UNIVERSIDADE SANTA CECÍLIA
PROGRAMA DE PÓS-GRADUAÇÂO EM ENGENHARIA MECÂNICA
DOUTORADO EM ENGENHARIA MECÂNICA

CLÁUDIO LUÍS MAGALHÃES FERNANDES

CÉLULAS NEURAIS ARTIFICIAIS PARACONSISTENTES UTILIZADAS NO MÉTODO DE APRENDIZAGEM POR DEMONSTRAÇÃO (ApD) PARA MOVIMENTAÇÃO DE ROBÔS INDUSTRIAIS ARTICULADOS COM SEIS GRAUS DE LIBERDADE

SANTOS/SP
2024

CLÁUDIO LUÍS MAGALHÃES FERNANDES

CÉLULAS NEURAIS ARTIFICIAIS PARACONSISTENTES UTILIZADAS NO MÉTODO DE APRENDIZAGEM POR DEMONSTRAÇÃO (ApD) PARA MOVIMENTAÇÃO DE ROBÔS INDUSTRIAIS ARTICULADOS COM SEIS GRAUS DE LIBERDADE

Tese apresentada a Universidade Santa Cecília como parte dos requisitos para obtenção de título de doutor no Programa de Pós-Graduação em Engenharia Mecânica, sob a orientação do Prof. Dr. João Inácio da Silva Filho e coorientação do Prof. Dr. Maurício Conceição Mário.

SANTOS/SP

2024

511.31 Fernandes, Cláudio Luís Magalhães.
S412c Células Neurais Artificiais Paraconsistentes utilizadas no Método de Aprendizagem por Demonstração (Apd) para movimentação de Robôs Industriais Articulados com seis Graus de liberdade/Cláudio Luís Magalhães Fernandes
2024
149 p.

Orientador: Prof. Dr. João Inácio da Silva Filho
Coorientador: Prof. Dr. Mauricio Conceição Mario

Tese (Doutorado) -- Universidade Santa Cecília, Programa de Pós-Graduação em doutorado em Engenharia Mecânica, Santos, SP, 2024.

1. Lógica Paraconsistente. 2. Aprendizagem por demonstração. 3. Redes Neurais Artificiais Paraconsistentes. 4. Sistema de Aprendizagem por demonstração Paraconsistente.
I. Da Silva Filho, João Inácio, orient. II. Mario, Mauricio Conceição, coorient. III. Células Neurais Artificiais Paraconsistentes utilizadas no Método de Aprendizagem por Demonstração (Apd) para movimentação de Robôs Industriais Articulados com seis Graus de liberdade

Elaborada pelo SIBi – Sistema Integrado de Bibliotecas – Unisanta

DEDICATÓRIA

Dedico este trabalho à minha família por todo incentivo, à minha esposa e ao meu filho que me apoiaram das mais diversas maneiras durante esta importante etapa de minha vida.

AGRADECIMENTOS

Um trabalho científico não é desenvolvido somente pelo seu autor. Dentro deste conceito, diversas pessoas tiveram participação importante na concepção, seja através de contribuições técnicas para a confecção dos experimentos, seja através da palavra de conforto e amiga. Portanto, nesse momento, não poderia deixar de agradecer a algumas destas pessoas especiais.

Aos meus orientadores, professores João Inácio da Silva Filho e Maurício Conceição Mario, pela dedicação e orientações relevantes para o desenvolvimento e conclusão desta tese.

Aos amigos Joseffe Barroso de Oliveira e Davi Silvestre da turma de doutorado da UNISANTA, que sempre através de um convívio baseado na amizade e cooperação mutua foram muito além do coleguismo, se tornando hoje meus amigos pessoais.

Aos colegas professores do SENAI-SP, e em especial ao amigo Ricardo Martinez Vicentini, que foi de fundamental importância na conclusão dos experimentos que deram origem a esta dissertação.

A todos os meus familiares que me apoiaram e incentivaram no transcorrer destes três anos de superações, de tratamentos ininterruptos, de dificuldades encaradas dia após dia. Menções especiais a minha querida esposa Karoline que me acolheu nos momentos de dificuldade e me incentivou a finalizar essa etapa tão importante da minha vida.

A todo corpo docente do programa de Pós-Graduação de Doutorado em Engenharia Mecânica da UNISANTA que contribuíram para minha formação, e uma menção especial aos professores Marcos Tadeu Tavares Pacheco e Victor da Silva Rosa, que entenderam e me apoiaram em um momento complexo de saúde, além de toda equipe da secretaria, especialmente as amigas Sandra e Imaculada, pela simpatia, profissionalismo e atenção durante todo o curso.

E finalmente agradeço a DEUS, pois, sem ele nada seria possível. Agradeço por me abençoar e me dar forças para conseguir concluir mais essa etapa em um momento tão difícil da minha vida.

RESUMO

A complexidade na programação e adaptação dos robôs industriais as linhas fabris, pode levar à estagnação na produção, especialmente diante da necessidade crescente da customização dos produtos industrializados. O desafio da reprogramação manual dos robôs, que resulta em tempo significativo de inatividade e redução da eficiência da produção industrial, tem como uma abordagem promissora a ser explorada, o aprendizado por demonstração que utiliza métodos de transferência de conhecimento orientados pela aprendizagem humana. Destaca-se nesse contexto a utilização da Lógica Paraconsistente Anotada (LPA) como base para um processo de aprendizagem por demonstrações repetitivas, visando simplificar a reprogramação de robôs industriais e minimizar o tempo necessário para a alteração de tarefas. Nesta tese são apresentados os resultados de uma aplicação da Lógica Paraconsistente Anotada com anotação de dois valores (LPA2v) em Sistemas Robóticos Industriais utilizando o método de Aprendizado por Demonstração (ApD), que implica em uma máquina executar novas tarefas ao imitar procedimentos apresentados a ela, sem a necessidade de reconfiguração ou reprogramação de seu *software*. A LPA2v possibilita a construção de algoritmos que formam redes de análise capazes de processar sinais, simulando o aprendizado humano. Entre esses algoritmos está o da Célula Neural Artificial Paraconsistente de Aprendizagem (CNAPap), que quando submetido a repetidos sinais padronizados à sua entrada, é capaz de armazenar gradualmente essa informação, ajustando seu nível de resposta na saída de forma assintótica, controlado por um Fator de Aprendizagem (F_A). Neste estudo, foram implementados testes para validar o desempenho de uma rede neural composta por células CNAPap, no contexto do Aprendizado por Demonstração (ApD) na movimentação dos eixos de um Robô Industrial com larga utilização em processos industriais. Diante do contexto é demonstrado nesse trabalho a construção de um programa computacional fundamentado no Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v através do aplicativo de computação numérica MATLAB, sendo esse integrado a um Controlador Programável da marca Schneider modelo M262 através de uma comunicação *Open Platform Communication Unified Architecture* (OPC UA). No processo o CP atua como responsável pelo envio e coleta de dados junto a um robô industrial da marca YASKAWA, modelo MOTOMAN - GP8, escolhido para a implementação prática. Como comunicação entre o controlador e o robô foi utilizado o protocolo de comunicação Ethernet e toda a aplicação do CP teve a sua construção baseada na norma IEC 61131-3. Os resultados demonstraram que as CNAPap exibem propriedades dinâmicas com robustez a perturbações, tanto no processo de aprendizagem por demonstração como nos comandos de ações do robô industrial. Com base nesses resultados, é possível conceber redes neurais artificiais paraconsistentes de maior complexidade, compostas por células CNAPaps, proporcionando uma valiosa contribuição à pesquisa em Aprendizado por Demonstração na área da robótica industrial.

Palavras Chave: Lógica Paraconsistente. Aprendizagem por Demonstração. Robô Industrial. Célula Neural Paraconsistente de Aprendizagem. Sistema Paraconsistente de Aprendizagem por Demonstração.

ABSTRACT

The complexity in programming and adapting industrial robots to manufacturing lines can lead to production stagnation, especially given the growing need for customization of industrialized products. The challenge of manually reprogramming robots, which results in significant downtime and reduced industrial production efficiency, involves learning by demonstration as a promising approach to be explored, using knowledge transfer methods guided by human learning. In this context, the use of Paraconsistent Annotated Logic (PAL) as a basis for a learning process through repetitive demonstrations stands out, aiming to simplify the reprogramming of industrial robots and minimize the time required to change tasks. This thesis presents the results of an application of Paraconsistent Annotated Logic with annotation of two values (PAL2v) in Industrial Robotic Systems using the Learning for Demonstration (LfD) method, which involves a machine executing new tasks by imitating procedures presented to it, without the need to reconfigure or reprogram your software. PAL2v makes it possible to build algorithms that form analysis networks capable of processing signals, simulating human learning. Among these algorithms is the Paraconsistent Artificial Neural Learning Cell (PANCell), which when subjected to repeated standardized signals at its input, is capable of gradually storing this information, adjusting its response level at the output in an asymptotic manner, controlled by a Factor Learning (FA). In this study, tests were implemented to validate the performance of a neural network composed of PANCells, in the context of Learning from Demonstration (LfD) in the movement of the axes of an Industrial Robot with wide use in industrial processes. In view of the context, this work demonstrates the construction of a computational program based on the Paraconsistent Learning System by Demonstration - PLSLfD-PAL2v through the numerical computing application MATLAB, which is integrated with a Programmable Controller from the Schneider model M262 through Open communication Platform Communication Unified Architecture (OPC UA). In the process, the CP acts as responsible for sending and collecting data together with an industrial robot from the YASKAWA brand, model MOTOMAN - GP8, chosen for practical implementation. The Ethernet communication protocol was used as communication between the controller and the robot and the entire CP application was built based on the IEC 61131-3 standard. The results demonstrated that PANCell exhibit dynamic properties that are robust to disturbances, both in the learning process by demonstration and in commanding actions to the industrial robot. Based on these results, it is possible to design paraconsistent artificial neural networks of greater complexity, composed of PANCells, providing a valuable contribution to research in Learning from Demonstration in the industrial robotics area.

