo do tempo a variabilidade no deslocamento linear foi reduzida.

O deslocamento angular dos três controladores também ficou limitada na faixa de ±1^{o} grau (figura 93) indicando a estabilidade do equilíbrio do robô.

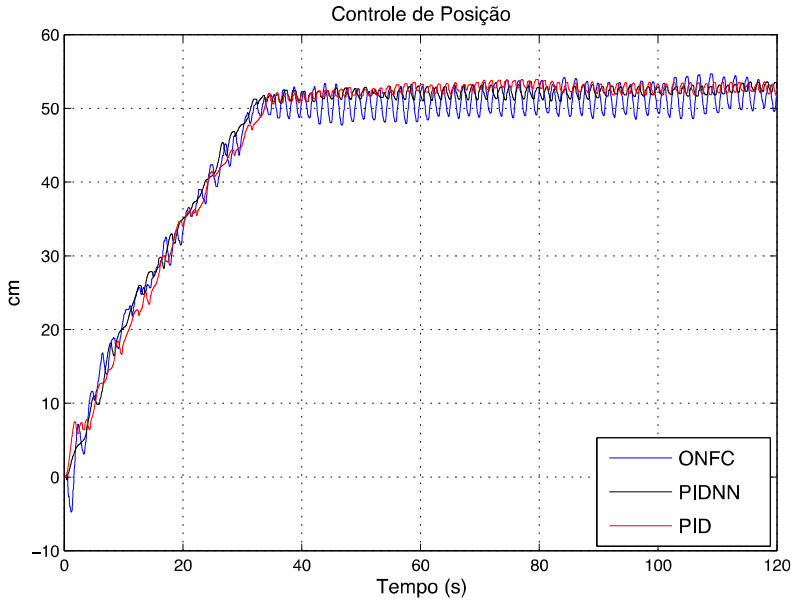


Figura 92 – Controle de Posição ONFC, PIDNN e PID: deslocamento linear.

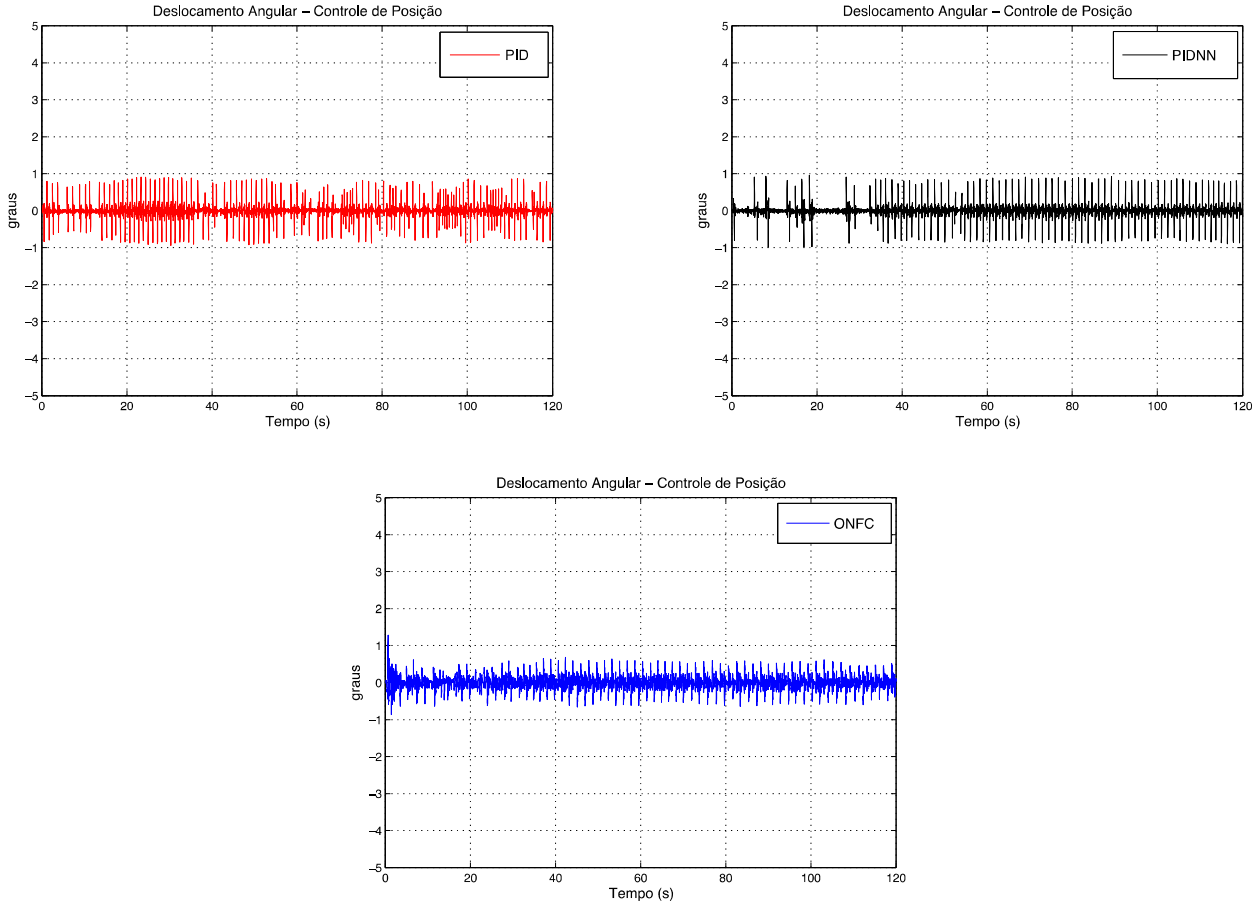


Figura 93 – Controle de Posição ONFC, PIDNN e PID: deslocamento angular.

Para fins comparativos, a tabela 13 exibe os ganhos sintonizados dos controladores PIDNN, com sintonia *off-line* e *online*, e PID. Os valores de ganhos obtidos garantiram a estabilidade do auto-equilibrante em todas condições de operação.

Tabela 13 – Comparativo entre as sintonias dos ganhos dos controladores.

Ganhos do controlador	PIDNN – *off-line*	PIDNN - *online*	PID
k_p	0,0364	0,0360	0,0336
k_i	0,1998	0,1998	0,2688
k_d	0,0150	0,000358	0,000504

6.4 Estudo de caso 2

Para avaliação de desempenho dos algoritmos inteligentes propostos foram realizados testes do auto-equilibrante em ambientes com acréscimo de complexidade: terrenos irregulares e obstáculos no meio do percurso. Esses elementos adicionais no ambiente exigem maior capacidade na resposta dos atuadores, principalmente com relação a sua adaptabilidade. A resposta do controle PID, nessa condição, servirá como referência para avaliação da melhor performance entre as técnicas de controle clássica e inteligente.

Algumas propriedades do terreno podem afetar o modelo matemático do auto-equilibrante, enquanto um obstáculo vertical pode complicar a estratégia de controle.

Uma consideração comum durante o desenvolvimento das equações de um sistema dinâmico é que o terreno é plano, ou seja, com superfície horizontal. Controladores projetados para planos inclinados e/ ou com degraus podem ter sucessos limitados. Por outro lado, a consideração no projeto de elementos do ambiente com maior complexidade aumenta a dificuldade do desenvolvimento do controlador.

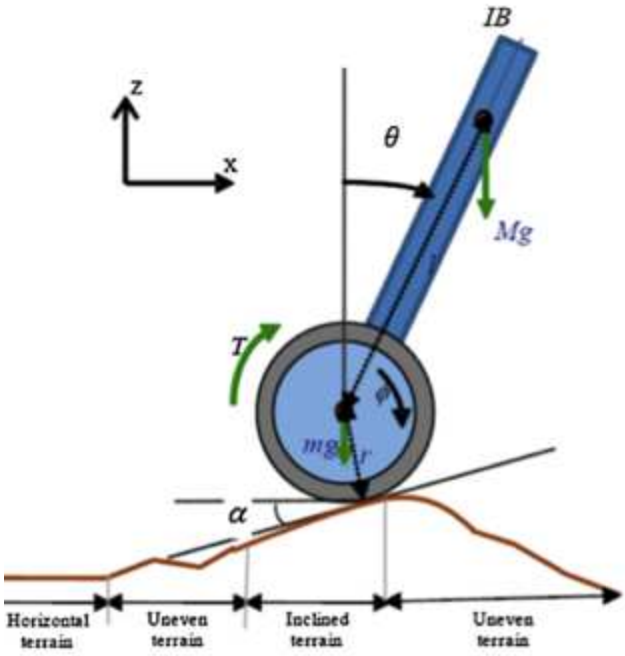


Figura 94 – Exemplo subida em rampa por um auto-equilibrante (CHAN et al., 2013).

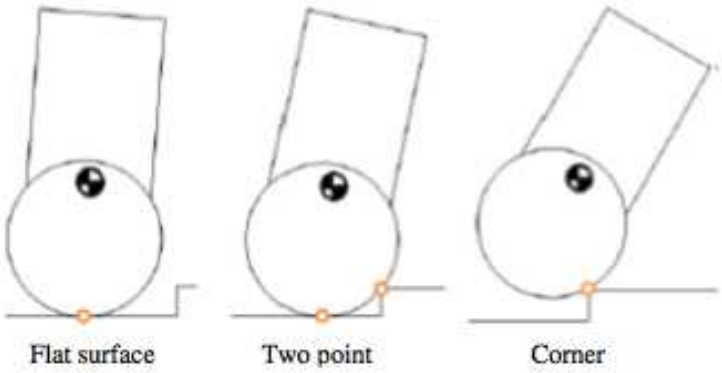


Figura 95 – Exemplo de subida sobre um degrau por um auto-equilibrante (CHAN et al., 2013).

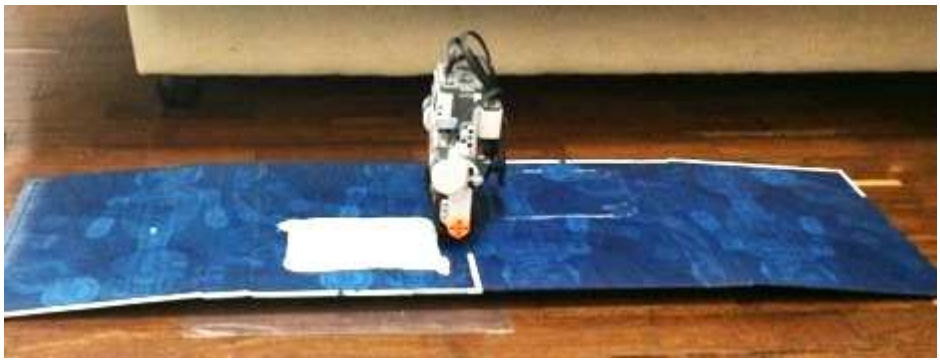
Para análise de robustez entre os controladores estudados foram analisados dois testes em condições do ambiente com maior complexidade: controle em terreno inclinado e controle em terreno com degrau.

6.4.1 Controle em terreno inclinado

O auto-equilibrante foi testado sobre um terreno inclinado. Esse terreno foi representado por uma giga composta por rampas de subida e descida e um trecho sem inclinação, conforme figura 96. As rampas possuíam aproximadamente 10 graus de inclinação. O percurso total da rampa foi de aproximadamente 110 cm, sendo 26 cm o trecho de subida e 26 cm o trecho de descida.

(A)

(B)



(C)

Figura 96 – Operação do auto-equilibrante em terreno inclinado. (A) rampa subida; (B) terreno plano; (C) rampa descida.

Pode-se observar na figura 97 o ajuste dos pesos do controlador ONFC durante operação em terreno inclinado. Nota-se mudança no perfil de ajuste dos pesos durante o trecho de subida e descida do NXT, evidenciando a capacidade do controlador ONFC de adaptação em tempo real.

