CONFIGURAÇÃO DO ALGORITMO DE TREINAMENTO DE REDE NEURAL ARTIFICIAL PARA O CONTROLE DE UM ROBÔ MÓVEL

FERNANDO BALTAR VERONEZ[1]; THOMAS LUCAS IRIGOITE BARROCO[2]; JONATAS LEMUEL BISPO ZOTTIS[3]; MARCELO ESPOSITO[4]

[1]*Universidade Federal de Pelotas – fernandoveronez_@hotmail.com*
[2]*Universidade Federal de Pelotas – lucasbarroco@globomail.com*
[3]*Universidade Federal de Pelotas – lemueltra@gmail.com*
[4]*Universidade Federal de Pelotas – marcelo.esposito@ufpel.edu.br*

1. INTRODUÇÃO

Utilizando o conhecimento desenvolvido pelos autores em outros trabalhos como, por exemplo, o da estrutura de controle *ON/OFF* aplicada a um robô seguidor de linha e um sistema de aquisição de dados implementado exclusivamente para este robô (BARROCO et al., 2016), o objetivo neste trabalho passou a ser a amostragem dos sinais de modo a tornar possível o controle em tempo real do robô usando uma rede neural artificial (RNA) (HAYKIN, 2001). Barroco et al. utilizaram programação em Java (2016) no ambiente Eclipse e as ferramentas de desenvolvimento Lejos (2016) para o atendimento aos seus objetivos. O Lejos fornece uma interface de programação de aplicação (API) em Java, para o Lego Mindstorms NXT destacando-se a API Robotics.

Os dois códigos citados em Barroco et al. (2016) foram alterados, dando origem a um novo projeto no Eclipse. Três novas classes e três novos métodos em Java (um em cada classe) foram criados. Na primeira classe Java (chamada de *ler_dados*) foi inserido o algoritmo de aquisição de dados e na segunda classe Java (chamada de *seguidor*) foi inserido, o algoritmo de controle *ON/OFF* do seguidor de linha. Na terceira classe (chamada de *major*), foram introduzidos os comandos de execução das outras duas classes, melhorando a organização e o desempenho do código. Foi utilizada uma técnica de processamento paralelo de dados, a ferramenta *thread*, possibilitou que as classes fossem executadas simultaneamente. O uso de *threads* se assemelha muito ao corpo humano, que é capaz de realizar uma variedade de operações em paralelo (respiração, circulação sanguínea e digestão, por exemplo). Uma *thread* nada mais é do que um recurso capaz de configurar fluxos de execução onde cada *thread* recebe um determinado código de programação que pode ser executado simultaneamente com outros códigos configurados em outras *threads*, possibilitando um enorme ganho em desempenho do algoritmo.

2. METODOLOGIA

O processamento paralelo impôs restrições com relação à função de aquisição do ângulo de giro dos motores, pois a mesma não foi capaz de fornecer dados confiáveis ao ser executada em um processamento paralelo de mais de duas classes. A alternativa encontrada para solucionar este problema foi a união dos códigos das classes *ler_dados* e *seguidor* em apenas uma classe, mantendo o processamento paralelo entre as classes *major* e *ler_dados* (que incorporou as instruções e comandos da classe *seguidor*).

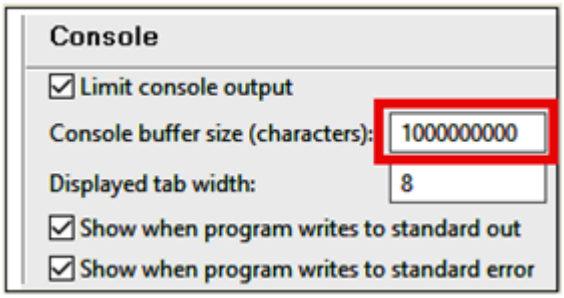

Figura 1 – Console Java-Eclipse.

Na etapa de aquisição de dados do robô seguidor de linha foi preciso aumentar o limite de caracteres no *buffer* destinado a saída de dados do Eclipse. Sem esta alteração não seria possível obter a quantidade necessária de dados para o treinamento da rede neural artificial. O valor de 12500 foi alterado para o valor mostrado na Figura 1.