Keywords: Paraconsistent Logic. Learning from Demonstration. Robot Industrial. Paraconsistent Learning Neural Cell. Paraconsistent System of Learning from Demonstration.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ApD	Aprendizagem por Demonstração
LfD	*Learning from Demonstration*
CNAP	Célula Neural Artificial Paraconsistente
CNAPap	Célula Neural Artificial Paraconsistente de aprendizagem
CAPb	Célula Artificial Paraconsistente básica
NAP	Nó de Análise Paraconsistente
LP	Lógica Paraconsistente
LPA	Lógica Paraconsistente Anotada
LPA2v	Lógica Paraconsistente Anotada com anotação de dois valores
SPApD	Sistema Paraconsistente de Aprendizagem Por Demonstração
CP	Controlador Programável
IEC	*International Electrotechnical Commission*

LISTA DE SÍMBOLOS

G_C	Grau de Certeza
G_{CT}	Grau de Contradição
G_{CR}	Grau de Certeza Real
F_{tCt}	Fator de tolerância a Contradição
F_{tC}	Fator de tolerância a Certeza
F_{tD}	Fator de tolerância a Decisão
F_A	Fator de Aprendizagem
S1	Saída 1
S2	Saída 2
T	Inconsistente
V	Verdadeiro
F	Falso
$\perp$	Paracompleto ou Indeterminado
μ	Grau de evidência favorável
λ	Grau de evidência desfavorável
φ	Intervalo de certeza
μ_E	Intervalo de evidência resultante
μ_{ER}	Grau de evidência resultante real
μ_{CTR}	Grau de contradição normalizado

SUMÁRIO

1 INTRODUÇÃO

1.1 Importância

A competitividade industrial cresce a cada dia e com isso as exigências relacionadas a produção se elevam na mesma proporção. Dessa forma a indústria a fim de garantir maior eficiência, maior qualidade com custos reduzidos, busca tornar os seus processos cada vez mais automatizados (FERNANDES et al., 2012).

Atualmente um produto tem como característica possuir um ciclo de vida reduzido, no qual a sua renovação ou até mesmo inovação devem ser constantes, isto faz com que grandes empresas invistam em tecnologias, tornando os seus processos mais flexíveis a modificações. Dessa forma, a automação busca otimizar o processo produtivo, fornecendo um produto com menor custo, em tempos curtos e com a maior quantidade e qualidade possível (ROSÁRIO, 2009).

A produção em massa, concebida há mais de um século, envolvia trabalhadores com operações manuais em linhas de montagem estáticas, construindo produtos a partir de peças com montagens distintas. Com a introdução da automação na manufatura, parte das tarefas repetitivas da linha de montagem foi retirada dos seres humanos, resultando em processos mais ágeis e confiáveis. Essas tecnologias proporcionaram na época, avanços significativos na velocidade de produção.

Os Controladores Programáveis (CPs) surgiram como resposta a essas demandas, assumindo o papel de elemento central no controle de uma planta industrial, atuando como o "cérebro" desse sistema. Desde sua concepção em 1968, esses controladores passaram por uma evolução contínua. Nesse percurso, incorporaram conceitos ligados à Inteligência Artificial (IA) suportada pelas lógicas não clássicas, que possuem propriedades que se opõem aos fundamentos estritamente binários da lógica clássica. Devido a isso os algoritmos não clássicos têm sido normatizados para a suas aplicações através dos CPs adquirido relevância em diversas variantes, sempre visando otimizar a automação de processos industriais de maneira mais eficaz (FERNANDES et al., 2012).

No campo de aplicações dos Controladores Programáveis (CPs), dentre inúmeras lógicas não clássicas, tem se destacado a Lógica Paraconsistente Anotada - LPA que, por sua vez, trata as regiões não atingidas pela lógica clássica, como os

valores intermediários entre verdadeiro e falso. Essa condição possibilita um melhor tratamento de indefinições, ambiguidades e inconsistências (DA SILVA FILHO, 2007). Com base na LPA, surgiram as Redes Neurais Artificiais Paraconsistentes (RNAPs), formadas basicamente por Células Neurais Artificiais Paraconsistente (CNAP) e que pertencem à área da Inteligência Artificial (IA), cuja característica é ter o comportamento de um neurônio biológico. Embora o uso prático de Inteligência Artificial, em particular de Células Neurais Artificiais Paraconsistentes, esteja em ascensão constante, sua aplicação, especialmente em Controladores Programáveis (CPs) e sistemas de controle robótico, ainda é limitada, com grande parte desse conhecimento circunscrito ao âmbito acadêmico (DA SILVA FILHO, 2007).

Entre as técnicas utilizadas em inteligência artificial com potencial de aplicação surge a “Aprendizagem por Demonstração - ApD (*Learning from Demonstration - LfD*), que propõe que um robô aprenda com as demonstrações feitas por um “professor” (BRENNA et al., 2008). Com base nesta técnica, novos estudos surgiram a fim de contribuir para a pesquisa visando implementações reais desta técnica nas indústrias que utilizam células robóticas em suas plantas de produção.

Como uma necessidade essencial de estudos e novos caminhos que possam aumentar a eficiência da produção, apareceram nas plantas industriais a robótica, a automação e controle de máquinas. Estas áreas estão cada vez mais sofisticadas proporcionando assim crescentes ganhos produtivos as empresas. No entanto, o elevado custo associado à programação desses equipamentos, majoritariamente realizada manualmente por operadores, além do tempo consumido para novas implementações, consomem recursos valiosos e impõem restrições à flexibilidade quando se consideram alterações nos processos fabris (KARLSSON et al., 2017). Nesse sentido, pesquisas precisam avançar a fim de que a programação de células robóticas possa ser realizada de maneira mais rápida, flexível e intuitiva.

1.2 Problemática

Diante da necessidade de otimizar esses processos fabris com redução de custos, aumento da eficiência operacional e aprimoramento das condições de trabalho, a aplicação de células robotizadas despertou um significativo interesse no

setor industrial (SCHOLZ-REITER, 2007). Uma célula robotizada representa um conjunto de máquinas organizadas em um espaço automatizado, onde um ou mais robôs são integrados para maximizar, agilizar e simplificar um módulo completo de produção, visando alcançar elevados padrões de consistência na qualidade do produto. Assim, a decisão de implantar Células Robotizadas está intrinsecamente ligada à automação da produção, oferecendo vantagens tais como a redução de desperdícios, melhoria da eficiência, diminuição do tempo de produção e minimização de tarefas perigosas ou insalubres.

O controle dos movimentos dos robôs requer algoritmos altamente sofisticados. A programação dessas máquinas envolve *softwares* e plataformas específicas e estruturados para realizar tarefas de movimentação e transporte de objetos com máxima precisão. Entretanto, a cada alteração na trajetória do robô ou nas características dos produtos manipulados, torna-se necessário o envolvimento de especialistas em programação para ajustar o sistema a essas novas demandas (LEVINE, 2016).

Com isso, atualmente, os benefícios alcançados com essa automação começaram a estagnar a produção devido à crescente complexidade dos bens de consumo e a dificuldade na alteração de uma célula robótica inserida em uma planta industrial. Observa-se uma tendência natural de readequação da produção tendo como base a ascensão na ênfase na sua customização, o que amplia cada vez mais a complexidade da utilização de robôs inseridos em um processo. Além disso, uma parcela dos consumidores busca uma experiência mais personalizada e adaptada, ressaltando ainda mais a urgência de se adotar processos de personalização e desenvolvimento de métodos para lidar com produtos altamente complexos.

Os processos pelos quais a automação da manufatura tem como base, a utilização de células robotizadas, por característica apresentam a grande parte de suas operações sendo realizadas de forma manual. Esses robôs são programados para realizar tarefas específicas podendo, caso necessário, serem completamente reprogramados. No entanto, a transferência de novos programas para os robôs acarreta um considerável tempo de inatividade da máquina, reduzindo a eficiência global da instalação e produção, além de prejudicar a retomada da operação em um novo processo, isso devido ao fato de que a programação da reorientação do robô é

realizada manualmente, baseando-se em posições, movimentos e peças fixas.

Esse procedimento, pelo qual a simulação manual do movimento dos robôs e a validação dos processos de produção, onde é incluída a validação da programação original, deixam muito a desejar, especialmente em um contexto de produtos cada vez mais customizados e complexos.

Considerando a multiplicidade de tarefas trabalhosas necessárias para introduzir um robô de uma planta fabril fixa em uma nova tarefa, torna-se evidentemente necessária uma medida que reduza o tempo de inatividade das máquinas, aumentando o rendimento e os lucros.

Nesse contexto, o aprendizado por demonstração emerge como uma abordagem capaz de acelerar a adaptação das estratégias de movimento das máquinas, utilizando a transferência de conhecimento orientada pelos princípios da aprendizagem humana. Isso evidencia a necessidade de algoritmos adaptáveis na programação desses equipamentos, os quais devem ser capazes de transferir habilidades por meio de métodos como intervenção direta, observação, emulação de objetivos, imitação e outras formas de interação (KARLSSON, 2017).

Seguindo essa linha de pesquisa este trabalho apresenta a utilização da Lógica Paraconsistente Anotada LPA como base para criar um processo de aprendizagem por demonstrações repetitivas visando melhorar os procedimentos de reprogramação de robôs industriais e que assim possam minimizar o tempo empregado e simplificar a mudança de suas tarefas na planta industrial. Para isto utiliza-se redes de algoritmos baseados em LPA onde o algoritmo denominado de Célula Neural Artificial Paraconsistente de aprendizagem- CNAPap é o elemento base para a construção de um Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v instalado em CPs e capaz de reprogramar robôs industriais.

1.3 Objetivo

1.3.1 Objetivo geral

Com o objetivo de tornar mais flexíveis novas implementações, no que diz

respeito à programação de máquinas na área de automação industrial e robótica, este trabalho busca novas alternativas no controle da automação, de modo a agregar maior eficiência para os sistemas utilizados na produção industrial, principalmente no que tange as células robóticas instaladas em pátios fabris.

Desta forma, esse trabalho tem como objetivo principal o desenvolvimento e construção de um Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v composto de Células Neurais Artificiais Paraconsistente de aprendizagem (CNAPap) dedicado à programação de aprendizagem por demonstração em Robôs Industriais articulados com seis graus de liberdade.