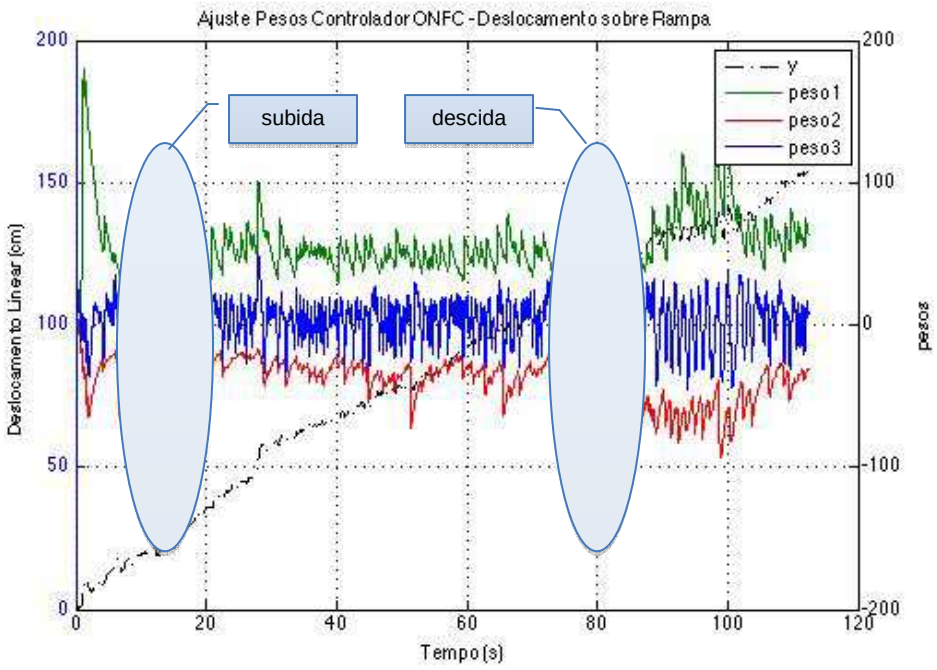


Figura 97 – Ajuste dos pesos do controlador ONFC durante deslocamento em terreno inclinado.

A saída de controle dos controladores PID, PIDNN e ONFC durante operação em terreno inclinado é observada na figura 98. Observa-se que o PIDNN obteve melhor desempenho entre os demais controladores, por apresentar menor variação na saída de controle e menor variação no deslocamento angular durante operação, conforme figura 99.

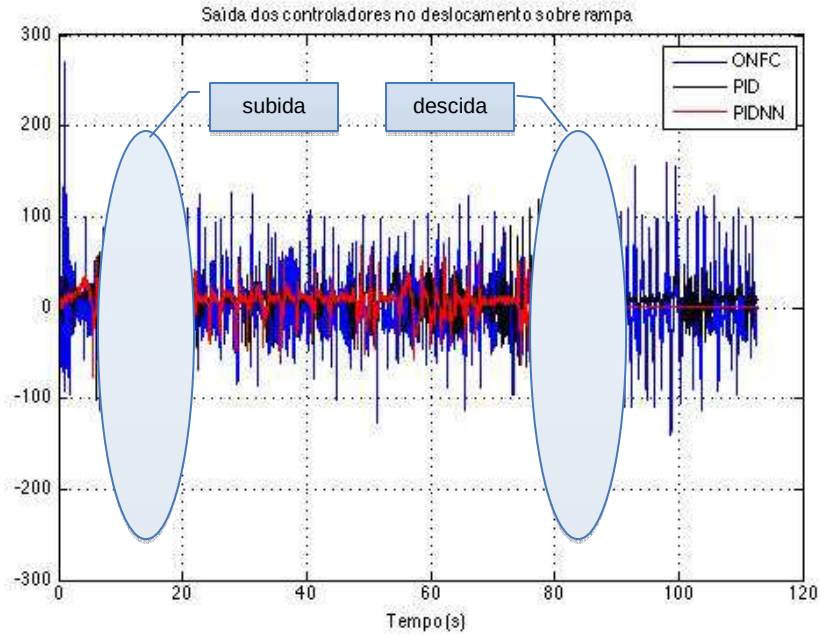


Figura 98 – Saída de controle dos controladores PID, PIDNN e ONFC durante operação em terreno inclinado.

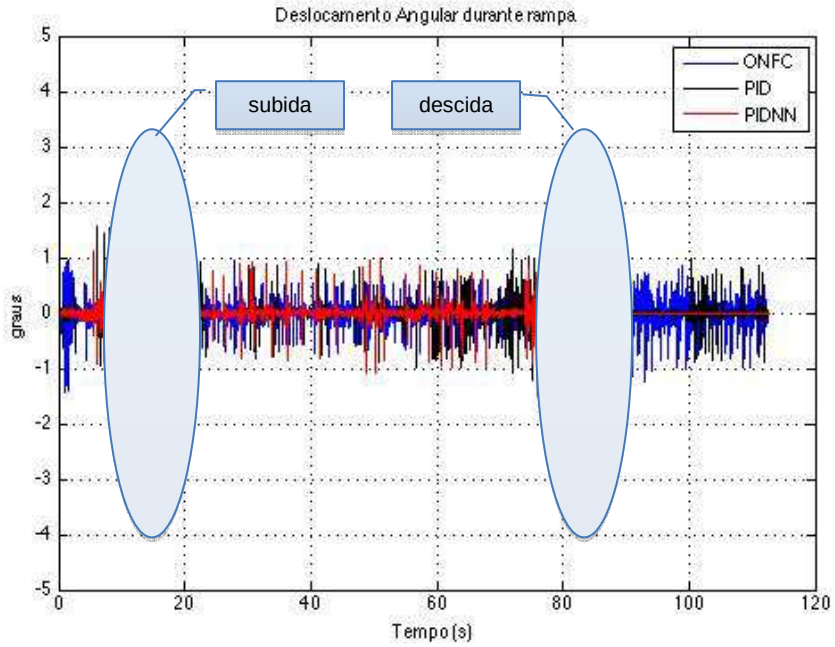


Figura 99 – Deslocamento angular dos controladores PID, PIDNN e ONFC durante operação em terreno inclinado.

Um ponto notável pode ser observado na resposta do deslocamento linear do controlador ONFC, conforme figura 100. O percurso realizado pelo controle

ONFC foi menor em relação aos demais controladores indicando um caminho mais otimizado. Este resultado deve-se à maior velocidade do controlador ONFC na correção do erro da malha de controle.

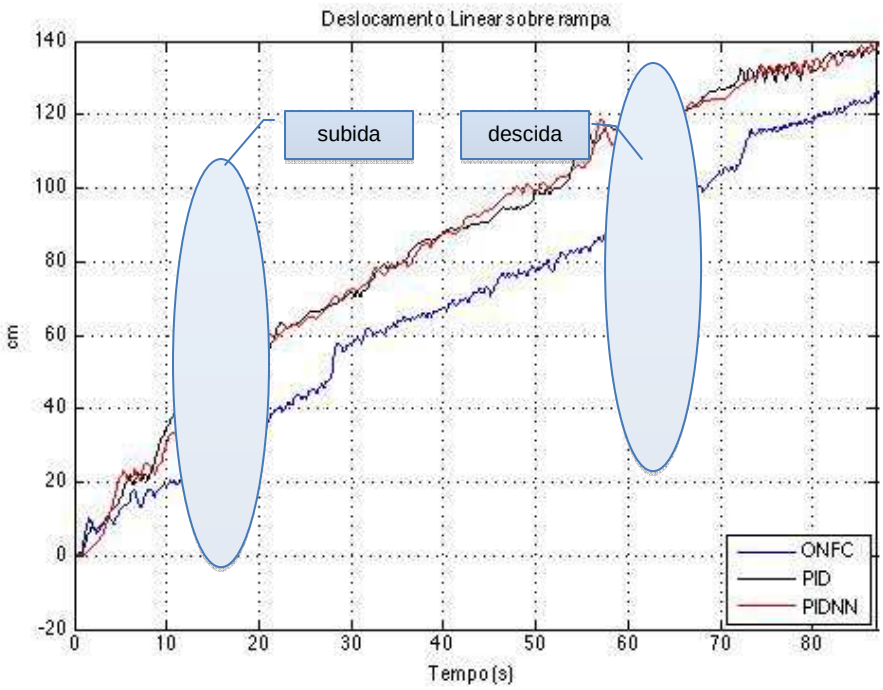


Figura 100 – Deslocamento linear dos controladores PID, PIDNN e ONFC durante operação em terreno inclinado.

6.4.2 Controle em terreno com degrau

Outro teste foi realizado para análise da resposta na subida sobre um degrau pelo auto-equilibrante, conforme figura 101. Este teste reproduz a mudança abrupta da altura de um terreno. Na figura 101, observa-se o momento de subida do auto-equilibrante sobre um degrau. A altura de teste foi de 4 mm.

Como o perfil dinâmico de resposta do controlador PIDNN é parecido com o controle PID, as curvas do PIDNN serão omitidas nas análises seguintes.

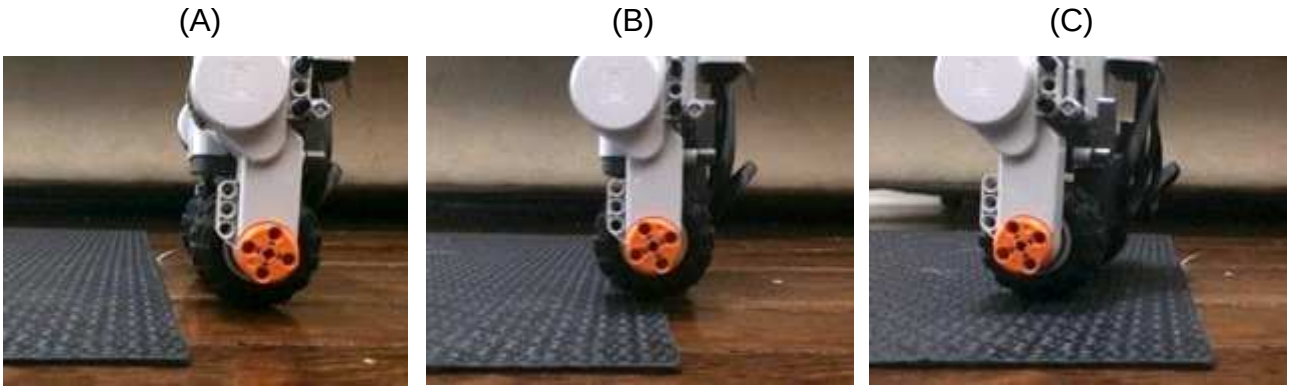


Figura 101 – Subida sobre um degrau pelo auto-equilibrante.

Observou-se que a resposta dinâmica do controlador ONFC foi superior ao controlador PID. O controlador PID apresentou elevadas variações de amplitude durante a subida sobre um degrau, conforme figura 102. Esta resposta indica que o controlador PID possui uma resposta mais lenta que o controle ONFC nesta condição. O ajuste dos pesos durante a subida sobre o degrau é apresentado na figura 103. A partir dos primeiros 17 segundos observa-se a rápida atualização dos pesos no momento de subida ao degrau.