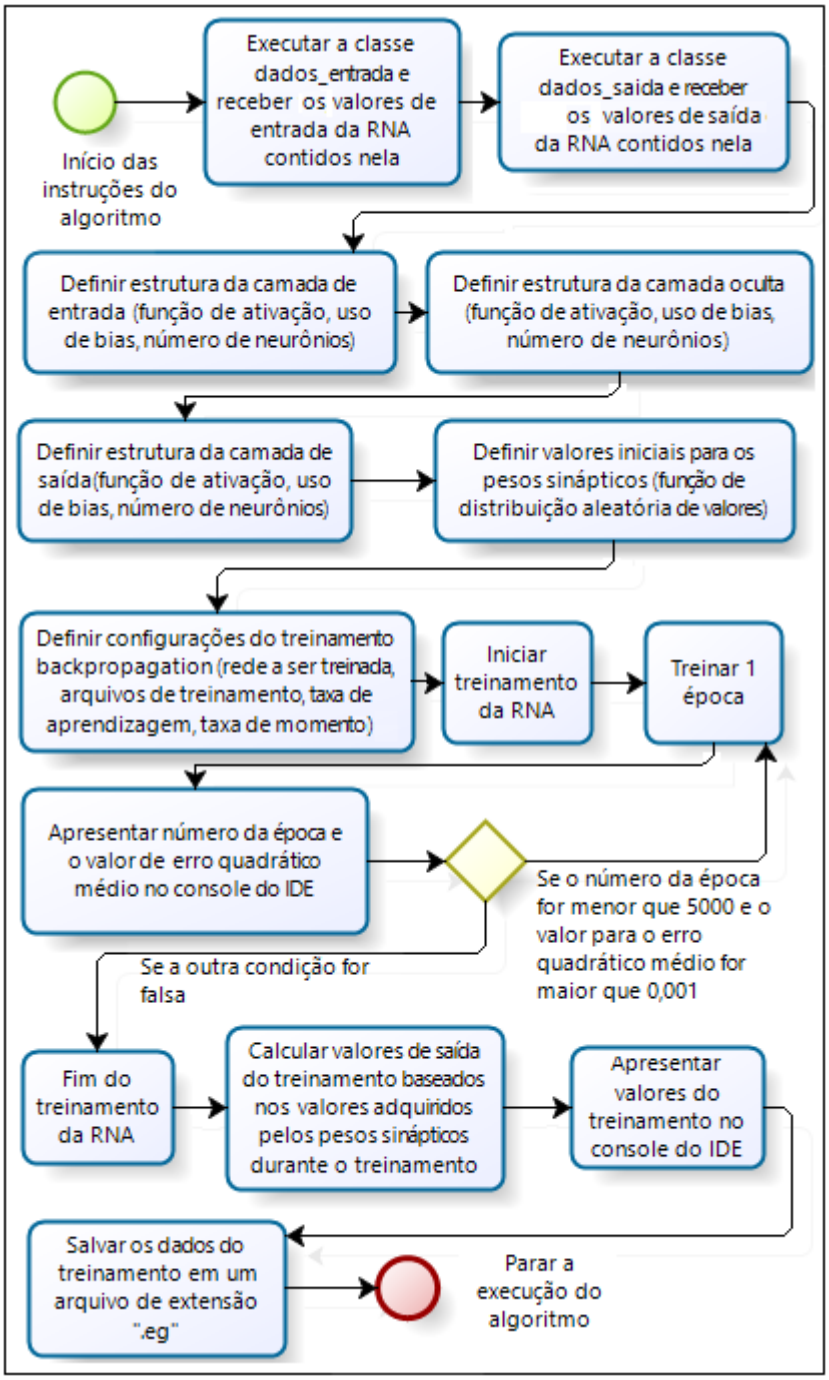

Figura 2 - Diagrama de estados do algoritmo da classe treina

Neste trabalho além das plataformas citadas também foi utilizado o *framework* Encog versão 3.3.0 (2016). Um *framework* define uma arquitetura para uma família de subsistemas e oferece as ferramentas básicas para criá-los, facilitando o desenvolvimento de algumas funcionalidades específicas do código a ser implementado. A maior parte dos algoritmos de treinamento Encog são multi *threads*, ou seja, as funções são realizadas de forma paralela.

Utilizando o algoritmo de treinamento *backpropagation* o fluxograma do algoritmo desenvolvido é apresentado em maiores detalhes em Zottis et. al, 2016).pode ser visto na Figura 2.

Os critérios de parada do treinamento mais utilizados são: parada por número de ciclos, que consiste em determinar qual o número máximo de vezes em que os padrões serão apresentados à rede para treinamento. Um número alto pode dificultar a generalização e um número baixo de ciclos pode comprometer o desempenho da rede. Na parada por erro, um valor para o erro quadrático médio é pré-definido, quando este for atingido o treinamento é encerrado. A parada por validação consiste em descobrir o momento exato em que o processo começa a perder a capacidade de generalização. Nesta técnica utiliza-se um conjunto de dados teste e certo número de ciclos para estimar um erro da rede, a partir do momento em que o erro passa a apresentar um aumento, o processo é encerrado.