1.3.2 Objetivos específicos

Visando alcançar o objetivo geral do trabalho, teve-se como objetivos específicos:

a) Implantar Células Neurais Artificiais Paraconsistentes (CNAP) utilizando como base um conjunto de CNAPaps para a tarefa do aprendizado por Demonstração e o controle das repetições atuando nos eixos do robô YASKAWA GP8 com controlador YRC1000micro;
b) Investigar os protocolos de comunicação da máquina e assim estabelecer uma comunicação OPC UA entre o conjunto das CNAPaps desenvolvidas no *software* MATLAB e o CP Schneider LOGIC/MOTION M262 para permitir a ação de teleoperação e recebimento dos sinais que expressem corretamente as posições reais das juntas do robô Industrial YASKAWA de 6 graus de liberdade;
c) Estabelecer através do protocolo Ethernet uma comunicação confiável e separada dos canais de comunicação originais, entre o CP Schneider LOGIC/MOTION M262 e o robô YASKAWA GP8 com controlador YRC1000micro;
d) Implementar um algoritmo secundário utilizando o MATLAB que seja capaz de fazer tratamento de dados com as equações fundamentadas em Lógica Paraconsistente Anotada LPA com a finalidade de gerenciar o envio de informações sobre as posições de movimentação dos eixos ao robô YASKAWA GP8 com controlador YRC1000micro com os estados lógicos relacionados às posições aprendidas.

1.4 Fundamentação Teórica

Esse item do capítulo 1 aborda os fundamentos teóricos que serviram de base para esse trabalho. Para tal estão sendo apresentados a seguir os conceitos da Lógica Paraconsistente, os algoritmos da LPA2v com ênfase na Célula Neural Artificial de Aprendizagem (CNAPap), os conceitos da aprendizagem supervisionada, dos controladores programáveis com ênfase na IEC 61131-3, do software MATLAB, da comunicação OPC UA e protocolo ETHERNET e dos robôs industriais articulados com 6 graus de liberdade.

1.4.1 Lógica Paraconsistente

A tecnologia utilizada nos processos industriais que envolvem a automação da manufatura continua a ser fundamentada no que se conhece como lógica clássica, ou seja, a que utiliza como base apenas valores binários, ou seja, falso (0) ou verdadeiro (1), não permitindo o uso de valores intermediários, como quase verdadeiro ou quase falso. Contudo, à medida que a indústria evolui tecnologicamente e se torna mais automatizada, sofisticada e “inteligente” surgem demandas por lógicas capazes de interpretar dados de maneira mais precisa e próxima das situações reais, as quais geralmente estão repletas de incertezas e contradições.

Para atender essa necessidade, algumas lógicas não-clássicas foram desenvolvidas e hoje estão disponíveis para utilização em diversos temas de pesquisa. Dentre estas, as que mais se destacam são a *Fuzzy*, as Trivalentes, as Multivaloradas e as Paraconsistentes (NETTO et al.,2013).

Na área de tratamento de incertezas, as lógicas paraconsistentes, especialmente na forma conhecida como Lógica Paraconsistente Anotada (LPA), é uma lógica não clássica, proposicional e evidencial e já foi formalizada e descrita com suas equações e todos os seus predicados em (ABE, 1998) e (DA SILVA FILHO et al.,2008).

A LPA tem se destacado pela eficácia de seus métodos na abordagem de informações contraditórias, valorizando-as como possíveis evidências para estruturar resultados que melhor refletem a realidade (DA SILVA FILHO, 2006). A aplicação da LPA se realiza por meio de algoritmos criados através de equações matemáticas

obtidas em análises no seu Reticulado associado, sendo os adotados neste estudo os algoritmos da Lógica Paraconsistente Anotada com anotação de dois valores (LPA2v) (DA SILVA FILHO, 1999).

A Lógica Paraconsistente Anotada com anotação de dois valores (LPA2v) tem como fundamento a interpretação de duas evidências, provenientes de fontes distintas, e denominadas como grau de crença ($\mu 1$) e de descrença ($\mu 2$), onde a primeira é considerada favorável e outra desfavorável (DA SILVA FILHO, 2006) a proposição P que está sendo analisada. Em diversos cenários que empregam a LPA2v, os graus de evidência favorável (crença) e desfavorável (descrença) são considerados como dados de entrada do sistema, como por exemplo, sinais analógicos de corrente e tensão representando medidas de variáveis em processos industriais, leituras de protocolos, entre outros.

A representação da LPA2v é geralmente feita por meio de um reticulado de 4 vértices que, através de anotações de dois valores, possibilita-se expressar estados lógicos extremos em cada um de seus vértices. Estas anotações são geradas a partir de medições efetuadas no meio físico que podem, através da representação no Reticulado da LPA2v, atribuir conotação lógica à proposição P. Estas informações, que são interpretadas como entradas, servem de base para através de análises utilizando a LPA2v obter graus de certeza (Gc) e de contradição (Gct) e assim formar a tomada de decisão a partir do algoritmo. A interpretação dos estados lógicos paraconsistentes resulta da intersecção no reticulado associado à LPA2v dos valores de Graus de Certeza (Gc) e Graus de contradição (Gct), cujos cálculos são mostrados a seguir.

$$G_c = \mu_1 - \lambda \quad (1)$$

$$G_{ct} = \mu_1 + \lambda - 1 \quad (2)$$

Onde: μ é o grau de evidencia favorável à proposição P.

λ é o grau de evidencia desfavorável à proposição P.

Os graus de evidência μ, λ pertencem ao conjunto dos números reais e estão contidos no intervalo fechado [1,0].

Com base nos dois valores gerados de G_c e G_{ct}, é formado um par ordenado que fornece um ponto situado dentro do reticulado definido como estado lógico paraconsistente. Dessa forma, no reticulado da Lógica Paraconsistente Anotada com

Anotação de dois valores (LPA2v) pode-se gerar uma infinidade de pontos e por consequência infinitos estados lógicos no interior do seu Reticulado associado (DA SILVA FILHO et al., 2012).

Para possibilitar o controle de algumas aplicações, o reticulado pode ser dividido em apenas 12 regiões, no qual os estados lógicos extremos são delimitados por C1, C2, C3 e C4 definidos como: Verdadeiro, Falso, Indeterminado e Inconsistente (DA SILVA FILHO et al., 2012). A partir das regiões delimitadas do reticulado pode-se relacionar estados lógicos resultantes obtidos através das interpolações dos graus de certeza e de contradição, que por sua vez são dependentes dos valores dos Graus de Evidência obtidos de medições no meio físico.

Na figura 1 é apresentado o reticulado da LPA2v contendo os 4 estados lógicos extremos representados pelas regiões que ocupam os vértices do reticulado: Verdadeiro, Falso, Indeterminado e Inconsistente formando o conjunto {**V, F, ⊥, T}** e também os 8 valores Não-extremos (DA SILVA FILHO et al., 2012).

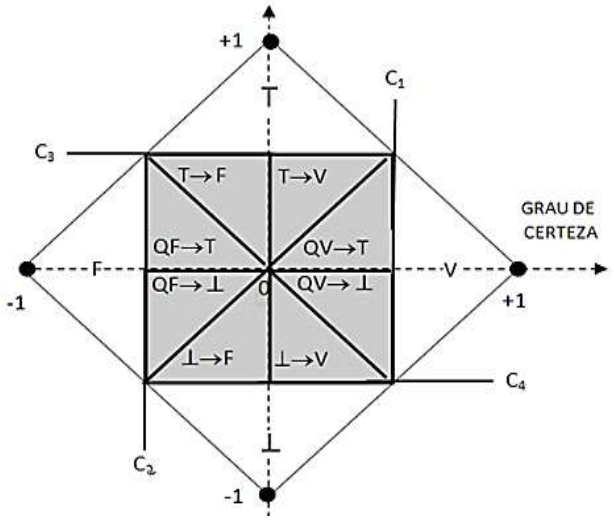


Figura 1 - Reticulado da LPA2v e regiões delimitadas
Fonte: (DA SILVA FILHO, 2007)

A seguir são apresentados os oito estados não extremos:

⊥ → **F** Indeterminado tendendo a Falso
⊥ → **V** Indeterminado tendendo a Verdadeiro
T → **F** Inconsistente tendendo a Falso
T → **V** Inconsistente tendendo a Verdadeiro
QV →**T** Quase verdadeiro tendendo a inconsistente
QF →**T** Quase – falso tendendo a Inconsistente

QV $\rightarrow\perp$ Quase – falso tendendo a Indeterminado
QF $\rightarrow\perp$ Quase – verdadeiro tendendo a Indeterminado

A partir dos valores inseridos nas entradas $\mu 1$ e $\mu 2$, o algoritmo da LPA2v, que é denominado Para-Analisador, faz o processamento e fornece um dos 12 estados lógicos determinados pelas regiões delimitadas (DA SILVA FILHO et al., 2012). Os valores de grau de certeza e de contradição também poderão ser considerados como sinais de saída, caso os mesmos sejam utilizados, por exemplo, em realimentação para um sistema de controle contínuo.

1.4.2 Célula Artificial Paraconsistente de Aprendizagem e Desaprendizagem

O algoritmo Para-Analisador, descrito a partir dos conceitos da Lógica Paraconsistente Anotada de Anotação de dois valores (LPA2v) possui a característica de tratar informações consideradas incompletas e contraditórias, demonstrando, portanto, semelhanças ao comportamento humano. Portanto, as informações recebidas pelos sentidos é um processo mental biológico que pode ser modelado tornando possível assim, construir algoritmos com características de Células Neurais Artificiais Paraconsistentes (MARIO et al., 2007).

De acordo com as contribuições de (DA SILVA FILHO, 2006), é possível compor um conjunto de equações matemáticas da Lógica Paraconsistente Anotada, juntamente com suas interpretações, para conceber um algoritmo denominado Nó de Análise Paraconsistente (NAP). Este algoritmo denominado de NAP representa a unidade fundamental de uma Célula Neural Artificial Paraconsistente (CNAP) (DE CARVALHO JR, 2017) (DA SILVA FILHO, 2006).

A Célula Artificial Paraconsistente básica (CAPb) possui uma estrutura que disponibiliza, a partir dos sinais de graus de evidência favorável (μ) e desfavorável (λ) apresentados em sua entrada, três estados lógicos denominados Graus de Contradição resultante (G_{ct}), Grau de certeza resultante (G_c), ou Indefinição (I) em sua saída. A simbologia da CNAPb é apresentada a seguir na figura 2.