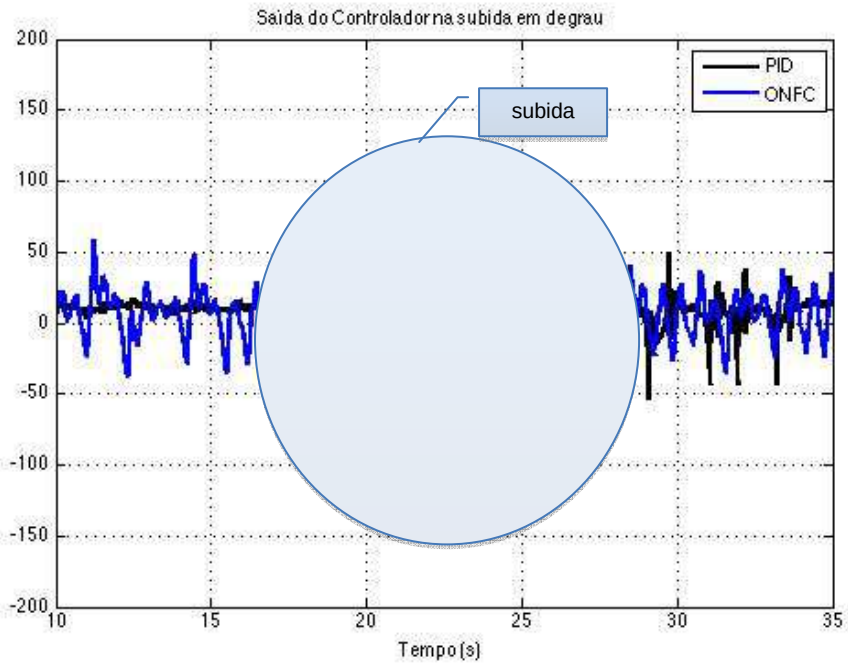


Figura 102 – Saída dos controladores durante subida (t=17s) sobre um degrau.

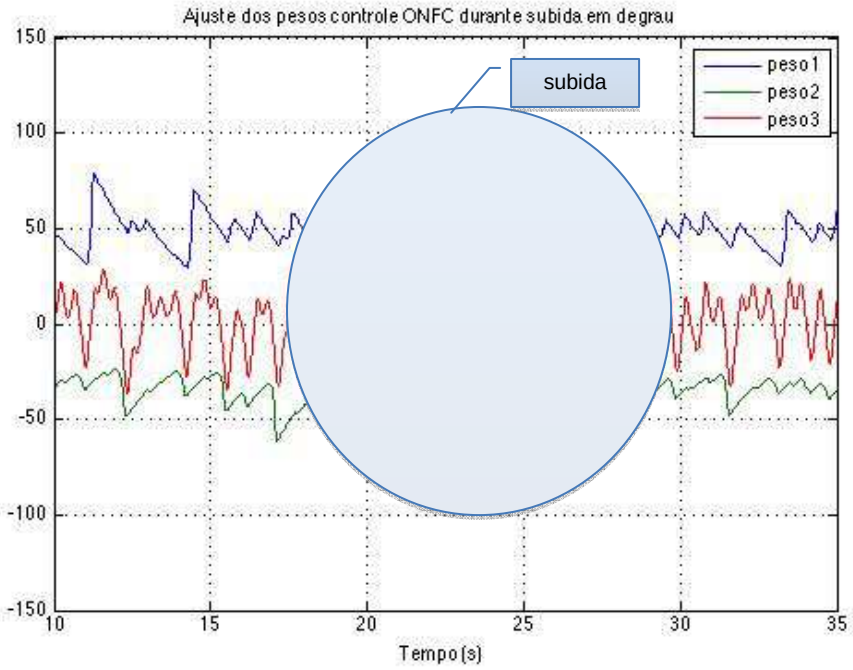


Figura 103 – Ajuste dos pesos do controlador ONFC durante subida sobre um degrau.

Outro fato ainda mais importante pode ser verificado na figura 104. O percurso total desenvolvido pelo NXT durante subida em degrau é inferior ao percurso total. Este fato indica que a resposta do controle ONFC foi mais otimizada e mais rápida em relação ao controle PID.

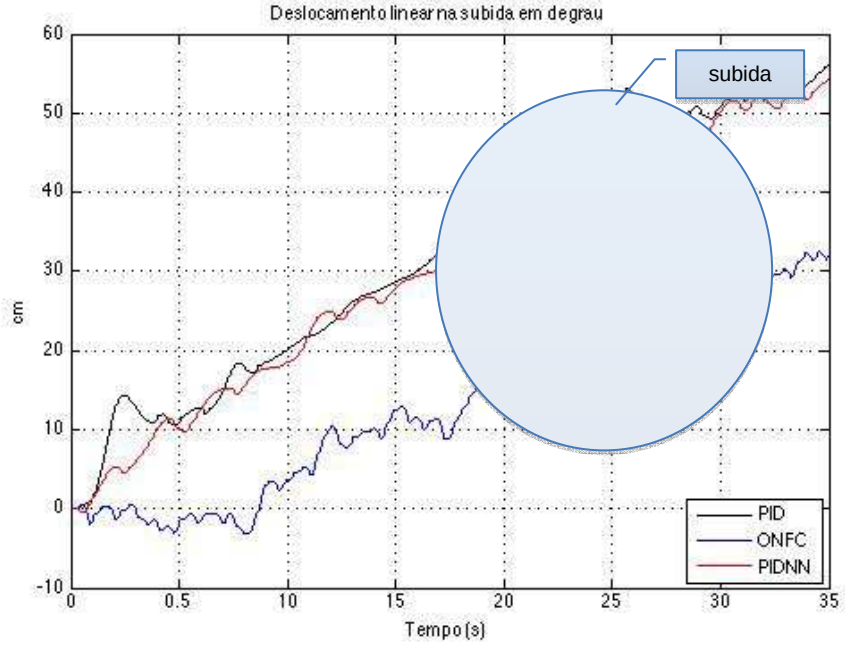


Figura 104 – Deslocamento linear durante subida sobre um degrau.

6.5 Discussão dos resultados

- **Contribuição deste trabalho**

Como mencionado anteriormente, o controle ONFC deste trabalho aplicado no robô auto-equilibrante de duas rodas pertence à classe de controladores neuro-*fuzzy* adaptativos. O controlador ONFC desenvolvido obteve resultados competitivos, inclusive quando comparado com os que foram obtidos por (Su et al., 2010), que foi considerado no artigo (CHAN et al., 2013) como sendo o estado da arte (vide figura 105).

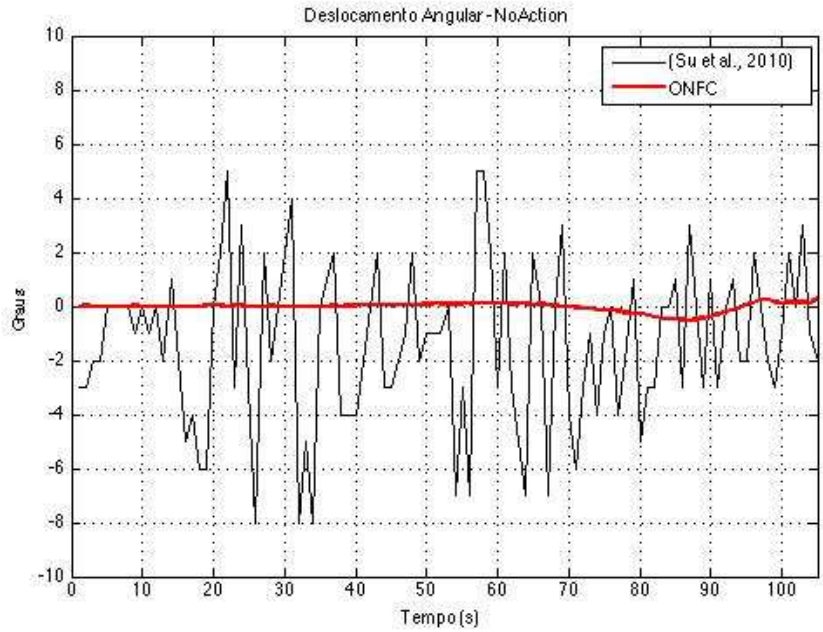


Figura 105 – Deslocamento angular controle ONFC e controle *neuro-fuzzy* (Su et al., 2010).

(Su et al. 2010) também propõem um método *neuro-fuzzy* para controle de um auto-equilibrante. Seus resultados foram obtidos de testes experimentais. Outro ponto importante é que o robô de Su (2010) conta com dois sensores inerciais: um giroscópio e um acelerômetro. Nosso projeto mostra vantagem, visto que a estabilidade do auto-equilibrante foi melhor mesmo utilizando um único sensor inercial e mesmo tendo uma construção mecânica mais instável: roda de pequeno diâmetro e centro de gravidade mais distante do chão.

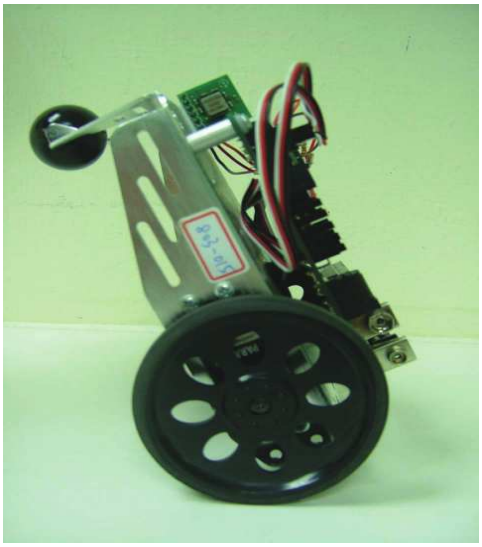


Figura 106 – Robô auto-equilibrante de duas rodas (Su et al., 2010).

- **Limitação a rejeição de perturbações externas**

O auto-equilibrante não é resistente a perturbações externas enquanto em movimento. Se a força for aplicada na direção do movimento, o robô pode cair. Devido os motores funcionarem próximos da velocidade máxima, eles podem não responder tão rápido para compensar o aumento do ângulo. Este problema não pode ser evitado a menos que o auto-equilibrante esteja operando em baixa velocidade.

- **Condições iniciais**

Observa-se uma considerável influência das condições iniciais (C.I.) na resposta do auto-equilibrante, conforme figuras 107 e 108. Diferentes condições iniciais foram testadas no NXT e observa-se que acima de ±5 graus, o robô cai rapidamente após início da operação.

C.I. = 80^o	C.I. = 85^o	C.I. = 90^o	C.I. = 95^o	C.I. = 100^o
				

Figura 107 – Desempenho do NXT para diferentes condições iniciais.

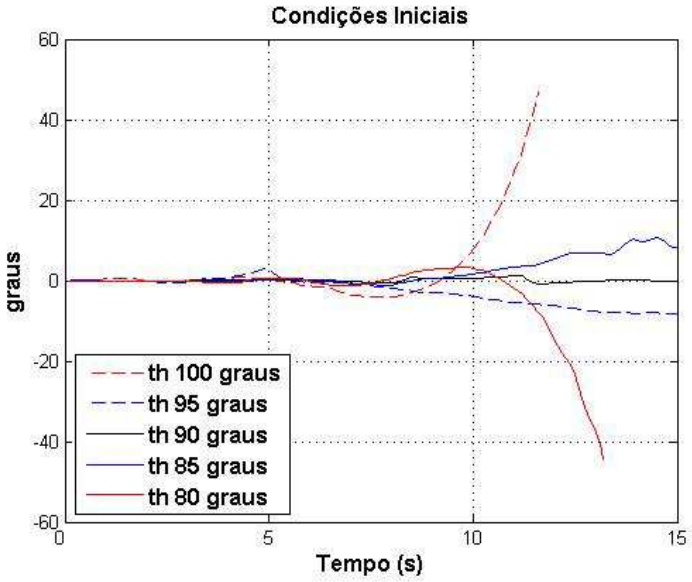


Figura 108 – Resposta do deslocamento angular para diferentes condições iniciais.

- **Comunicação *bluetooth***

Foi implementada a comunicação *bluetooth* entre o NXT e o RobotC de modo a evitar a interferência do cabo de comunicação, entre computador e robô, na desempenho do auto-equilibrante (figura 109).

Figura 109 –Interferência do cabo de comunicação durante operação do NXT.

7 CONCLUSÕES

Neste trabalho foi estudado o controle de equilíbrio e posição de um robô auto-equilibrante de duas rodas. O interesse particular nesta aplicação vem da sua estrutura e da riqueza de sua dinâmica que o torna pertencente à classe de sistemas não lineares, multivaríaveis, instáveis, de alta ordem, fortemente acoplados e subatuados. Por se tratar de um problema complexo e não trivial, houve interesse em avaliar controladores inteligentes especialmente desenvolvidos.