3. RESULTADOS E DISCUSSÃO

Grande atenção foi dada ao sentido físico do comportamento dinâmico das variáveis medidas ou calculadas para que posteriormente pudessem ser usadas no treinamento da rede neural. A frequência de amostragem utilizada foi de 1kHz. Na Figura 3 percebe-se a sintonia entre os dados fornecidos pelos encoders incrementais (ângulo de giro) e as velocidades calculadas para cada motor. O crescimento em patamares é decorrente da estratégia de controle *ON/OFF.* A medida que o robô se move ao longo da pista (BARROCO et al., 2016) o ângulo de giro é incrementado e enquanto a velocidade em um motor é diferente de zero no outro ela cresce e depois decresce até parar. Tal comportamento é coerente com a potência dos motores mostrada na Figura 4.

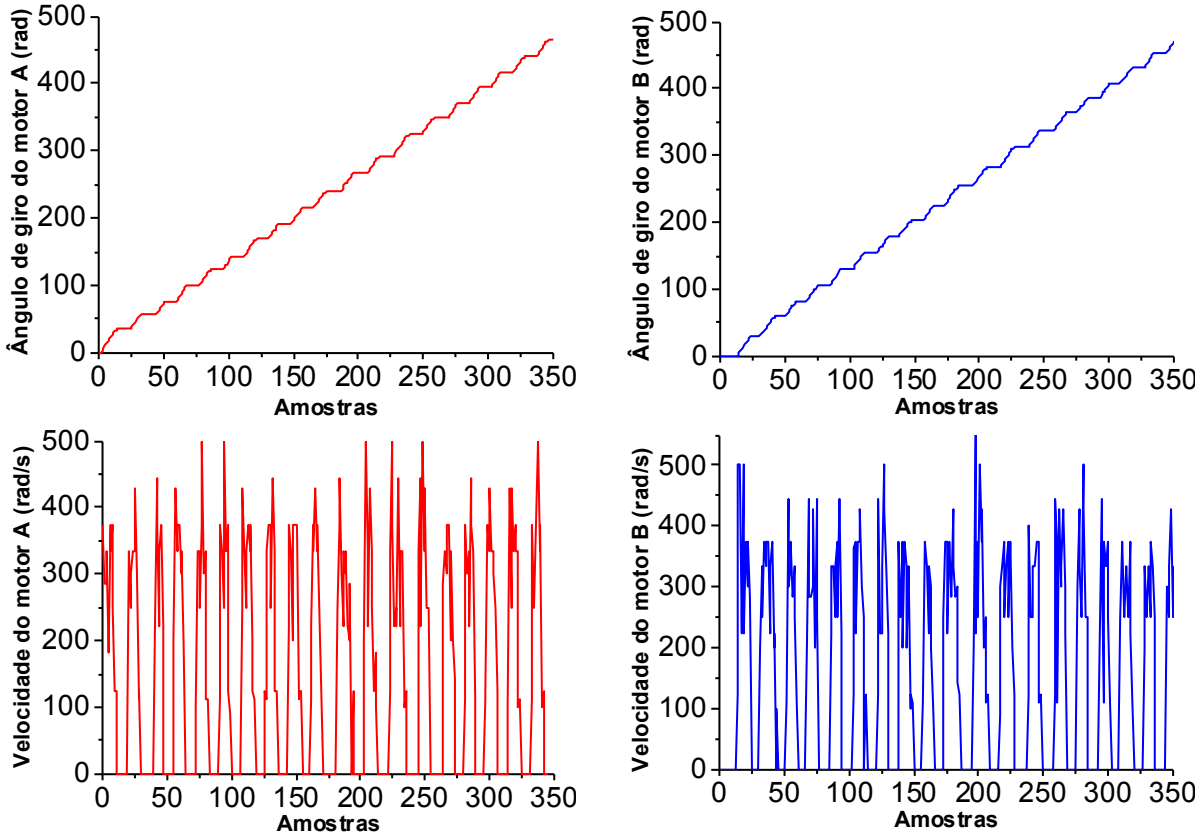

Figura 3 – Comportamento dinâmico do ângulo de giro e das velocidades com a estratégia de controle *ON/OFF*.