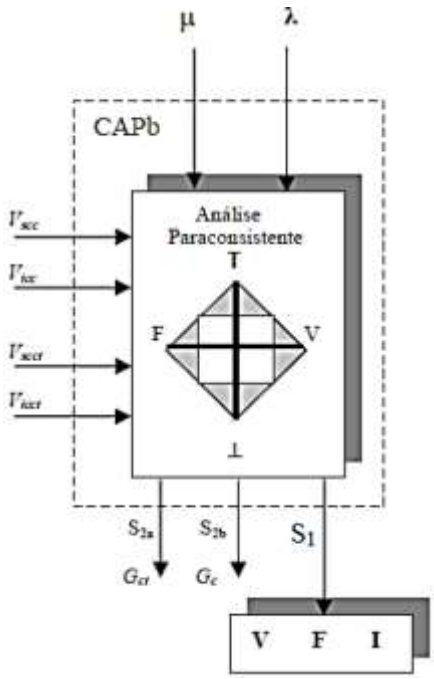


Figura 2 - Símbolo da CNAP básica.
Fonte: De Carvalho Jr, A., 2017

A partir da Célula Artificial Paraconsistente básica (CAPb) foram criadas outras Células denominadas de Células Neurais Artificiais Paraconsistentes (CNAP's), tais como, CNAP de Conexão Analítica, CNAP de Decisão, CNAP de Passagem e CNAP de Aprendizagem, cada qual com uma finalidade específica (DE CARVALHO JR, 2017) (DA SILVA FILHO, 2006). A estrutura de uma CNAP é apresentada na figura 3 a seguir.

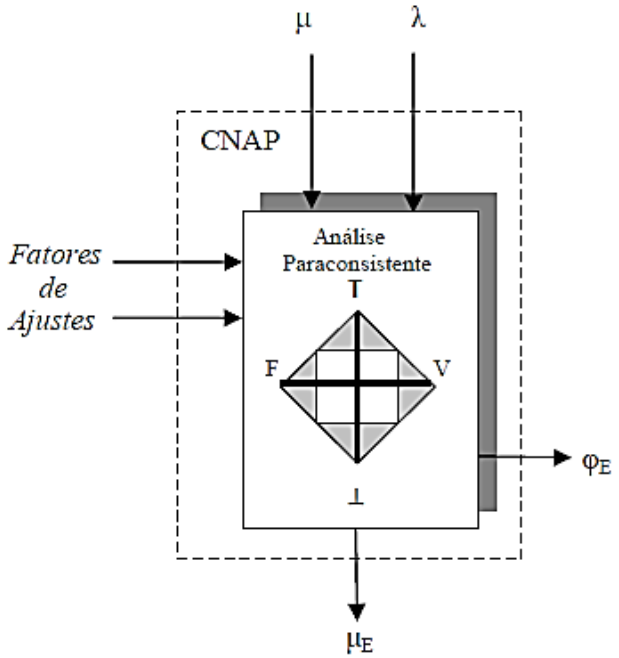


Figura 3 - Símbolo da CNAP.
Fonte: (MARIO et al., 2007)

Uma Célula Neural Artificial Paraconsistente é um elemento capaz de, ao receber um par (λ) de Evidência Favorável e Desfavorável em sua entrada, produzir na saída um resultado composto por um valor de Grau de Evidência resultante (μE) da análise e um valor de Intervalo de Evidência resultante (φEr). Como elemento difusor foi criada a Célula Neural Artificial Paraconsistente padrão CNAPp que foi equipada com todos os fatores de controle e equações para que seja realizada uma análise paraconsistente. A partir da inibição de algumas linhas do algoritmo ou retirada de controles externos da CNAPp, é possível se construir os diversos tipos de células utilizadas por uma rede neural Artificial Paraconsistente-RNAP. Na figura 4 é apresentado a CNAP com as entradas para aplicação dos valores para análise, para controle de ajuste interno e as saídas resultantes do processamento (DE CARVALHO JR, 2017) (DA SILVA FILHO, 2006).

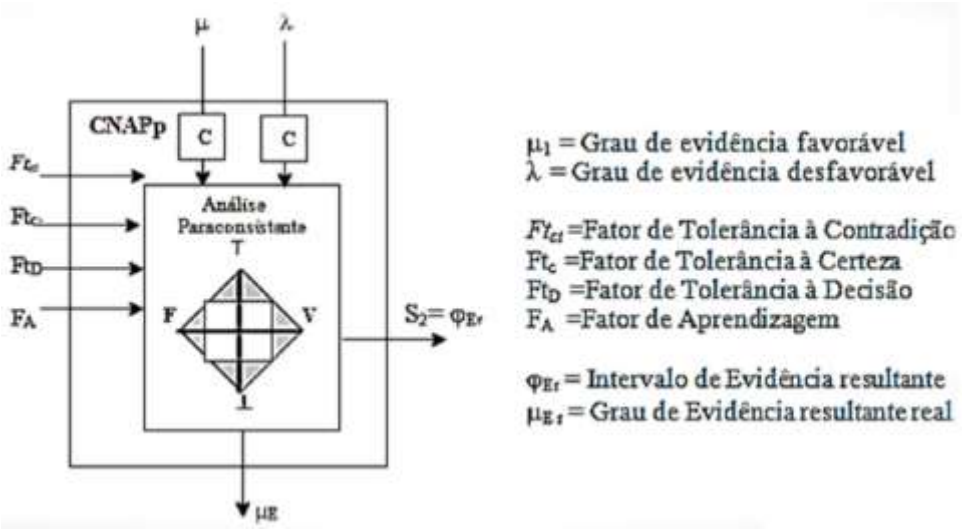


Figura 4 - Símbolo da CNAP com entradas/saídas.
Fonte: (MARIO et al., 2007)

A Célula Neural Artificial Paraconsistente de Aprendizagem (CNAPap) é derivada da CNAP e tem por premissa que após o treinamento, seja capaz de aprender e memorizar padrões entre os valores reais 0 e 1. O padrão em uma rede neural Artificial Paraconsistente é definido como um dígito binário, no qual 1 equivale ao estado lógico “verdadeiro” e o valor 0 significa estado lógico “falso”. O aprendizado do padrão verdade ocorre quando a CNAPap recebe em sua entrada um valor normalizado equivalente a 1 repetidas vezes e a cada repetição, o grau de evidência resultante μE é incrementado, de acordo com o algoritmo, até atingir o valor 1 normalizado (DA SILVA FILHO, 2001).

No aprendizado do padrão de falsidade a CNAPap recebe em sua entrada um valor 0 normalizado repetidas vezes até o grau de evidência resultante chegar a 0, ou seja, a falsidade.

Um fator de aprendizado (F_A) pode ser ajustado externamente de acordo a aplicação, sendo esse responsável por controlar o número de iterações e a precisão da aprendizagem. O fator de aprendizado possui valores também operando no intervalo entre 0 e 1. Quando o F_A for ajustado com valor igual a 1, a CNAPap necessitará de mais passos para concluir um treinamento, neste caso é caracterizado por uma capacidade de aprendizado natural. Quando o F_A é reduzido, o aprendizado natural da célula diminui e quando F_A é ajustado igual 0 a CNAPap perde a capacidade de aprendizagem.

Assim como no padrão de aprendizagem, pode-se também ser empregado um Fator de Desaprendizagem (F_{DA}), o qual é ajustado externamente à célula. Neste caso, ao ser aplicado um valor 0 repetidas vezes na entrada da CNAPap, o grau de evidência resultante será decrementado chegando a um valor de indefinição. Com a repetição na entrada, o grau de evidência continuará diminuindo até chegar a zero, caracterizando assim o desaprendizado do padrão. Para um processo de aprendizagem a sinalização da CNAPap também deve ser 1, então para isso é utilizado um operador NOT (DA SILVA FILHO et al.,2008).

A simbologia que representa uma Célula Neural Artificial Paraconsistente de Aprendizagem (CNAPap) é apresentada na figura 5.

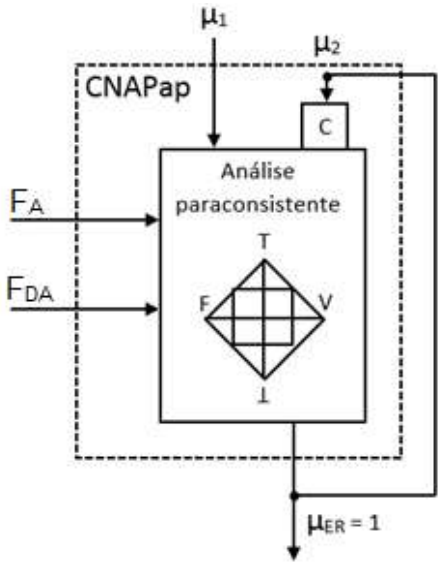


Figura 5 - Símbolo de uma CNAPap.
Fonte: (MARIO et al., 2007)

A equação para determinar o grau de evidencia resultante com base em processo de aprendizagem do padrão de verdade, é apresentada a seguir:

Para: $0 \leq F_A \leq 1$

$$\mu_{E(K+1)} = \frac{\{\mu 1 - (\mu E(K)c)FA\} + 1}{2} \quad (3)$$

Sendo,

$\mu_{E(K)C} = 1 - \mu_{E(K)}$

A equação para determinar o padrão de falsidade é mostrada a seguir:

Para: $0 \leq F_{DA} \leq 1$

$$\mu_{E(K+1)} = \frac{\{\mu 1c - (\mu E(K)c) F DA\} + 1}{2} \quad (4)$$

Sendo,

$\mu_{1C} = 1 - \mu_1$ e $\mu_{E(K)C} = 1 - \mu_{E(K)}$

A Célula Neural Artificial Paraconsistente de Aprendizagem CNAPap tem por finalidade, ao fim de um treinamento, aprender o padrão apresentado na entrada da célula. Para o processo de aprendizagem do padrão de verdade, é aplicado o valor normalizado 1, sendo que a cada passo realizado é gerado um valor de grau de evidência resultante $\boldsymbol{\mu_{ER}}$, através do qual as diferenças geradas entre a entrada e saída são consideradas contradições que se anularão quando o valor da saída $\boldsymbol{\mu_{ER}}$ chegar ao valor 1 desejado. Em um processo de desaprendizagem se dá o processo contrário, ou seja, ao ser aplicado um valor 0 repetidas vezes na entrada da CNAPap, o grau de evidência resultante será decrementado chegando a um valor de indefinição (DA SILVA FILHO et al.,2008). Com a repetição desse valor, o grau de evidência continuará diminuindo até chegar a zero, caracterizando assim o aprendizado de um padrão contrário. A sinalização da CNAPap para um processo de aprendizagem também deve ser 1. O fluxograma do algoritmo de aprendizagem é apresentado na Figura 6.