Inicialmente foi desenvolvido um controle clássico do tipo PID para ser comparado com a implementação de outros dois tipos de controladores inteligentes. O primeiro controlador inteligente proposto foi um controlador *neuro-fuzzy* e em seguida um controlador por redes neurais artificiais.

O uso da regra de Ziegler-Nichols se mostrou inadequado em relação à complexidade do sistema estudado, para a sintonia do controlador do tipo PID. Enquanto o controle clássico PID exige a sintonia e ajuste de três parâmetros para o controle de equilíbrio, a proposta de um método de controle inteligente minimiza o esforço para o ajuste desses parâmetros evidenciando a versatilidade e simplicidade do projeto de controladores inteligentes aplicados a plantas complexas.

Outro importante ponto discutido neste trabalho é a questão da sintonia dos parâmetros associados ao controle ONFC. Neste trabalho, apresenta-se a sintonia do universo de discurso das funções de pertinência, a taxa de aprendizado e o termo de regularização. Este termo mostrou-se como um importante fator para manter a relação adequada entre o crescimento dos pesos e eliminação do ganho excessivo.

O desempenho do algoritmo de ajuste dos pesos do controle PIDNN também foi avaliado. Apesar da implementação apenas por simulação, o método *on-line* pelo modelo de referência da planta revelou-se promissor para manter a relação entre a sintonia dos pesos e a redução do erro de controle do sistema.

O ambiente de simulação desenvolvido demonstrou ser uma ferramenta valiosa de estudo para o controle do robô auto-equilibrante de duas rodas. Cabe mencionar que a consideração das características do gyro (*bias* e *deriva*) no modelo simulado foi significativa para a validação com a resposta real do NXT em operação.

O tipo de sensor empregado, de baixo custo e com baixa acurácia, causaram uma pequena perda de desempenho no robô auto-equilibrante. Durante o regime estacionário do controle de equilíbrio observa-se o deslocamento angular oscilante em torno de $\pm 1^{o}$ grau. Apesar da compensação do problema de distorção (*bias*) do gyro ter sido implementada no código, esta calibração não é exata podendo haver ainda uma pequena quantidade de *bias* mesmo depois de zerar o giroscópio.

Foram analisados dois estudos de casos visando a comparar o desempenho dos controladores desenvolvidos. O primeiro caso avaliou o controle de equilíbrio e posição do robô auto-equilibrante de duas rodas sobre um terreno plano, visando a observar o desempenho intrínseco do sistema sob ausência de fatores externos. O segundo caso estudou o controle de equilíbrio e posição do robô em terrenos irregulares, visando a investigar a resposta do sistema sob influência das condições adversas em seu ambiente.

A estabilidade dos controladores na operação de auto-equilíbrio foi alcançada com sucesso em todos os casos, chegando a atingir a faixa de erro do sensor gyro utilizado em ±1 grau. O controle de posição também obteve resultado satisfatório em todos os controladores. No entanto, devido ao erro do sensor inercial e das condições iniciais de operação verifica-se o deslocamento linear em todos os controladores mesmo na condição de equilíbrio estacionário.

Os algoritmos inteligentes contemplam, como mostrado neste trabalho, simplicidade, robustez, versatilidade e eficiência. Sua implementação no robô auto-equilibrante de duas rodas apresentou bons resultados experimentais e por simulação, sendo que suas propriedades adaptativas permitiram manter este desempenho mesmo sob alterações das condições operacionais e da planta que se pretendia controlar.

7.1 Recomendações de trabalhos futuros

Este trabalho apresenta diversas possíveis direções futuras que podem ser seguidas. A primeira extensão que poderia ser realizada está na implementação experimental do controle PID neural com ajuste *on-line* no robô auto-equilibrante de duas rodas.

Outro ponto interessante, seria avaliar o desempenho dos controladores inteligentes desenvolvidos neste trabalho aplicados em outros tipos de robôs auto-equilibrantes. O *ballbot* é um robô que se equilibra em torno de uma única bola esférica (Lauwers et al., 2006). Outro exemplo é o uniciclo *airwheel self-balancing unicycle* desenvolvido pela empresa *AirWheel* que representa um dos mais modernos transportes desenvolvidos cuja tecnologia envolve teoria de controle de ação aeroespacial, algoritmo *fuzzy*, e sensor inercial.

Os ambientes de simulação desenvolvidos constituem numa valiosa ferramenta para novos estudos em controle de sistemas auto-equilibrantes. Especificamente o controle do robô auto-equilibrante de duas rodas pode ser estendido para o controle de rotação e desvio de obstáculos. Tais ambientes poderão ser usados em pesquisas e no ensino da pós-graduação de Engenharia, bem como encorajar estudantes e profissionais a explorar a utilização de técnicas inteligentes para alcançar resultados melhores em suas aplicações.

Da mesma forma, novos testes experimentais podem ser realizados explorando o controle de rotação e o desvio de obstáculos utilizando o sensor de ultrassom.

Com relação ao sensoriamento, sensores mais precisos, ou mesmo a combinação de mais sensores, como o acelerômetro, poderiam ser utilizados para investigar um desempenho melhor no sistema.

Um outro ponto importante é a implementação dos mesmos controladores desenvolvidos em mais de um kit LEGO, permitindo analisar sua repetibilidade e desempenho em mais de um sistema. A cooperação robótica para realização de tarefas também poderia ser empregada estendendo as funções dos controladores inteligentes desenvolvidos em um cenário com mais de um auto-equilibrante.

Testes em ambientes mais complexos também poderiam ser implementados para analisar o desempenho do auto-equilibrante a fatores externos.

REFERÊNCIAS

ACOSTA, J. Á. Furuta's pendulum: A conservative nonlinear model for theory and practice. Mathematical Problems in Engineering, vol. 2010.

AMARAL, J. F. M. Síntese de Sistemas *Fuzzy* por Computação Evolucionária. Tese de Doutorado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil, 2003.

AMBROSE, R. O. & SAVELY, R. T. & GOZA, S. M & STRAWSER, P. & DIFTLER, M. A. & SPAIN, I. & RADFORD, N. Mobile manipulation using NASA's robonaut. IEEE ICRA, 2104-2109. 2004.

ANDREV, F. & AUCKLY, D. & GOSAVI, S. & KAPITANSKI, L. & KELKAR, A. & WHITE, W. Matching, linear systems, and the ball and beam. Automática, vol. 38, n°12, pp. 2147- 2152. 2002

AOYAMA, T. & SEKIYAMA, K. & HASEGAWA, Y. & FUKUDA, T. Passive dynamic autonomous control for the multi-locomotion robot. In A.C. Pina Filho (Eds.), Biped Robots, Rijeka: InTech, pp.115-128. 2011.

ÅSTRÖM, K. J. & FURUTA, K. Swinging up a pendulum by energy control. Automática, vol. 36, pp. 287-295. 2000.

ÅSTRÖM, K. J. & KLEIN, R. E. & LENNARTSSON, A. Bicycle dynamics and control. IEEE Control Systems Magazine, vol. 25, n°4, pp. 26-47. 2005.

BAGEANT, M. R. Balancing a Two-Wheeled Segway Robot. Massachusetts Institute of Technology. 2011.

BIGGE, B. & HARVEY, I. R. Evolvable Mechanics: Hardware tools for evolutionary robotics. 2008.

BLOCK, D. J. & ÅSTRÖM, K. J. & SPONG, M.W. The Reaction Wheel Pendulum. San Rafael: Morgan and Claypool. 2007.

BONGARD, J. C. & PAUL, C. Making evolution an offer it can't refuse: Morphology and the extradimensional bypass. In Proc. of 6th European Conf. on Artificial Life, Prague, CZ, pp. 401–412. 2001.

BONGARD, J. C. & PFEIFER, R. Evolving complete agents using artificial ontogeny. In Morpho-functional Machines: The New Species: Designing Embodied Intelligence, R. Hara and R. Pfeifer, Eds.: Springer-Verlag, pp. 237-258. 2003

BORTOFF, S. A. Robust swing up control for a rotational double pendulum. In: Proceedings of the 13th World Congress of IFAC, San Francisco, pp. 413-419. 1996.

BOUBAKER, O. The Inverted Pendulum benchmark in Nonlinear Control Theory: A

Survey. International Journal of Advanced Robotic Systems. 2012.

BULLINARIA, J. A. From biological models to the evolution of robot control systems. Philosophical Transactions of the Royal Society of London A, Vol 361, pp. 2145-2164. 2003.

BULLINARIA, J.A. & RIDDELL, P.M. Neural network control systems that learn to perform appropriately. International Journal of Neural Systems, Vol 11, pp. 79-88. 2001.

CAMINHAS, W. M., TAVARES, H. M. F., GOMIDE, F. A. C., PEREIRA, GUIHERME A. S. Identificação de Sistemas Dinâmicos: Abordagem Baseada em Neurônio Nebuloso. Simpósio Brasileiro de Redes Neurais, Belo Horizonte, MG. Anais do V Simpósio Brasileiro de Redes Neurais, pp. 105-110, 1998.

CAMPBELL, S. A. & CRAWFORD, S. & MORRIS, K. Friction and the inverted pendulum stabilization problem. Journal of Dynamic Systems, Measurement and Control, vol. 130, n°5. 2008.

CARVALHO, M. A. Controlador *neuro-fuzzy* com aprendizado *on-line*: teoria e aplicação na indústria de petróleo. Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, MG, 2010.

CARVALHO, M. A., GOUVÊA, M. R., SILVA, R. G., CAMINHAS, W. M. Uso de controlaores *neuro-fuzzy* baseado em redes NFN para controle de pH. In: Congresso Brasileiro de Automática, 2006, Salvador, BA. Anais do XVI Congresso Brasileiro de Automática, vol. 1. pp. 1-6, 2006.

CASTILLO, O. & AGUILAR, L. T. & Cázarez-Castro, N. R. & Cardenas, S. Systematic design of a stable type-2 *fuzzy* logic controller. Applied Soft Computing, vol. 8, n°3, pp. 1274-1279. 2008.

CASTILLO, O. & MELIN, P. Optimization of type-2 *fuzzy* systems based on bio-inspired methods: A concise review. Information Sciences, vol. 205, pp. 1-19. 2012.

CÁZAREZ-CASTRO, N. R. & AGUILAR, L. T. & CASTILLO, O. Designing type-1 and type-2 *fuzzy* logic controllers via *fuzzy* Lyapunov synthesis for non smooth mechanical systems. Engineering Applications of Artificial Intelligence, vol. 25, n°5, pp. 971-979. 2012.

CÁZAREZ-CASTRO, N. R. & AGUILAR, L. T. & CASTILLO, O. *Fuzzy* logic control with genetic membership function parameters optimization for the output regulation of a servomechanism with nonlinear backlash. Expert Systems with Applications, vol. 37, n°6, pp. 4368-4378. 2010.

CHAN, R. P. M; STOL, K. A.; HALKYARD C. R. Review of modeling and control of two-wheeled robots. Annual reviews in control 37, 89-103, 2013.

CHATURVEDI, K. T., PANDIT, M., SRIVASTAVA, L. Modified neo-fuzzy neuron-based approach for economic and environmental optimal power dispatch Applied

Soft Computing 8, pp. 1428–1438, 2008.

CHIU, C.-H., & PENG, Y.-F. Design and implement of the self-dynamic controller for two-wheel transporter. In IEEE international conference on fuzzy systems (pp. 480–483). 2006.

CHUNG, C. C. & HAUSER, J. Nonlinear control of a swinging pendulum. Automática, vol. 31, n°6, pp. 851-862. 1995.

CLARK, C. M. & MILLS, J. K. Robotic system sensitivity to neural network learning rate: theory, simulation and experiments. International Journal of Robotics Research, Vol 19, pp. 955-968. 2000.

DAVIDOR, Y. Genetic algorithms and robotics: A heuristic strategy for the optimization. World Scientific Publishing, Singapore. 1991.