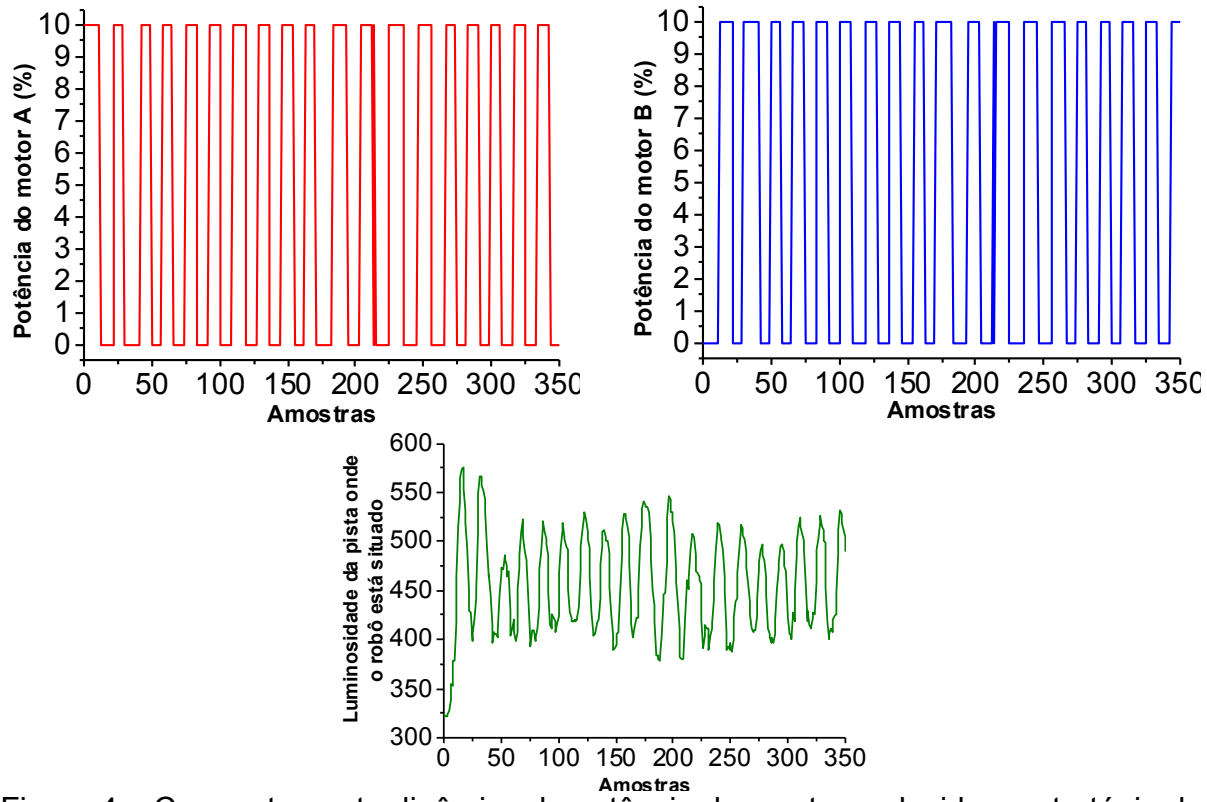

Figura 4 – Comportamento dinâmico da potência dos motores devido a estratégia de controle *ON/OFF* que tem como variável controlada a luminosidade da pista.

4. CONCLUSÕES

Para o adequado treinamento de uma RNA utilizando o algoritmo *backpropagation* é necessário ter bons dados experimentais sobre o comportamento dinâmico do sistema que se deseja controlar. No presente trabalho a estratégia de controle ON/ OFF foi utilizada com sucesso no mapeamento do caminho o ser seguido por um robô móvel. O objetivo de ter um banco de dados do robô operando em malha fechada, para se ter dados de referência, foi cumprido.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BARROCO, T. L. I.; ZOTTIS, J. L. B.; VERONEZ, F. B.; ESPOSITO, M. Aquisição de dados e controle *ON/OFF* dedicados ao treinamento de rede neural artificial. In: **II Congresso de Ensino de Graduação da Universidade Federal de Pelotas**, 2016, Pelotas. Anais do II CEG 2016.

ENCOG. **Encog Machine Learning Framework**. Acessado em 18 jul. 2016. Online. Disponível em: http://www.heatonresearch.com/encog/.

HAYKIN, S., Redes Neurais: Princípios e Prática. 2ª ed. Bookman, 2001.

JAVA. **Java no ambiente Eclipse**. Acessado em 18 jul. 2016. Online. Disponível em: http://www.sun.com/java/.

LEJOS. **Eclipse com *firmware open source***. Acessado em 18 jul. 2016. Online. Disponível em: http://www.leJOS.org/.

ZOTTIS, J. L. B.;BARROCO, T. L. I.; VERONEZ, F. B.; ESPOSITO, M. Rede neural artificial aplicada no controle em tempo real de um robô móvel. In: **II Congresso de Ensino de Graduação da Universidade Federal de Pelotas**, 2016, Pelotas. Anais do II CEG 2016.