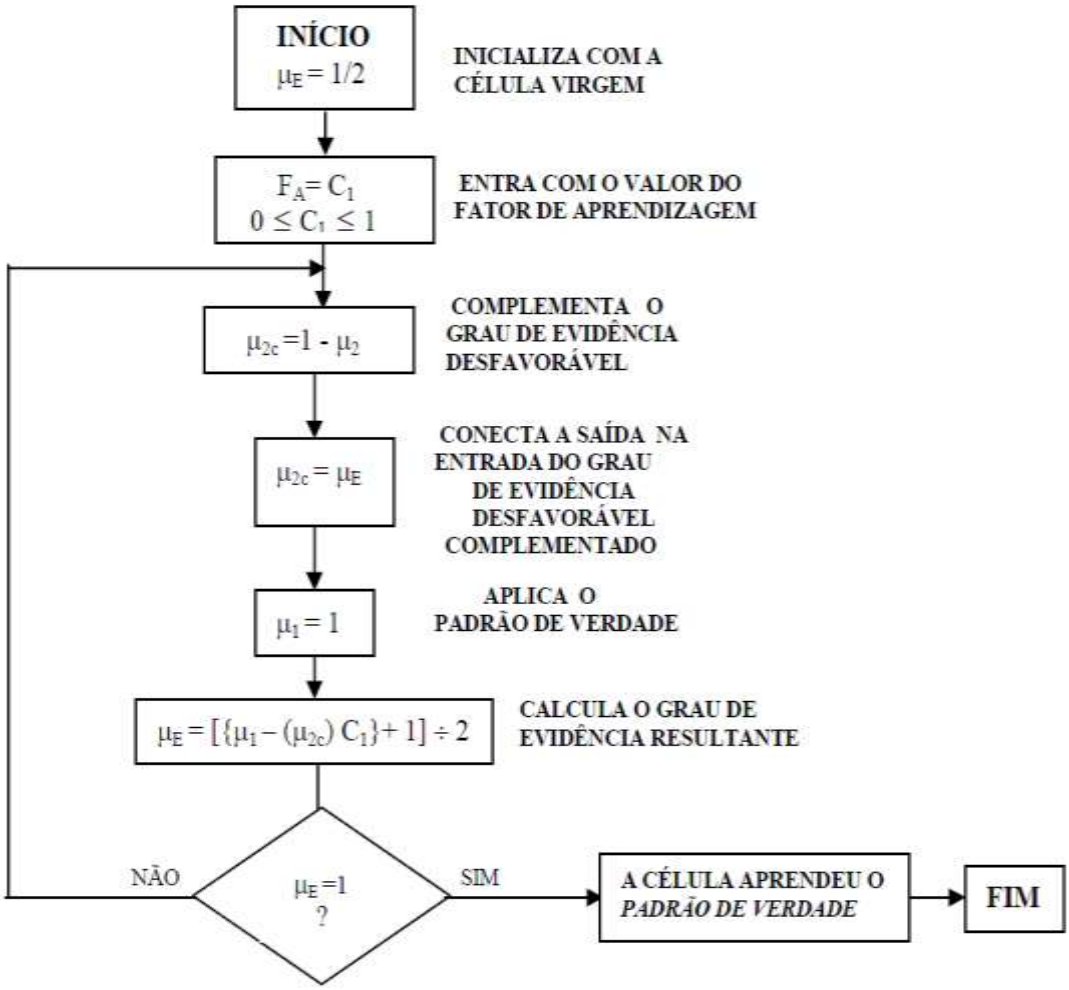


Figura 6 - Fluxograma para aprendizagem do *padrão de Verdade* da CNAPap. Fonte: Da Silva Filho, 2008

Na sequência é apresentado o algoritmo de aprendizagem do padrão de verdade (DA SILVA FILHO et al.,2008) utilizado nesse trabalho.

1) Início
   μE = 0,5 (Célula Virgem)

2) Entre com o padrão de entrada inicial:
   μi = Pdi (0 ≤Pdi ≤ 1)

3) Calcule o grau de evidência inicial
   $\mu E = \frac{\mu i - 0{,}5 + 1}{2}$

4) Determine o padrão a ser aprendido pelas condições:
   Se μE = 0,5, então vá para o item 1 (início)
   Se μE > 0,5, então vá para o item 6

5) Complemente a entrada do Grau de evidência favorável
   $\mu i = 1 - Pd$

6) Entre com o fator de aprendizagem, FA:
   $FA = C4 \quad (0 \leq FA \leq 1)$

7) Conecte a saída da célula na entrada do grau de Evidência desfavorável
   $\mu 2 = \mu E$

8) Aplique o Operador Complemento na entrada do grau de Evidência desfavorável
   $\mu 2c = 1 - \mu 2$

9) Calcule o grau de evidência
   $$\mu E = \frac{\{\mu 1 - (\mu 2c) C4\} + 1}{2}$$

10) Determine o próximo passo pelas condições:
   Se $\mu E = 1$, sinalize célula treinada com padrão de Verdade e vá para o item 14.
   Se $0 < \mu E < 1$, então vá para o item 15.
   Se $\mu E = 0$, sinalize célula treinada com padrão de Falsidade e vá para o próximo item.

11) Faça a negação lógica na saída:
   $\mu E = 1$

12) Faça o complemento na entrada do grau de Evidência favorável
   $\mu iC = 1 - PD$

13) Apresente o fator de aprendizagem
   $FA = C4$

14) Entre com o novo padrão de entrada:
   $Pn = Pd \quad (0 \leq Pn \leq 1)$

15) Retorne ao passo 7

De acordo com os padrões 0 e 1 na entrada da CNAPap e com fator de aprendizagem igual a 1, serão apresentados a seguir na tabela 1 os valores fornecidos pelo algoritmo da CNAPap durante um processo completo de aprendizagem e desaprendizagem.

Tabela 1 - Processo de aprendizagem e desaprendizagem de uma CNAPap

Passo	μ_1	G_c	G_{ct}	μ_E	Observações
0	1	0,0000000000	-1,0000000000	0,5000000000	**Início**
1	1	0,5000000000	0,5000000000	0,7500000000	**Aprendizagem**
2	1	0,7500000000	0,2500000000	0,8750000000	
3	1	0,8750000000	0,1250000000	0,9375000000	
4	1	0,9375000000	0,0625000000	0,9687500000	
5	1	0,9687500000	0,0312500000	0,9843750000	
6	1	0,9843750000	0,0156250000	0,9921875000	
7	1	0,9921875000	0,0078125000	0,9960937500	
8	1	0,9960937500	0,0039062500	0,9984687500	
9	1	0,9984687500	0,0019531250	0,9992343750	
10	1	0,9992343750	0,0009765620	0,9996171870	
	1	**1,0000000000**	**0,0000000000**	**1,0000000000**	**Término**
11	0	-1,0000000000	-1,0000000000	0,5000000000	**Início**
12	0	-0,5000000000	-0,5000000000	0,2500000000	**Desaprendizagem**
13	0	-0,7500000000	-0,2500000000	0,1250000000	
14	0	-0,8750000000	-0,1250000000	0,0625000000	
15	0	-0,9375000000	-0,0625000000	0,0312500000	
16	0	-0,9687500000	-0,0312500000	0,0156250000	
17	0	-0,9843750000	-0,0156250000	0,0078125000	
18	0	-0,9960937500	-0,0039062500	0,0039062500	
19	0	-0,9980468750	-0,0019531250	0,0015312500	
20	0	-0,9990234370	-0,0009765620	0,0004882810	
21	0	-1,0000000000	0,0000000000	0,0000000000	**Término**
22	**0**	**1**	**0**	**1**	**Confirma Falsidade**

Fonte: (DA SILVA FILHO et al.,2008)

De acordo com os valores dos graus de evidência obtidos na tabela anterior é apresentado o gráfico correspondente na figura 7.

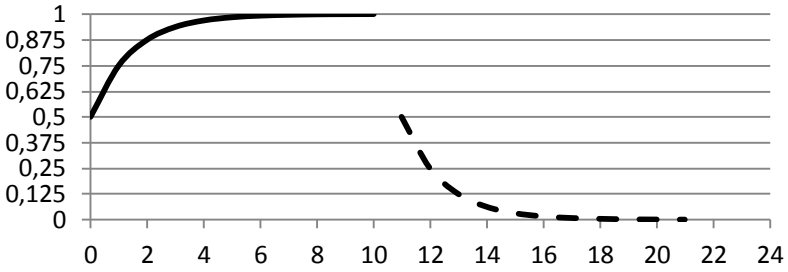


Figura 7 - Valores obtidos no processo de aprendizagem e desaprendizagem. Fonte: (DA SILVA FILHO et al.,2008)

1.4.3 Aprendizado por demonstração (ApD)

Com o avanço de novos meios de produção surgem novas técnicas onde apresenta-se necessidade de se aprofundar em estudos que envolvem a aplicação de novos tipos de lógicas não clássicas e algoritmos de aprendizagem visando melhorias na identificação de padrões (DA SILVA FILHO et al., 2008).

A inovação nesse campo demanda investigações para a assegurar autonomia das máquinas em relação aos processos de programação, robótica autônoma e aprendizado de máquina. Esse trabalho introduz alternativas no controle de robôs industriais que requerem o aprendizado de máquina para aprimorar a eficiência, implementando estruturas algorítmicas fundamentadas na Lógica Paraconsistente - PL (FERNANDES et al. 2012).