DEEGAN, P. & THIBODEAU, B. & GRUPEN, R. Designing a self-stabilizing robot for dynamic mobile manipulation. Robotics: Science and Systems - Workshop on Manipulation for Human Environments. 2006.

DELGADO, M. R. B. S. Projeto Automático de Sistemas Nebulosos: Uma abordagem Co-evolutiva. Tese de Doutorado, Faculdade de Engenharia Elétrica e Computação - Unicamp, Campinas, Brasil. 2002.

DUNG, V. B. T. Developing a Self-balancing Robot with Lego NXT Mindstorms. Bachelor's Thesis, Turku University of Applied Sciences. 2010.

EIBEN, A. E. & SMITH, J. E. Introduction to Evolutionary Computing. Springer-Verlag: Berlin, Germany. 2003.

ER, M. J. & LIEW, K. C. Control of Adept One SCARA robot using neural networks. IEEE Transactions on Industrial Electronics, Vol 44, pp. 762-768. 1997.

ER, M. J. & LOW, C. B. & NAH, K. H. & LIM, M. H. & NG, S. Y. Real-time implementation of a dynamic fuzzy neural networks controller for a SCARA. Microprocessors and Microsystems, Vol 26, pp. 449-461. 2002.

ER, M. J. & TAN, T. P. & LOH, S. Y. Control of a mobile robot using generalized dynamic fuzzy neural networks. Microprocessors and Microsystems, Vol 28, pp. 491-498. 2004.

FANTONI, I. & LOZANO, R. & SPONG, M. W. Energy based control of the Pendubot. IEEE Transactions on Automatic Control, vol. 45, n°4, pp. 725-729. 2000.

FERRARI, A. C. K. Controlador PID sintonizado por redes neurais artificiais. Monografia, Curso de Engenharia Elétrica, Universidade Federal do Paraná, Curitiba, 2011.

FURUTA, K. & OCHIAI, T. & ONO, N. Attitude control of a triple inverted pendulum. International Journal of Control, vol. 39, pp. 1351-1365. 1984.

FURUTA, K. & OKUTANI, T. & SONE, H. Computer control of a double inverted

pendulum. Computer and Electrical Engineering, vol. 5, pp. 67-84. 1978.

FURUTA, K. & YAMAKITA, M. & KOBAYASHI, S. Swing-up control of inverted pendulum using pseudo□state feedback. Journal of Systems and Control Engineering vol. 14, pp. 263-269. 1992.

GARCÍA, M. A. P. & MONTIEL, O. & CASTILLO, O. & Sepúlveda, R. & Melin, P. Path Planning for Autonomous Mobile Robot Navigation with Ant Colony Optimization and Fuzzy Cost Function Evaluation. Applied Soft Computing Journal, vol. 9, pp. 1102-1110. 2009.

GEN, M and CHENG, R. Genetic algorithms and engineering optimization. John Wiley: New York, NY. 2000.

GÖÇMEN, A. Design of Two Wheeled Electric Vehicle. Master of Science Thesis, Atilim Universtity. 2012.

GÖÇMEN, A. Design of Two Wheeled Electric Vehicle. Master of ScienceThesis, AtilimUniverstity, 2012.

GOHER, K. M. & TOKHI, M. O. Modeling and Control of a Two Wheeled Machine: A Genetic Algorithm-Based Optimization Approach. Journal of Selected Areas in Robotics and Control (JSRC). 2010.

GOMES, A. C. D. N. Notas de aula. Sistemas Multivariáveis. Programa de Pós Graduação em Controle, Automação e Robótica UFRJ. 2009.

GÓMEZ, G. & EGGENBERGER HOTZ, P. Investigations on the robustness of an evolved learning mechanism for a robot arm. In Proceedings of the 8th International Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands, pp. 818-827. 2004.

GONG, J. Q. & YAO, B. Neural network adaptive robust control of nonlinear systems in semi- strict feedback form. Automática, Vol 37, pp. 1149-1160. 2001.

GORDILLO, F. & ARACIL, J. A new controller for the inverted pendulum on a cart. International Journal of Robust and Nonlinear Control, vol.18, n°17, pp. 1607- 1621. 2008.

GOUVÊA, M. R. Controle Neurofuzzy de Motor de Indução Com Estimação de Parâmetros e Fluxo de Estator. Tese de Doutorado, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, MG, 2005.

GRASSER, F. & D'ARRIGO, A. & COLOMBI, S. & RUFER, A. C. JOE: A Mobile, Inverted Pendulum. IEEE Transactions on Industrial Electronics, vol. 49, no. 1, pp. 107–114. 2002.

GREWAL, M. S. & ANDEWS, A. P. Kalman filtering: Theory and practice. Prentice Hall: Englewood Cliffs, NJ. 1993

HAM. F. M. & KOSTANIC, I. Principles of neurocomputing for science and

engineering. McGraw-Hill: New York, NY. 2001.

JANKOVIC, M. & FONTAINE, D. & KOKOTOVIĆ, P. V. TORA example: Cascade□ and passivity-based control designs. IEEE Transactions on Control Systems Technology, vol. 4, n°3, pp. 292-297. 1996.

JEONG, S. H. & TAKAYUKI, T. Wheeled Inverted Pendulum Type Assistant Robot: Design Concept and Mobile Control. IEEE IROS, 1932-1937. 2007.

JIN, D. Development of a Stable Control System for a Segway. Royal Institute of Technology. 2013.

KIM, S. & JUNG, S. Control experiment of a wheel-driven mobile inverted pendulum using neural network. IEEE Trans. On Control Systems Technology, 16(2), 297-303. 2008.

KIM, Y. & KIM, H. K. & KWAK, Y. K. Dynamic Analysis of a Nonholonomic Two-Wheeled Inverted Pendulum Robot. Journal of Intelligent and Robotic Systems, 44(1), 25-44. 2005.

KIMURA, H. & FUKUOKA, Y. & COHEN, A. H. Adaptive Dynamic Walking of a Quadruped Robot on Natural Ground Based on Biological Concepts. Int. J. of Robotics Research, vol. 26, pp. 475-490. 2007.

LAUWERS, T. B. & KANTOR, G. A. & HOLLIS, R. L. A Dynamically Stable Single Wheeled Mobile Robot with Inverse Mouse-Ball Drive. In Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, FL. 2006.

LEE, C.C. Fuzzy logic in control systems: Fuzzy logic controllers – Parts I, II. IEEE Transactions on Systems, Man, and Cybernetics, Vol 20, pp. 404-435. 1990.

LEE, G. H. & JUNG, S. Line Tracking Control of a Two-Wheeled Mobile Robot Using Visual Feedback. International Journal of Advanced Robotic Systems. 2013.

LEWIS, F. W. & JAGANNATHAN, S. & YESILDIRAK, A. Neural network control of robot manipulators and non-linear systems. Taylor & Francis: Oxford, UK. 1998.

LI, H. & MIAO, Z. & WANG, J. Variable universe adaptive fuzzy control on the quadruple inverted pendulum. Science in China, Series E: Technological Sciences, vol. 45, pp. 213-224. 2002.

LIDSTROM, N. Development of a control system for stabilizing a LEGO Segway model. Research Academy for Young Scientists. 2013.

LIZARRALDE, F. & HSU, L. Controle de Robôs Manipuladores. COPPE/UFRJ. 2006.

LUO, A. C. J. & MIN, F. The chaotic synchronization of a controlled pendulum with a periodically forced, damped Duffing oscillator. Communications in Nonlinear Science and Numerical Simulation, vol. 16, n°12, pp. 4704-4717. 2011.

MACINNES, I. & DI PAOLO, E. A. Crawling out of the simulation: Evolving real robot

morphologies using cheap, reusable modules. In Proc. of 9th Int. Conf. on the Simulation and Synthesis of Living Systems, Boston. 2004.

MATSUMOTO, D. E. & MENDONÇA, M. & ARRUDA, L. V. R. & PAPAGEORGIOU, E. Embebed Dynamic Fuzzy Cognitive Maps Applied To The Control Of An Industrial Mixer. Simpósio Brasileiro Automação Inteligente - SBAI 2013. 2013.

MEHDI, H. & BOUBAKER, O. Impedance controller tuned by particle swarm optimization for robotic arms. International Journal of Advanced Robotic Systems, vol.8, n°5, pp. 93-103. 2011.

MENDEL, J. M. Fuzzy Logic Systems for Engineering: A Tutorial. Proceedings of the IEEE v. 83, n. 3, pp. 345-377. 1995.

MILLER, W. T. & SUTTON, R. S. & WERBOS, P. J. Neural networks for control. MIT Press: Cambridge, MA. 1990.

MOREIRA, M. V. B. Notas de aula. Sistemas Lineares. Programa de Pós Graduação em Controle, Automação e Robótica UFRJ. 2008.

MORIARTY, D. E. & MIIKKULAINEN, R. Evolving obstacle avoidance behavior in a robot arm. In Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior. Cambridge, MA: MIT Press, pp. 468-475. 1996.

NASIR, A. N. K., AHMAD, M. A., GHAZALI, R., & PAKHERI, N. S. Performance comparison between fuzzy logic controller (FLC) and PID controller for a highly nonlinear two-wheels balancing robot. In First international conference on informatics and computational intelligence (ICI), 2011 (pp. 176–181). 2011.

NGUYEN, H. G. & MORRELL, J. & MULLENS, K. & BURMEISTER, A. & MILES, S. & FARRINGTON, N. & THOMAS, K. & GAGE, D. W. Segway Robotic Mobility Platform. In SPIE Proc. 5609: Mobile Robots XVII, Philadelphia, PA. 2004.

NOH, J. S. & LEE, G. H. & JUNG, S. Position control of a mobile inverted pendulum system using radial basis function network. IEEE WCCI, 371-377. 2008.

NÜRNBERGER, A. & NAUCK, D. & KRUSE, R. Neuro-fuzzy control based on the NEFCON-model: recent developments. Soft Computing Abstract Volume 2 Issue 4 pp 168-182. 1999.

OGATA, K. Engenharia de Controle Moderno. 4ª Edição Pearson Prentice Hall, pp. 72-79. 2003.

OOI, R. C. Balancing a Two-Wheeled Autonomous Robot. The University of Western Australia, 2003.

OOI, R. C. Balancing a Two-Wheeled Autonomous Robot. The University of Western Australia. 2003.

ORTEGA, R. & SPONG, M. W. & GÓMEZ-ESTERN, F. & BLANKENSTEIN, G. Stabilization of a class of under-actuated mechanical systems via interconnection and damping assignment. IEEE Transactions on Automatic Control, vol. 47, n°8, pp.

1218-1233. 2002.

PATHAK, K. & FRANCH, J. & AGRAWAL, S. Velocity and position control of a wheeled inverted pendulum by partial feedback linearization. IEEE Trans. on Robotics, 21, 505-513. 2005.

PELOSO, E. F. Desenvolvimento de um Robô Auto-equilibrante de Duas Rodas. Curso de Pós-Graduação em Engenharia Mecatrônica UERJ. 2009.

PEZESHKI, S. & BADALKHANI, S. & JAVADI, A. Performance Analysis of a Neuro-PID Controller Applied to a Robot Manipulator. International Journal of Advanced Robotic Systems, vol. 9, pp. 163-173. 2012.

PIRES A. V. Controladores baseados em técnicas de inteligência computacional: Análise, projeto e aplicação. Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, MG, 2007.

RUSSEL, S. & NORVIG, P. Artificial Intelligence. Elsevier, Rio de Janeiro. 2003.

SAKTHIVEL, G. & RAJASEKAR, S. Diffusion dynamics near critical bifurcations in a nonlinearly damped pendulum system. Communications in Nonlinear Science and Numerical Simulation, vol.17, n°3, pp.1303-1311. 2012.

SANTOS, S. Estudo sobre Plataformas para Desenvolvimento Evolucionário de Circuitos Eletrônicos. Projeto de Graduação UERJ. 2007.

SASAKI, K. & MURAKAMI, T. Pushing operation by two-wheel inverted mobile manipulator. IEEE Workshop on Advanced Motion Control, 33-37. 2008.

SCIAVICCO, L. & SICILIANO, B. Modelling and Control of Robot Manipulators. 2nd ed., Springer-Verlag. 2000.

SEKIGUCHI, A. & KAMETA, K. & Yuichi, Y. & Nenchev, D. Biped walk based on vertical pivot motion of linear inverted pendulum. IEEE Conf. on Advanced Motion Control, 1-6. 2007.

SHEN, J. & SANYAL, A. K. & CHATURVEDI, N. A. & BERNSTEIN, D. & MCCLAMROCH, H. Dynamics and control of a 3D pendulum. In: Proceedings of the 43rd IEEE Conference on Decision and Control, Nassau, vol. 1, pp. 323-328. 2004.

SIM, K. & LEE, D. W. & ZHANG, B. T. Behavior Evolution of Autonomous Mobile Robot (AMR) using Genetic Programming based on Evolvable Hardware. International Journal of Fuzzy Logic and Intelligent Systems, Vol. 2, no 1, pp. 20-25. 2002.

SICILIANO, B., & KHATIB, O. Springer Handbook of Robotics. Springer. 2008.

SLOTINE, J. J. E. & LI, W. Applied Nonlinear Control. Prentice Hall. 1991.

SMITH, T. M. C. The evolvability of artificial neural networks for robot control.

Submitted for the degree of D. Phil. University of Sussex. 2002.

SPADINI L. M. et al. Desenvolvimento de controle neurofuzzy para plantas não lineares: aplicação em tanque acoplados. Simpósio Brasileiro de Automação Inteligente, 2013, Fortaleza, CE. Anais do XI Simpósio Brasileiro de Automação Inteligente. 2013.

SPADINI, L. M. & SILVA, P. E. & CAMPOS, L. F. F. & ARAÚJO, C. J. F. & NIED, A. Desenvolvimento de Controle Neurofuzzy para Plantas Não Lineares: Aplicação em Tanques Acoplados. Simpósio Brasileiro Automação Inteligente - SBAI 2013. 2013.

SPONG, M. W. The swing up control problem for the acrobat. IEEE Control Systems Magazine, vol. 15, pp. 72-79. 1995.

STILMAN, M. & OLSON, J. & GLOSS, W. Golem Krang: Dynamically stable humanoid robot for mobile manipulation. In IEEE Int'l Conf. on Robotics and Automation, pp. 3304–3309. 2010.

SU, K.-H., & CHEN, Y.-Y. Balance control for two-wheeled robot via neural- fuzzy technique. In Proceedings of SICE annual conference 2010 (pp. 2838–2842). 2010.

SUNDIN, C. THORSTENSSON F., Autonomous Balancing Robot. Master of ScienceThesis, Chalmers University of Technology, Göteborg, Sweden, 2012.

TAKAHASHI, Y. & OGAWA, S. & MACHIDA, S. Step climbing using power assist wheel chair robot with inverse pendulum control. In Proc. IEEE Intl. Conf. on Robotics and Automation, pp. 1360–65. 2000.

TSAI, C.-C., HUANG, H.-C., & LIN, S.-C. Adaptive Neural Network Control of a Self-Balancing Two-Wheeled Scooter. 2010.

TSAI, M. & SHEN, B. H. Synchronization control of parallel dual inverted pendulums driven by linear servomotors. IET Control Theory and Applications, vol. 1, pp. 320-327. 2007.

WALTER, J. A. & SCHULTEN, K. J. Implementation of self-organizing neural networks for visual- motor control of an industrial robot. IEEE Transactions on Neural Networks, Vol 4, pp. 86-95. 1993.

WILLIAMSON, M. M. Control of Rhythmic Arm Movements. Neural Networks Special Issue on Neural Control of Movement, vol. 11, pp. 1379-1394. 1998.

Witzand S. J. Coordinated LEGO Segways. The University of New South Wales. Australian Defence Force Academy. 2009.

WU, J., & ZHANG, W. Design of fuzzy logic controller for two-wheeled self-balancing robot. In 6th International forum on strategic technology (IFOST), 2011 (pp. 1266–1270). 2011b.

YAMAKAWA, T. et al. A Neo Fuzzy Neuron and its Applications to System Identification and Predictions to System Behavior. Proc. Of the 2nd IIZUKA, Iizuka-

Japan, pp. 477-483, 1992.

YAMAMOTO, Y. NXTway-GS Model-Based Design - Control of self-balancing two-wheeled robot built with LEGO Mindstorms NXT, ANNEXE C – Modèle mathématique du robot lego NXT. 2010.

YOSHIDA, K. Swing-up control of an inverted pendulum by energy-based methods. In: Proceedings of the American Control Conference, San Diego, pp. 4045-4047. 1999.

YOUNIS, W. & ABDELATI, M. Design and implementation of an experimental Segway model. Proceedings of AIP Conference Proceedings, vol. 1107, pp. 350-354. 2009.

ZHAO, J. & SPONG, M. W. Hybrid control for global stabilization of the cart-pendulum system. Automática, vol. 37, n°12, pp. 1941-1951. 2001.

ZIEGLER JG, NICHOLS NB. Optimum Settings for Automatic Controllers. [Online] 1942. Available from: http://www2.eie.ucr.ac.cr/\~valfaro/docs/Ziegler\%26Nichols.pdf [Accessed: July 09 2013]

Anexo- código programação

Código fonte controle ONFC:

```
/* Controle ONFC

Author: Sender Rocha dos Santos

*/

int steering = 0;
int acceleration = 20;
int speed = 0;
bool starting_balancing_task = true;

float gn_dth_dt,gn_th,gn_y,gn_dy_dt,kp,ki,kd,mean_reading,gear_down_ratio,dt,e_m
float th = 0,y = 0;
float u = 0;

#ifdef HTTechnic_Gyro
int calibrate_hitechnic();
#endif

task balancing()
{
  gear_down_ratio = 1;
  dt = 0.010;

  gn_dth_dt = 0.23;
  gn_th = 25;
  gn_y = 272.8;
  gn_dy_dt = 24.6;

  ////////////////////////////
  //END ADVANCED USER CONTROL
  ////////////////////////////

  //MOTOR SETUP
  nMotorPIDSpeedCtrl[motorB] = mtrNoReg;
  nMotorPIDSpeedCtrl[motorC] = mtrNoReg;
  nMotorEncoder[motorC] = 0;
  nMotorEncoder[motorB] = 0;

  //SENSOR SETUP
  int nSensorsDefined = 0;
  #ifdef HTTechnic_Gyro
      SensorType[Gyro] = sensorLightInactive;
      mean_reading = calibrate_hitechnic();
      nSensorsDefined++;
  #endif

  //MATH CONSTANTS
  const float radius = your_wheel_diameter/1000;
  const float degtorad = PI/180;

  //SETUP VARIABLES FOR CALCULATIONS
  float   dth_dt = 0,   // Angular velocity of robot (degree/sec)
          th_prev =0;
  float e = 0,//Error             // Sum of four states to be kept zero: th, dth
  float pid = 0,
        pid_prev = 0;
  float dy_dt = 0,//dy/dt            // Measured motor velocity (degrees/sec)
        v = 0,//velocity          // Desired motor velocity (degrees/sec)
```

```
      a,gama,
    p1=0,
    p2=0,
    p3=0,
    p1_prev=0,p11_prev=0,
    p2_prev=0,p21_prev=0,
    p3_prev=0,

    w1_prev=0,
    w2_prev=0,
    w3_prev=0,
    w1=0,
    w2=0,
    w3=0,
      y_ref = 0;//reference pos // Desired motor position (degrees)
int  motorpower = 0,              // Power ultimately applied to motors
    last_steering = 0,           // Steering value in previous cycle
    straight = 0,                // Average motor position for synchronizing
    d_pwr = 0;
                // Change in power required for synchronizing
const int n_max = 7;             // Number of measurement used for floating mot
int n = 0,n_comp = 0,            // Intermediate variables needed to compute me
encoder[n_max];                  // Array containing last n_max motor positions
memset(&encoder[0],0,sizeof(encoder));
starting_balancing_task = false;// We're done configuring. Main task now resum

ClearTimer(T4);                  // This timer is used in the driver.
while(true)
{
  //READ GYRO SENSOR
  #ifdef HTechnic_Gyro
    u =   SensorRaw[Gyro];wait1Msec(2);
    u = u+SensorRaw[Gyro];
  #endif

  //COMPUTE GYRO ANGULAR VELOCITY AND ESTIMATE ANGLE
  dth_dt = u/2 - mean_reading;
  mean_reading = mean_reading*0.999 + (0.001*(dth_dt+mean_reading));
  th = th + dth_dt*dt;
  th_prev=th;

  //ADJUST REFERENCE POSITION ON SPEED AND ACCELERATION
  if(v < speed*10){
  v = v + acceleration*10*dt;}
  else if(v > speed*10){
  v = v - acceleration*10*dt;}
  y_ref = y_ref + v*dt;

  //COMPUTE MOTOR ENCODER POSITION AND SPEED
  n++;if(n == n_max){n = 0;}
  encoder[n] = nMotorEncoder[motorB] + nMotorEncoder[motorC] + y_ref;
  n_comp = n+1;if(n_comp == n_max){n_comp = 0;}
  y = encoder[n]*degtorad*radius/gear_down_ratio;
  dy_dt = (encoder[n] - encoder[n_comp])/(dt*(n_max-1))*degtorad*radius/gear_d

  // ERROR:
  e = (gn_y * y)+(gn_dy_dt * dy_dt)+(gn_th * th)+(gn_dth_dt * dth_dt);

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

```
// FUNCOES DE PERTINENCIA             //
////////////////////////////////////////
if (e>0 & e<(e_max*100)){
  w1=e/(100*e_max);}
  if(e<0){w1=0;}
  if(e>(e_max*100)){w1=1;}

if (e>(-e_max*100) & e<0){
  w2=-e/(100*e_max);}
  if(e<(-e_max*100)){w2=1;}
  if(e>0){w2=0;}

if (e>0 & e<(e_max*100)){
  w3=-(e/(100*e_max))+1;}
  else {w3=(e/(100*e_max))+1;}
  if(e<=(-e_max*100)){w3=0;}
  if(e>=(e_max*100)){w3=0;}

w1_prev=w1;
w2_prev=w2;
w3_prev=w3;

////////////////////////////////////////
// TAXA APRENDIZADO                   //
////////////////////////////////////////
a=0.055;//0.05

////////////////////////////////////////
// REGULARIZACAO                      //
////////////////////////////////////////
gama=0.005;//0.005

////////////////////////////////////////
// PESOS                              //
////////////////////////////////////////
p1_prev=0.5*(p1_prev+p11_prev);
p2_prev=0.5*(p2_prev+p21_prev);

p1=p1_prev+(w1*a*e)-((p1_prev)*gama);
p2=p2_prev+(w2*a*e)-((p2_prev)*gama);
p3=p3_prev+(w3*a*e);

p12_prev=p11_prev;
p11_prev=p1_prev;

p22_prev=p21_prev;
p21_prev=p2_prev;

p1_prev=p1;
p2_prev=p2;
p3_prev=p3;

////////////////////////////////////////
// CONTROLE                           //
////////////////////////////////////////
pid=(p1*w1)+(p2*w2)+(p3*w3);
pid_prev=pid;