Um processo de Aprendizado por demonstração (ApD) (*Learning from Demonstration* (LfD)) envolve a capacidade de uma máquina executar novas tarefas ao imitar procedimentos demonstrados, sem exigir reconfiguração ou reprogramação de seu *software* (FERNANDES, MARIO & DA SILVA FILHO, 2012). Pesquisas relacionadas à aplicação dessa técnica buscam formas eficazes de substituir o trabalho manual de programação das atividades da máquina por um processo automático, reproduzindo exclusivamente a tarefa demonstrada por um especialista (CHERNOVA, 2009).

Conforme visto em (HAYKIN, 2008), um aprendizado por demonstração supervisionado pode ser efetuado com suporte de redes neurais artificiais. Para este tipo de aprendizado onde se espera aplicar o método ApD em um Robô destacam-se três estágios, denominados: Demonstração; Aprendizagem e Imitação. Estes estágios são detalhados a seguir:

a- **Demonstração** - O tutor executa as atividades que o robô precisa aprender. *Na prática, o* tutor, que compreende o ambiente representado por entradas e saídas, orienta o processo. O aprendiz, representado pela rede neural *artificial*, desconhece o ambiente. Ambos tutor e aprendiz são expostos a um vetor de treinamento, com o tutor fornecendo a resposta ideal à rede neural, refletindo a ação ótima conforme o vetor (DA SILVA FILHO et al.,2023) (FERNANDES et al. 2012).

b- **Aprendizagem** - Nessa fase, os parâmetros da rede neural são ajustados com

base no vetor de treinamento e no erro da resposta. A rede simula o tutor, transferindo conhecimento do ambiente para o aprendiz. A verificação da transferência é feita estatisticamente para garantir a completa absorção do conhecimento.

c- **Imitação** – Esta fase é iniciada após completada a fase de aprendizagem. Neste caso a rede neural artificial finaliza o treinamento supervisionado e o tutor não é mais necessário. O aprendiz (Robô), é instruído pela rede Neural artificial com os estados lógicos que foram aprendidos nas duas fases anteriores e assim consegue lidar com o ambiente de forma autônoma

A figura 8 apresenta o diagrama de bloco das etapas principais de um aprendizado supervisionado típico descrito anteriormente.

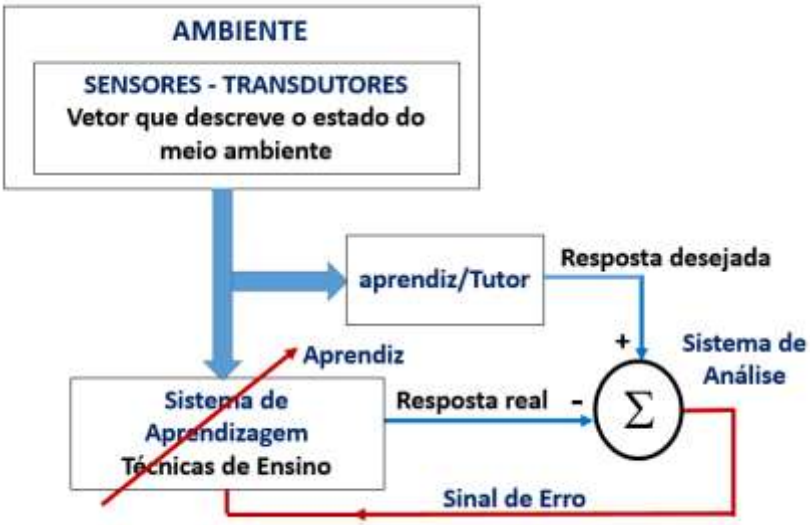


Figura 8 - Diagrama de Blocos de Aprendizagem por Supervisão.
(Da Silva Filho et al.,2023)

Assim, no aprendizado por demonstração supervisionado, os parâmetros da rede são ajustados levando em conta a influência do vetor de treinamento e do erro de sinal, que representa a discrepância entre a resposta real e aquela desejada pela rede (HAYKIN, 2008 conforme foi visto na figura 8.

No que diz respeito aos tipos de abordagem para o estágio de demonstração, na qual o tutor executa as atividades que o robô precisa aprender, tem-se três formas mais utilizadas (SCHAAL, S. 2006), (NIEKUM et al. 2015). São estas:

a- **Cinestésica** - O Tutor demonstra movendo fisicamente o robô, ou a máquina, através dos movimentos ou trajetórias desejadas.
b- **Teleoperação** – O Tutor não entra em contato direto com o Aprendiz, que neste caso é o robô.
c- **Observações Passivas**. O Tutor realiza a tarefa usando seu próprio corpo, que às vezes pode ser instrumentado por sensores adicionais para facilitar o rastreamento.

Os principais equipamentos e aplicativos utilizados nessa pesquisa são descritos a seguir.

1.4.4 Controladores Lógicos Programáveis (CLPS) e a Norma IEC 61131-3

A utilização inicial dos CLPs surgiu quando a equipe técnica da General Motors ofereceu uma especificação para um módulo a relês, que atendeu às necessidades da indústria manufatureira como um todo. Desde então, para atender às demandas do mercado, os CLPs evoluíram incorporando blocos lógicos digitais, aumento da velocidade de processamento, introdução de entradas e saídas analógicas para o controle de Processos Contínuos e, especialmente, melhorias na interface com o usuário (HUGHES,1989).

Nos sistemas de Automação Industrial atuais, diferentes Controladores Lógicos Programáveis (CLPs) de fabricantes variados estão interconectados em uma mesma planta de processos. Eles desempenham funções de gerenciamento e controle do processo industrial e precisam estabelecer comunicação entre si e com o respectivo "Supervisório de Controle". Essa necessidade impulsionou a criação de uma padronização entre os fabricantes de tais equipamentos, resultando no que é atualmente conhecido como "Controlador Programável" (CP). Este termo é adotado devido à sua capacidade de executar funções mais complexas do que os antigos CLPs, como o controle de malhas analógicas usando blocos mais elaborados e o aumento da capacidade aritmética (OLIVEIRA et al., 2007).

No desenvolvimento desse trabalho foram utilizados os conceitos da norma IEC 61499 a qual é uma evolução da IEC 61131-3, cujo objetivo é estabelecer padrões para Controladores Programáveis (CPs) no que diz respeito a execução do programa,

a estrutura de *software* e principalmente para linguagens de programação. Antes de a norma ser criada, cada fabricante de CP adotava a sua, ou seja, não existia um padrão comum definido e isso gerava grandes dificuldades para o setor industrial. Em um processo totalmente automatizado onde existe uma grande quantidade de controladores instalados, se em algum momento for necessária uma alteração por motivos como aumento de produção e melhoria na segurança, se faz necessário a alteração da programação dos CPs envolvidos. Para isso, normalmente, apenas o programador que responsável pela arquitetura do programa teria condições de alterá-lo, pois, naquela época, este possuía características únicas devido à falta de padronização da linguagem, o que acabava sendo extremamente complicado de se realizar por outras pessoas (SILVEIRA, 2013).

Para resolver esse, dentre outros problemas, em 1992, a IEC publicou a primeira edição da norma IEC 61131-3. A partir daí, foi definido um padrão para linguagens de programação dos Controladores Programáveis e que os fabricantes deveriam seguir. Outro importante benefício da norma e que fora utilizado neste trabalho, está relacionado ao desenvolvimento de programas baseados nos seguintes princípios (JOHN, 2010):

a) Modularização: torna possível decompor um programa simples ou complexo em partes menores, possibilitando maior entendimento e controle sobre ele;
b) Estruturação: possibilita elaborar um programa de forma hierárquica, ou seja, em níveis, o que também proporciona a reutilização de blocos funcionais;
c) Tarefas (Tasks): controla a execução de programas ou de blocos funcionais de forma periódica ou mesmo por eventos. A sua criação e necessária em programas mais complexos, mas principalmente em situações de emergência. Por exemplo, no caso de algum defeito em um equipamento em que deve ocorrer a sua parada imediata, o sistema interrompe o ciclo normal de processamento e atende prioritariamente a linha de programa específico dessa emergência.

Atualmente a IEC 61499 expande a IEC 61131-3, aprimorando o encapsulamento de componentes de software para promover uma maior reutilização,

adotando um formato padrão independentemente do fornecedor e simplificando a comunicação entre controladores.

O padrão IEC 61499 expande e aprimora o IEC 61131-3, solucionando desafios de portabilidade, configurabilidade e interoperabilidade de software entre os diferentes fornecedores. Sua capacidade de distribuição e suporte a reconfiguração dinâmica oferece a infraestrutura essencial para aplicações no que denominamos de “Quarta Evolução Industrial” e na *Internet* das Coisas industrial (IIoT), definindo uma linguagem de alto nível para sistemas de controle distribuído, sendo a base técnica da *Universal Automation* devido a três principais razões:

a- Permite a criação de aplicativos de automação com componentes de software portáteis, independentemente do *hardware* subjacente.

b- Possibilita a distribuição flexível dos aplicativos em diferentes arquiteturas de *hardware*, desde sistemas altamente distribuídos até centralizados, sem esforço excessivo de programação.

c- Suporta práticas de *software* convencionais, facilitando a integração de aplicativos de automação com sistemas da Tecnologia da Informação, principalmente no que tange a utilização da IA impulsionando a transição de controladores proprietários para sistemas de automação baseados em componentes de *software* comprovados e de alto valor (OLIVEIRA et al., 2007).

O Bloco Funcional (FB), segundo a norma IEC 61499 encapsula as funcionalidades desejadas e, similarmente à IEC 61131-3, mantem suas entradas à esquerda e as saídas à direita, porém, sua interface distingue eventos de dados. Os eventos, representados por linhas vermelhas na parte superior do FB, ativam as funcionalidades dos blocos, que, por sua vez, utilizam os dados disponíveis nas entradas correspondentes. Os eventos e conexões de dados não são compatíveis, o que impede uma interconexão direta, no entanto, conexões de dados podem ser distribuídas de acordo com a necessidade. Sua representação visual é exibida na figura 9.