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

```
    // LOG VARIAVEIS                          //
    ////////////////////////////////////////
    float yb=nMotorEncoder[motorB]*0.04884;
    float yc=nMotorEncoder[motorC]*0.04884;

    writeDebugStreamLine("%.2f;%.2f;%.3f;%.3f;%.4f;%.2f;%.2f",dth_dt*dt,yb,p1,p2

    //ADJUST MOTOR SPEED TO STEERING AND SYNCHING
    if(steering == 0){
        if(last_steering != 0){
          straight = nMotorEncoder[motorC] - nMotorEncoder[motorB];}
        d_pwr = (nMotorEncoder[motorC] - nMotorEncoder[motorB] - straight)/(radi
    else{d_pwr = steering/(radius*10/gear_down_ratio);}
    last_steering = steering;

    //CONTROL MOTOR POWER AND STEERING
    motorpower =  pid;
    motor[motorB] = motorpower + d_pwr;
    motor[motorC] = motorpower - d_pwr;

    //ERROR CHECKING OR SHUTDOWN
    if(abs(th)>60 || abs(motorpower) > 2000)
    {
    StopAllTasks();}

    //WAIT THEN REPEAT
    while(time1[T4] < dt*1000){
      wait1Msec(1);}
    ClearTimer(T4);
  }
}

#ifdef HiTechnic_Gyro
int calibrate_hitechnic()
{
 //Function for finding the HiTechnic Gyro offset
 int mean_reading = 0, p = 0;

 while(nNxtButtonPressed == kEnterButton){}
 nxtDisplayTextLine(0,"Put Segway Down");
 wait1Msec(500);
 nxtDisplayTextLine(2,"Calibrating");
 nxtDisplayTextLine(3,"HiTechnic Gyro..");
 wait1Msec(1000);
 nxtDisplayTextLine(1,"");
 nxtDisplayTextLine(2,"CONTROLE ONFC");
 nxtDisplayTextLine(3,"");
 nxtDisplayTextLine(4,"UERJ");

 for(p = 0; p < 40;p++){
    mean_reading = mean_reading + SensorRaw[Gyro];
    wait1Msec(50);}
 mean_reading = mean_reading/40;
 PlayTone(500,50);

if(mean_reading < 550 || mean_reading > 640){
   wait1Msec(1000);
    }
 return(mean_reading);


 }
 #endif
```

Controle de equilíbrio ONFC:

```
#pragma DebuggerWindows("debugStream")
/**************
Segway-NoAction
***************

#define HiTechnic_Gyro

const tSensors Gyro = S3;
const float your_wheel_diameter = 56;

/*This is the Segway Driver code. Place in same directory as this program*/
#include "segway-driver-lv1-ONFC.h"

task main()
{
  clearDebugStream();
  //Start balancing and wait for configuration to finish

  StartTask(balancing);
  while(starting_balancing_task){

      //int angulo;
      speed = 0;
      steering = 0;
     }
}
```

Controle de Posição ONFC

```
#pragma DebuggerWindows("debugStream")
/*******INFO*******
Segway-Encoders
**************/
#define HTechnic_Gyro

const tSensors Gyro = S3;
const float your_wheel_diameter = 56;

/*This is the Segway Driver code. Place in same directory as this program*/
#include "segway-driver-lv1-ONFC.h"

task main()
{
  clearDebugStream();
  //Start balancing and wait for configuration to finish
  StartTask(balancing);
  while(starting_balancing_task){}

  //distancia desejada em cm:
  int dist=200;//cm
  int ang=dist/0.04884;

  while(true)
  {
    speed = -5;
    while(nMotorEncoder[motorB] < ang){wait1Msec(100);}
    //#1: Polling 10 times/sec is good enough -- Save processor power for the ba
    //#2: Going forwards means that the motors run backwards, because of their o
    //    So to go forward for 1000 degrees, the encoder must reach -1000 degree
    speed = 0;
    while(nMotorEncoder[motorB] > 0){wait1Msec(100);}
  }
}
```

Código fonte controle PID:

```
 /* Controle PID
*/

int steering = 0;
int acceleration = 50;
int speed = 0;
bool starting_balancing_task = true;

//GLOBAL VARIABLE SETUP
float gn_dth_dt,gn_th,gn_y,gn_dy_dt,kp,ki,kd,mean_reading,gear_down_ratio,dt;
float th = 0,y = 0;
float u = 0;

float d,s_0,s_1;

#ifdef HTechnic_Gyro
int calibrate_hitechnic();
#endif

task balancing()
{

  ////////////////////////////
  //ADVANCED USER CONTROL
  ////////////////////////////

  gear_down_ratio = 1;

  // Set the time each loop cycle should last. You can set it up to 0.03 seconds
  // need to. If you add code to the control loop below (such as to read another
  // all code can run in under dt seconds. If it takes more time, set dt to a hi
  // Default is 0.010 seconds (10 miliseconds).
  dt = 0.010;

  // Customize PID constants. These variables are global, so you can optionally
  gn_dth_dt = 0.23;//default: 0.23
  gn_th = 25.00;//default: 25.00
  gn_y = 272.8;//default: 272.8
  gn_dy_dt = 24.6;//default: 24.6
  kp = 0.0336;//default: 0.0336
  ki = 0.2688;//default: 0.2688
  kd = 0.000504;//default: 0.000504;

  ////////////////////////////
  //END ADVANCED USER CONTROL=
  ////////////////////////////

  //MOTOR SETUP
  nMotorPIDSpeedCtrl[motorB] = mtrNoReg;
  nMotorPIDSpeedCtrl[motorC] = mtrNoReg;
  nMotorEncoder[motorC] = 0;
  nMotorEncoder[motorB] = 0;

  //SENSOR SETUP
  int nSensorsDefined = 0;
  #ifdef HTechnic_Gyro
      SensorType[Gyro] = sensorLightInactive;
      // The following sets the average HiTechnic sensor value. If you know the
      // next time like so: mean_reading = 593.82; (if that's your sensor averag
```

```
    mean_reading = calibrate_hitechnic();
    nSensorsDefined++;
#endif
if(nSensorsDefined != 1){
  nxtDisplayTextLine(0,"Check Sensor");
  nxtDisplayTextLine(1,"definition!");
  wait1Msec(5000);StopAllTasks();
}

//MATH CONSTANTS
const float radius = your_wheel_diameter/1000;
const float degtorad = PI/180;

//SETUP VARIABLES FOR CALCULATIONS
float   dth_dt = 0;//dTheta/dt    // Angular velocity of robot (degree/sec)
float e = 0,//Error              // Sum of four states to be kept zero: th, dth
      de_dt = 0,//dError/dt      // Change of above error
      _edt = 0,//Integral Error // Accumulated error in time
      e_prev = 0;//Previous Error/ Error found in previous loop cycle
float pid = 0;                   // SUM OF PID CALCULATION
//float y = 0,//y                    // Measured Motor position (degrees)
float dy_dt = 0,//dy/dt             // Measured motor velocity (degrees/sec)
      v = 0,//velocity           // Desired motor velocity (degrees/sec)
      y_ref = 0;//reference pos // Desired motor position (degrees)
int motorpower = 0,              // Power ultimately applied to motors
    last_steering = 0,           // Steering value in previous cycle
    straight = 0,                // Average motor position for synchronizing
    d_pwr = 0;                   // Change in power required for synchronizing
const int n_max = 7;             // Number of measurement used for floating mot
int n = 0,n_comp = 0,            // Intermediate variables needed to compute me
encoder[n_max];                  // Array containing last n_max motor positions
memset(&encoder[0],0,sizeof(encoder));
starting_balancing_task = false;// We're done configuring. Main task now resum

ClearTimer(T4);                  // This timer is used in the driver.
while(true)
{

  //READ GYRO SENSOR
  #ifdef HiTechnic_Gyro
    u =   SensorRaw[Gyro];wait1Msec(2);
    u = u+SensorRaw[Gyro];
  #endif
  ////////////

  //COMPUTE GYRO ANGULAR VELOCITY AND ESTIMATE ANGLE
  dth_dt = u/2 - mean_reading;
  mean_reading = mean_reading*0.999 + (0.001*(dth_dt+mean_reading));
  th = th + dth_dt*dt;

  //ADJUST REFERENCE POSITION ON SPEED AND ACCELERATION
  if(v < speed*10){
  v = v + acceleration*10*dt;}
  else if(v > speed*10){
  v = v - acceleration*10*dt;}
  y_ref = y_ref + v*dt;

  //COMPUTE MOTOR ENCODER POSITION AND SPEED
  n++;if(n == n_max){n = 0;}
```

```
    encoder[n] = nMotorEncoder[motorB] + nMotorEncoder[motorC] + y_ref;
    n_comp = n+1;if(n_comp == n_max){n_comp = 0;}
    y = encoder[n]*degtorad*radius/gear_down_ratio;
    dy_dt = (encoder[n] - encoder[n_comp])/(dt*(n_max-1))*degtorad*radius/gear_d

    //COMPUTE COMBINED ERROR AND PID VALUES
    e = gn_th * th + gn_dth_dt * dth_dt + gn_y * y + gn_dy_dt * dy_dt;
    de_dt = (e - e_prev)/dt;
    _edt = _edt + e*dt;
    e_prev = e;
    pid = (kp*e + ki*_edt + kd*de_dt)/radius*gear_down_ratio;

    //Log Variaveis:
    float yb=nMotorEncoder[motorB]*0.04884;
    float yc=nMotorEncoder[motorC]*0.04884;
    //th, dth, yb, yc, y, dy, mb, mc, pi d, d_pw r:
    writeDebugStreamLine("%.2f;%.2f;%.2f;%.2f;%.2f;%.2f",dth_dt*dt,dth_dt,yb,dy_

    //ADJUST MOTOR SPEED TO STEERING AND SYNCHING
    if(steering == 0){
        if(last_steering != 0){
          straight = nMotorEncoder[motorC] - nMotorEncoder[motorB];}
        d_pwr = (nMotorEncoder[motorC] - nMotorEncoder[motorB] - straight)/(radi
    else{d_pwr = steering/(radius*10/gear_down_ratio);}
    last_steering = steering;

    //CONTROL MOTOR POWER AND STEERING
    motorpower =  pid;
    motor[motorB] = motorpower + d_pwr;
    motor[motorC] = motorpower - d_pwr;

    //ERROR CHECKING OR SHUTDOWN
    if(abs(th)>60 || abs(motorpower) > 2000){
    StopAllTasks();}

    //WAIT THEN REPEAT
    while(time1[T4] < dt*1000){
      wait1Msec(1);}
    ClearTimer(T4);
  }
}

#ifdef HiTechnic_Gyro
int calibrate_hitechnic()
{
 //Function for finding the HiTechnic Gyro offset
 int mean_reading = 0, p = 0;

 while(nNxtButtonPressed == kEnterButton){}
 nxtDisplayTextLine(0,"Put Segway Down");
 wait1Msec(500);
 nxtDisplayTextLine(2,"Calibrating");
 nxtDisplayTextLine(3,"HiTechnic Gyro..");
 wait1Msec(1000);
 nxtDisplayTextLine(0,"");
 nxtDisplayTextLine(1,"");
 nxtDisplayTextLine(2,"CONTROLE PID");
 nxtDisplayTextLine(3,"");
 nxtDisplayTextLine(4,"UERJ");
```

```
  for(p = 0; p < 40;p++){
     mean_reading = mean_reading + SensorRaw[Gyro];
     wait1Msec(50);}
  mean_reading = mean_reading/40;
  PlayTone(500,50);

 if(mean_reading < 550 || mean_reading > 640){
    wait1Msec(1000);
     }
  return(mean_reading);
 }
 #endif
```

Controle de Equilíbrio PID:

```
#pragma DebuggerWindows("debugStream")
/*******INFO*******
Segway-NoAction
***************/
/*CHOOSE A SENSOR: Comment out the sensors that you do not have*/
#define HiTechnic_Gyro

/*SELECT SENSOR PORT (S1, S2, S3 or S4), and WHEEL DIAMETER (milimeters).*/
const tSensors Gyro = S3;
const float your_wheel_diameter = 56;

#include "segway-driver-lv-PID.h"

task main()
{
  clearDebugStream();
  //Start balancing and wait for configuration to finish

  StartTask(balancing);
  while(starting_balancing_task){

      //int angulo;
      speed = 0;
      steering = 0;//
     }
}
```

Controle de Posição PID:

```
#pragma DebuggerWindows("debugStream")
/*******INFO*******
Segway-Encoders.c
***************/

#define HTechnic_Gyro

const tSensors Gyro = S3;
const float your_wheel_diameter = 56;

/*This is the Segway Driver code. Place in same directory as this program*/
#include "segway-driver-lv-PID.h"

task main()
{
  clearDebugStream();
  //Start balancing and wait for configuration to finish
  StartTask(balancing);
  while(starting_balancing_task){}

  //distancia desejada em cm:
  int dist=200;//cm
  int ang=dist/0.04884;


  while(true)
  {
    speed = -5;
    while(nMotorEncoder[motorB] < ang){wait1Msec(100);}
    //#1: Polling 10 times/sec is good enough -- Save processor power for the ba
    //#2: Going forwards means that the motors run backwards, because of their o
    //    So to go forward for 1000 degrees, the encoder must reach -1000 degree
    speed = 0;
    while(nMotorEncoder[motorB] > 0){wait1Msec(100);}

  }

}
```

Código fonte controle PIDNN:

```
/*
Controle PIDNN
*/

int steering = 0;
int acceleration = 20;
int speed = 0;
bool starting_balancing_task = true;

//GLOBAL VARIABLE SETUP
float gn_dth_dt,gn_th,gn_y,gn_dy_dt,kp,ki,kd,mean_reading,gear_down_ratio,dt,e_m
float th = 0,y = 0;
float u = 0;


#ifdef HiTechnic_Gyro
//Prototype (see full code below)
int calibrate_hitechnic();
#endif

task balancing()
{

//SaveNxtDatalog();
////////////////////////////
  //ADVANCED USER CONTROL
  ////////////////////////////
  gear_down_ratio = 1;

  dt = 0.010;//0.010

  // Customize PID constants. These variables are global, so you can optionally
  gn_dth_dt = 0.23;//default: 0.23 (g2)
  gn_th = 25;//default: 25.00 (g1)
  gn_y = 272.8;//default: 272.8 (g4)
  gn_dy_dt = 24.6;//default: 24.6 (g3)

  ////////////////////////////
  //END ADVANCED USER CONTROL=
  ////////////////////////////

  //MOTOR SETUP
  nMotorPIDSpeedCtrl[motorB] = mtrNoReg;
  nMotorPIDSpeedCtrl[motorC] = mtrNoReg;
  nMotorEncoder[motorC] = 0;
  nMotorEncoder[motorB] = 0;

  //SENSOR SETUP
  int nSensorsDefined = 0;
  #ifdef HiTechnic_Gyro
      SensorType[Gyro] = sensorLightInactive;
      // The following sets the average HiTechnic sensor value. If you know the
      // next time like so: mean_reading = 593.82; (if that's your sensor averag
      mean_reading = calibrate_hitechnic();
      nSensorsDefined++;
  #endif
  #ifdef DexterIndustries_dIMU
      mean_reading = 0;
      SensorType[Gyro] = sensorI2CCustomFastSkipStates;wait1Msec(200);
```

```
    if(!DIMUconfigGyro(Gyro, DIMU_GYRO_RANGE_250,false)){
      PlaySound(soundException);nxtDisplayTextLine(1,"dIMU Connected?");
      wait1Msec(2000);StopAllTasks();}
    wait1Msec(500);
    nSensorsDefined++;
#endif
#ifdef MicroInfinity_Cruizcore
  mean_reading = 0;
  SensorType[Gyro] = sensorI2CCustom;
  MICCreset(Gyro);
  wait1Msec(500);
  nSensorsDefined++;
#endif
#ifdef MindSensors_IMU
  int   ux,uy,uz;                 // Mindsensors Sensor Measurement
  mean_reading = 0;
  SensorType[Gyro] = sensorI2CCustomFastSkipStates;
  wait1Msec(500);
  MSIMUsetGyroFilter(Gyro, 0x00);
  wait1Msec(1000);
  nSensorsDefined++;
#endif
if(nSensorsDefined != 1){
  nxtDisplayTextLine(0,"Check Sensor");
  nxtDisplayTextLine(1,"definition!");
  wait1Msec(5000);StopAllTasks();
}

//MATH CONSTANTS
const float radius = your_wheel_diameter/1000;
const float degtorad = PI/180;

//SETUP VARIABLES FOR CALCULATIONS
//float u = 0;                    // Sensor Measurement (raw)
//float th = 0,//Theta            // Angle of robot (degree)
float   dth_dt = 0,//dTheta/dt    // Angular velocity of robot (degree/sec)
        th_prev =0;
float e = 0, e1=0,//Error             // Sum of four states to be kept zero: t
      de_dt = 0,//dError/dt     // Change of above error
      _edt = 0,//Integral Error // Accumulated error in time
      e_prev = 0;//Previous Error/ Error found in previous loop cycle
float pid = 0, pid1=0,                // SUM OF PID CALCULATION
      pid_prev = 0;
//float y = 0,//y                   // Measured Motor position (degrees)
float dy_dt = 0,//dy/dt             // Measured motor velocity (degrees/sec)
      v = 0,//velocity          // Desired motor velocity (degrees/sec)
      a,
      a1,gama,J,
    p1=0,
    p2=0,
    p3=0,
    p1_prev=0,
    p2_prev=0.2,
    p3_prev=0,
    o1_prev=0,
    o2_prev=0,
    o3_prev=0,
    o1=0,
    o2=0,
```

```
    o3=0,
      y_ref = 0;//reference pos // Desired motor position (degrees)
int motorpower = 0,             // Power ultimately applied to motors
    last_steering = 0,          // Steering value in previous cycle
    straight = 0,               // Average motor position for synchronizing
    s_1=0,s_11,
    s_2=0,s_22,
    d_pwr = 0;
                // Change in power required for synchronizing
const int n_max = 7;            // Number of measurement used for floating mot
int n = 0,n_comp = 0,           // Intermediate variables needed to compute me
encoder[n_max];                 // Array containing last n_max motor positions
memset(&encoder[0],0,sizeof(encoder));
starting_balancing_task = false;// We're done configuring. Main task now resum


ClearTimer(T4);                 // This timer is used in the driver. Do not us
while(true)
{

  //READ GYRO SENSOR
  #ifdef HiTechnic_Gyro
    u =   SensorRaw[Gyro];wait1Msec(2);
    u = u+SensorRaw[Gyro];
  #endif

  ////////////

  //COMPUTE GYRO ANGULAR VELOCITY AND ESTIMATE ANGLE
  dth_dt = u/2 - mean_reading;
  mean_reading = mean_reading*0.999 + (0.001*(dth_dt+mean_reading));
  th = th + dth_dt*dt;
  th_prev=th;

  //ADJUST REFERENCE POSITION ON SPEED AND ACCELERATION
  if(v < speed*10){
  v = v + acceleration*10*dt;}
  else if(v > speed*10){
  v = v - acceleration*10*dt;}
  y_ref = y_ref + v*dt;

  //COMPUTE MOTOR ENCODER POSITION AND SPEED
  n++;if(n == n_max){n = 0;}
  encoder[n] = nMotorEncoder[motorB] + nMotorEncoder[motorC] + y_ref;
  n_comp = n+1;if(n_comp == n_max){n_comp = 0;}
  y = encoder[n]*degtorad*radius/gear_down_ratio;
  dy_dt = (encoder[n] - encoder[n_comp])/(dt*(n_max-1))*degtorad*radius/gear_d

  ////////////////////////////////////////
  // ERRO 1                             //
  ////////////////////////////////////////

  e = (gn_y * y)+(gn_dy_dt * dy_dt)+(gn_th * th)+(gn_dth_dt * dth_dt);
  de_dt = (e - e_prev)/dt;
  _edt = _edt + e*dt;
  e_prev = e;

  ////////////////////////////////////////
  // PIDNN                              //
```

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

    o1=e;
    o2=_edt;
    o3=de_dt;

    p1=0.036;//default: 0.0336
    p2=0.1998;//default: 0.2688
    p3=0.000358;//default:0.000504

    pid = (p1*o1 + p2*o2 + p3*o3)/radius*gear_down_ratio;

    ////////////////////////////////////////
    // LOG VARIAVEIS                      //
    ////////////////////////////////////////

    float yb=nMotorEncoder[motorB]*0.04884;
    float yc=nMotorEncoder[motorC]*0.04884;

    writeDebugStreamLine("%.2f;%.2f;%.2f;%.2f;%.2f;%.2f",dth_dt*dt,dth_dt,yb,dy_

    //ADJUST MOTOR SPEED TO STEERING AND SYNCHING
    if(steering == 0){
        if(last_steering != 0){
          straight = nMotorEncoder[motorC] - nMotorEncoder[motorB];}
        d_pwr = (nMotorEncoder[motorC] - nMotorEncoder[motorB] - straight)/(radi
    else{d_pwr = steering/(radius*10/gear_down_ratio);}
    last_steering = steering;

    //CONTROL MOTOR POWER AND STEERING
    motorpower =  pid;
    motor[motorB] = motorpower + d_pwr;
    motor[motorC] = motorpower - d_pwr;

    //ERROR CHECKING OR SHUTDOWN
    if(abs(th)>60 || abs(motorpower) > 2000)
    {
    StopAllTasks();}

    //WAIT THEN REPEAT
    while(time1[T4] < dt*1000){
      wait1Msec(1);}
    ClearTimer(T4);
  }
}

#ifdef HiTechnic_Gyro
int calibrate_hitechnic()
{
 int mean_reading = 0, p = 0;

 while(nNxtButtonPressed == kEnterButton){}
 nxtDisplayTextLine(0,"Put Segway Down");
 wait1Msec(500);
 nxtDisplayTextLine(2,"Calibrating");
 nxtDisplayTextLine(3,"HiTechnic Gyro..");
 wait1Msec(1000);
 nxtDisplayTextLine(0,"");
 nxtDisplayTextLine(1,"");
```

```
    nxtDisplayTextLine(2,"CONTROLE PIDNN");
    nxtDisplayTextLine(3,"");
    nxtDisplayTextLine(4,"UERJ");

    for(p = 0; p < 40;p++){
       mean_reading = mean_reading + SensorRaw[Gyro];
       wait1Msec(50);}
    mean_reading = mean_reading/40;
    PlayTone(500,50);

   if(mean_reading < 550 || mean_reading > 640){
      wait1Msec(1000);
       }

    return(mean_reading);
   }
   #endif
```

Controle de Equilíbrio PIDNN:

```
#pragma DebuggerWindows("debugStream")
/*******INFO*******
Segway-NoAction
***************/

#define HTechnic_Gyro

const tSensors Gyro = S3;
const float your_wheel_diameter = 56;
#include "segway-driver-lv3-PIDNN.h"

task main()
{
  clearDebugStream();
  //Start balancing and wait for configuration to finish

  StartTask(balancing);
  while(starting_balancing_task){

      //int angulo;
      speed = 0;
      steering = 0;// sentido horario: se sinal (+)

     }
}
```

Controle de Posição PIDNN:

```
#pragma DebuggerWindows("debugStream")
/*******INFO*******
Segway-Encoders
***************/

#define HTechnic_Gyro

const tSensors Gyro = S3;
const float your_wheel_diameter = 56;

#include "segway-driver-lv3-PIDNN.h"

task main()
{
  clearDebugStream();
  //Start balancing and wait for configuration to finish
  StartTask(balancing);
  while(starting_balancing_task){}

  //distancia desejada em cm:
  int dist=200;//cm
  int ang=dist/0.04884;


  while(true)
  {
    speed = -5;
    while(nMotorEncoder[motorB] < ang){wait1Msec(100);}
    //#1: Polling 10 times/sec is good enough -- Save processor power for the ba
    //#2: Going forwards means that the motors run backwards, because of their o
    //    So to go forward for 1000 degrees, the encoder must reach -1000 degree
    speed = 0;
    while(nMotorEncoder[motorB] > 0){wait1Msec(100);}

  }

}
```