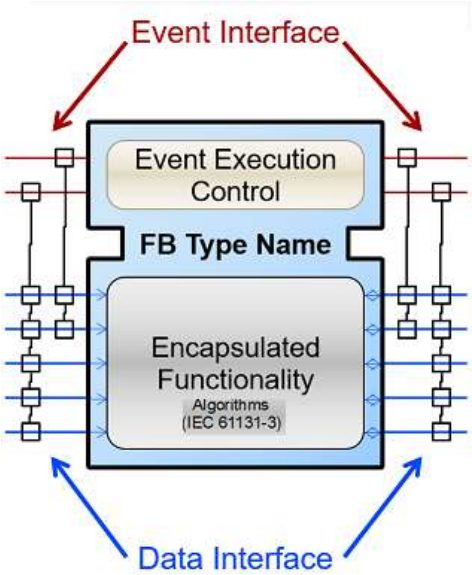


Figura 9 - Bloco Funcional – Norma IEC 61499.
Fonte: Manual CP Schneider M262

1.4.5 MATLAB

O MATLAB é um ambiente de computação técnica com reconhecida aplicabilidade em diversas pesquisas científicas. Alguns dos atributos fundamentais desta plataforma incluem (ATTAWAY, 2013):

a) Ambiente Interativo e Linguagem de Alto Nível: O MATLAB oferece uma interface interativa que permite a programação em sua linguagem de alto nível, facilitando o desenvolvimento de algoritmos e a manipulação de dados.
b) Ferramentas Avançadas de Computação Numérica: Reconhecido por suas capacidades de computação numérica avançada, o MATLAB é capaz de realizar cálculos complexos e análises numéricas (LATHI 2017).
c) Visualização e Gráficos: MATLAB oferece ferramentas poderosas para criação de gráficos e visualizações, essenciais na representação de dados. O MATLAB possui capacidade de produzir gráficos 2D e 3D de alta qualidade, facilitando a interpretação visual dos resultados (CHAPMAN, 2009).
d) Bibliotecas e Toolboxes: Conta com uma variedade de bibliotecas e Toolboxes especializadas para diferentes áreas cuja diversidade dessas ferramentas,

abrangem desde processamento de sinais e imagens até controle e processamento estatístico.

e) Versatilidade em Aplicações Científicas: O MATLAB é amplamente utilizado em uma variedade de pesquisas científicas, incluindo engenharia, física, biologia computacional e finanças, devido à sua flexibilidade e capacidade de lidar com problemas complexos (MOORE, 2012) (ATTAWAY, 2013).

Essas características, ressaltam a relevância do MATLAB como uma ferramenta fundamental para análise, modelagem e simulação em diversas áreas científicas e técnicas tais como a implantação de algoritmos voltados a IA

.

1.4.6 Comunicação OPC

O OPC (*Open Platform Communication* é um padrão de interoperabilidade utilizado na troca segura de dados entre diversos fabricantes de equipamentos utilizados em uma automação fabril. Esse padrão define um protocolo em tempo real que viabiliza a gestão de múltiplos dispositivos na planta industrial, incluindo sensores, dispositivos de campo, controladores e aplicações na linha de produção, além de facilitar a comunicação entre os sistemas OT (*Operational Technology*) e os sistemas corporativos de TI (*Tecnologia da Informação*) em nuvem (ATTAWAY, 2013).

Na Indústria o protocolo OPC é empregado como um protocolo de comunicação M2M (Máquina para Máquina). Sua aplicação permite a comunicação entre diferentes componentes do sistema de automação, possibilitando a coleta e análise de dados de cada componente. Isso resulta na otimização da eficiência, capacidade de resposta e na tomada de decisões mais embasadas no âmbito empresarial. Seu propósito é agir como um elo entre aplicações baseadas em Windows e dispositivos de controle de processos industriais. No contexto do OPC, o servidor desempenha a função de converter o protocolo de comunicação proveniente do hardware, enquanto qualquer software que necessite se conectar a esse hardware atua como cliente OPC.

As diretrizes do OPC são utilizadas por clientes e servidores OPC para identificar, enviar e controlar comandos executados em controladores ou módulos de E/S (Entrada e Saída) (ATTAWAY, 2013).

1.4.7 Protocolo Ethernet

O protocolo Ethernet é uma tecnologia utilizada na comunicação de redes, reconhecido por suas características distintas. Destacam-se suas propriedades fundamentais, tais como:

a- Padrão de Comunicação: O protocolo Ethernet é uma norma de comunicação amplamente utilizada em redes locais (LANs), apresentando um conjunto de regras e procedimentos para transmissão de dados.

b- Velocidade e Largura de Banda: Oferece uma variedade de velocidades de transmissão, desde 10 Mbps (Ethernet padrão) até gigabits por segundo (Ethernet Gigabit e além), proporcionando ampla largura de banda para transferência de dados.

c- Arquitetura de Quadro: Utiliza uma estrutura de quadro (frame) para encapsular os dados, contendo informações como endereços de origem e destino, controle de erro e dados úteis, possibilitando a transmissão confiável e eficiente de pacotes.

d- Flexibilidade e Adaptabilidade: É uma tecnologia flexível que se adapta a diferentes meios de transmissão (por exemplo, cabo de par trançado, fibra óptica) e topologias de rede (estrela, barramento), permitindo sua implementação em uma ampla gama de ambientes de rede.

e- Eficiência de Colisão e Controle de Acesso ao Meio: Utiliza métodos de detecção de colisão (CSMA/CD - *Carrier Sense Multiple Access with Collision Detection*) para gerenciar o acesso ao meio compartilhado, garantindo a eficiência da transmissão de dados em ambientes de rede com múltiplos dispositivos.

f- Padrões de Evolução: Evoluiu ao longo do tempo, com melhorias contínuas em suas especificações, como o desenvolvimento de velocidades mais altas (Ethernet de 10 Gbps, 40 Gbps, 100 Gbps), garantindo a compatibilidade com tecnologias emergentes.

Essas características do protocolo Ethernet são essenciais para a sua implementação em redes de comunicação, contribuindo significativamente para a eficiência, confiabilidade e escalabilidade dos sistemas de comunicação modernos sendo hoje utilizado em larga escala em dispositivos industriais como os CPs e robôs industriais (ATTAWAY, 2013).

1.4.8 Robô Industrial

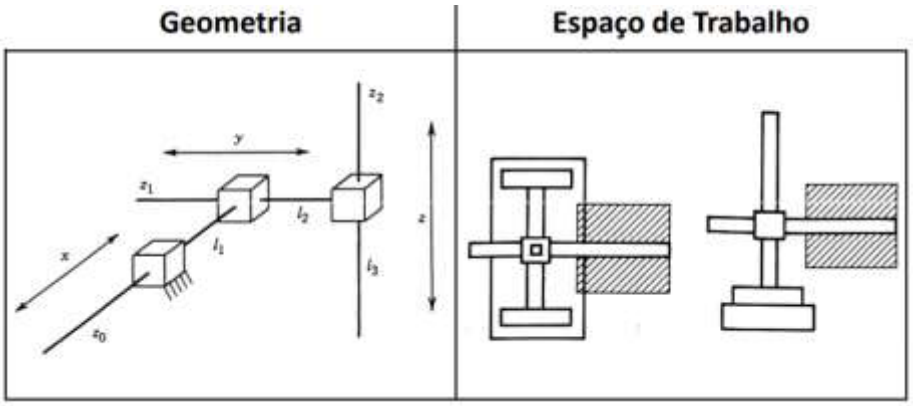
Os robôs são sistemas mecânicos automatizados que executam uma variedade de movimentos de acordo com programas pré-definidos, podendo se ajustar para atender às exigências específicas de diferentes tarefas. Esses dispositivos podem utilizar garras ou ferramentas adequadas para desempenhar suas funções designadas (GIENGER et al., 2010) (DA SILVA FILHO et al., 2006). Estes têm como premissa serem categorizados com base na sua estrutura mecânica, pois a combinação de diferentes elementos, como juntas e elos, possibilita a obtenção de configurações específicas (SCHIAVICCO & SICILIANO, 1995) (GIENGER et al., 2010).

Segundo Schiavicco & Siciliano (1995), as principais configurações básicas quanto à estrutura mecânica de um robô Industrial, apresentadas na figura 10 são:

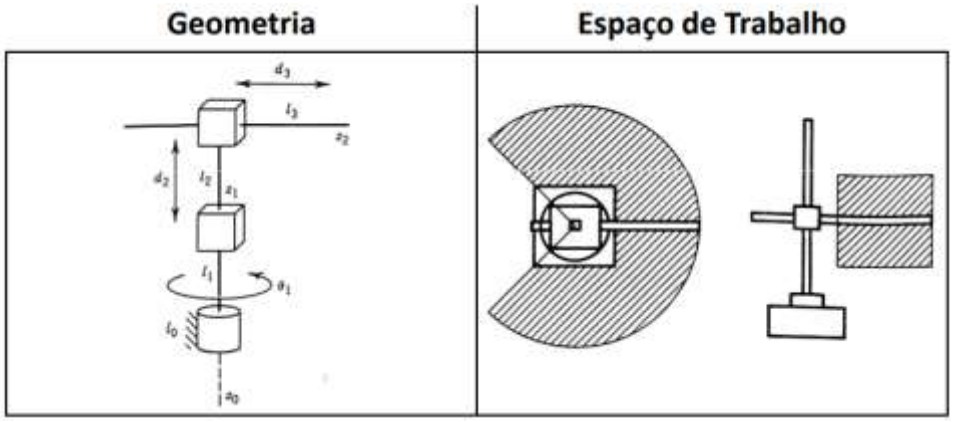
a) Robô de Coordenadas Cartesianas/Pórtico: Este tipo de robô possui três juntas prismáticas (PPP), realizando movimentos de três translações em um sistema de coordenadas cartesianas. Sua representação geométrica é ilustrada na Figura 10(a), destacando a geometria e o espaço de trabalho definido pelas coordenadas cartesianas (d1 [x], d2 [y], d3 [z]) para a posição da garra em relação à base.

b) Robô de Coordenadas Cilíndrica: Com movimentos descritos em coordenadas cilíndricas, esse robô possui dois eixos de translação e um de rotação (RPP). A representação geométrica está na Figura 10(b), evidenciando as coordenadas cilíndricas (θ1, d2, d3) para a posição da garra em relação à base.

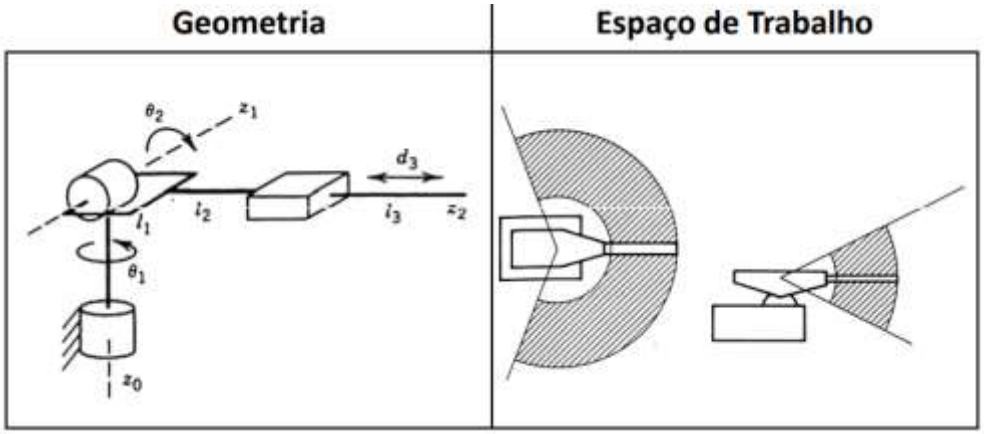
c) Robô de Coordenadas Esféricas: Este robô tem movimentos baseados em um sistema de coordenadas polar, com uma junta prismática e duas de rotação (RRP), executando uma translação e duas rotações. Sua representação geométrica na Figura 10(c) mostra as coordenadas esféricas (θ1, θ2, d3) para definir a posição da garra. Os eixos Z0, Z1 e Z2 são mutuamente perpendiculares (GIENGER et al., 2010).



(a)



(b)



(c)

Figura 10 - Configurações básicas quanto à estrutura mecânica de Robô Industrial.
a) Robô de Coordenadas Cartesianas/Pórtico.
b) Robô de Coordenadas Cilíndrica.
c) Robô de Coordenadas Esféricas.

Nos diversos setores industriais são encontrados modelos diferentes de robôs, tais como o cartesiano, cilíndrico, SCARA, polar, delta, colaborativo e o do tipo articulado, sendo esse o mais utilizado em ambientes industriais.

O robô articulado possui seu braço robótico anexado à sua base por meio de uma articulação de torção, permitindo assim a rotação do robô. O número de articulações rotativas conectando os elos do braço pode variar de duas a dez, e cada articulação acrescenta um grau adicional de liberdade ao sistema. Essas articulações são também conhecidas como eixos, sendo comum se referir a um robô com base no número de eixos, como por exemplo, robô de 4 eixos ou robô de 6 eixos. As articulações podem estar dispostas de forma paralela ou ortogonal entre si. Entre os manipuladores robóticos articulados, aqueles com seis eixos ou graus de liberdade são os mais amplamente utilizados na indústria, apresentado na figura 11 a seguir, sendo produzidos por diversos fabricantes como STAUBBLI, KUKA, SCHNEIDER, ABB, MITSUBISHI, YASKAWA MOTOMAN, entre outras, apresentando como vantagens (GIENGER et al., 2010): a) Alta velocidade de movimentação; b) Maior flexibilidade para aplicações que requerem menor espaço físico. c) Maior facilidade de alinhamento dos movimentos a variadas coordenadas (X,Y,Z).

Em contraponto, o modelo articulado necessita de um controlador dedicado à sua comunicação e programação, que nesse tipo de robô é mais complexa e apresenta mão de obra escassa. Como principais aplicações o robô articulado de seis eixos é utilizado em:

a- Embalagem de alimentos;
b- Produção de fármacos;
c- Soldagem a arco;
d- Soldagem a ponto;
e- Manuseio de materiais.
f- Alimentação de máquinas;
g- Montagem automotiva;
h- Paletização;
i- Corte de aço;
j- Manipulação de vidro;
k- Fundição e aplicação de forjamento entre outros.

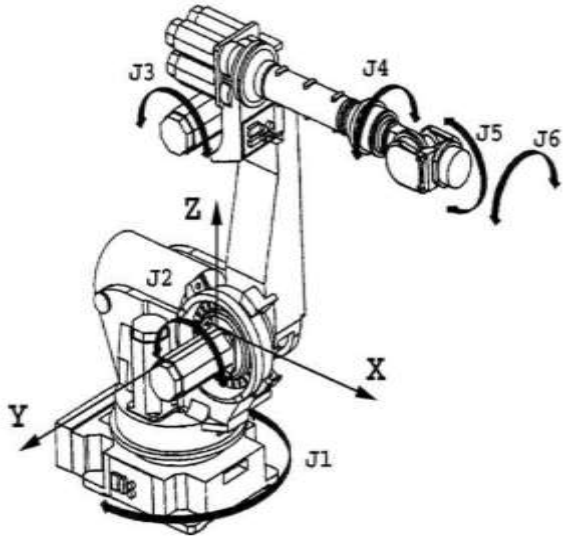


Figura 11 - Robô Articulado de 6 eixos ou graus de liberdade.
Fonte: Manual YASKAWA MOTOMAN – GP8

Para enviar e coletar dados para a movimentação de um robô é necessário utilizar um tipo de interface de comunicação compatível com o sistema do robô que possuem sistemas de controle e programação que permitem a comunicação por diferentes meios, como (GIENGER et al., 2010):

a- *Software* de Programação: O *software* utilizado para programar o robô pode oferecer opções para enviar e coletar dados, sendo único e exclusivo para cada fabricante, ou seja, cada marca de robô possui sua plataforma e console de programação.

b- *Interfaces* de Comunicação: Muitos robôs possuem *interfaces* de comunicação como Ethernet/IP, Profinet, DeviceNet, entre outros. Essas interfaces permitem a comunicação direta do robô com outros dispositivos ou sistemas.

c- Protocolos de Comunicação: Os robôs podem usar protocolos de comunicação específicos para trocar informações com outros equipamentos ou sistemas. Por exemplo, o uso de protocolos como Modbus, OPC-UA, ou até mesmo comunicação por meio de APIs específicas fornecidas pelo fabricante.

1.5 Justificativa

Na grande maioria das plantas industriais, mesmo em ambientes com células

robóticas convencionais, o controle total de um sistema é realizado por um Controlador Programável (CP). No entanto, o planejamento do caminho e as operações específicas do robô são executados em plataformas de controle distintas, geralmente propriedade do fabricante. Esta abordagem proprietária requer profissionais altamente especializados para programar, operar e integrar os robôs com os comandos fundamentais de um CP, fato que resulta em uma escassez de profissionais quando surgem necessidades como ajustes urgentes na correção de movimentações, ou mesmo, uma flexibilização devido a necessidade de alteração de uma linha de produção. A escassez de técnicos qualificados em robótica, muitas vezes proficientes apenas em uma ou duas marcas específicas, contribui para atrasos e períodos de inatividade da planta fabril, ocasionando com isso paradas de produção e prejuízos.

Essa especialização limitada aumenta a dependência dos fabricantes em integradores de sistemas ou no suporte do fabricante do robô para resolver problemas e fornecer serviços. Além disso, em grande parte, toda a programação de movimentação de um robô é realizada através de um *Pendant*, que é um controle manual de posicionamento para os eixos do robô, o que resulta em uma lentidão já que a velocidade de programação está diretamente ligada ao operador.

Nesse sentido, o tema desse trabalho concentra-se na exploração de novos métodos de aprendizado de máquinas e sua aplicação na área da automação da manufatura mais utilizado atualmente, que é robótica industrial. Portanto, nesse trabalho a pesquisa aborda a utilização de algoritmos baseados em lógica Paraconsistente - LP para desenvolver células de aprendizagem de modo que formem estruturas para compor sistemas de aprendizagem por demonstração. Portanto, a relevância e a inovação desta investigação residem no emprego de algoritmos fundamentados em lógica não clássica, como a lógica paraconsistente, capaz de lidar com sinais contraditórios e incompletos. Essa abordagem pioneira busca gerar resultados promissores que possam contribuir significativamente para o avanço da Inteligência Artificial nesse campo específico utilizando como base a Aprendizagem por Demonstração (ApD) o que tornará a reprogramação de Robôs Industriais mais ágeis contribuindo para a eficiência do setor produtivo.

2 MATERIAIS E MÉTODOS

Conforme foi exposto anteriormente, no aprendizado por demonstração o sistema responsável pelo processo da aprendizagem deve enviar sinais repetitivos de informação ao robô e receber respostas do sucesso ou do retrocesso do aprendizado sequencialmente.

Depois de uma indicação que a máquina aprendeu totalmente a tarefa, o sistema deve então emitir sinais para que a máquina reproduza o conhecimento adquirido. Portanto nessa pesquisa, para que fosse possível acessar as informações de controle e movimentação do Robô industrial, foi inicialmente desenvolvida e implantada uma infraestrutura física com dispositivos computacionais que permitiu ao sistema responsável pelo processo o acesso aos dados e aos protocolos de troca de informação entre o homem e a máquina. Somente assim foi possível efetuar o desenvolvimento, testes e a criação do sistema responsável pelo processo de Aprendizagem por Demonstração (ApD) utilizando um conjunto de Células Neurais Artificiais Paraconsistente de aprendizagem (CNAPap).

Para implementar os algoritmos da LPA2v com o seu padrão matemático e lógico que constituíram as CNAPaps foi utilizado o *software* MATLAB, que através de uma plataforma OPC se comunica com um Controlador Programável (CP) de marca Schneider modelo M262.

A comunicação por meio da Teleoperação foi estabelecida através do protocolo ETHERNET utilizado pelo CP para efetuar a comunicação com um robô YASKAWA modelo MOTOMAN GP8. Portanto, no processo de troca de informações entre Homem-Máquina desenvolvido nessa pesquisa o sistema utiliza o protocolo ETHERNET onde faz o envio e a coleta de dados de movimentações como também, por consequência, gera os sinais para a aprendizagem das CNAPap no processo de aprendizagem por demonstração e opera na fase de imitação dentro do método ApD.

A seguir serão mostrados os detalhes dos materiais e equipamentos utilizados na construção do modelo de aprendizagem por demonstração usado na pesquisa.

2.1 Robô YASKAWA modelo MOTOMAN GP8

Foi utilizado nesse trabalho o Robô Industrial de marca YASKAWA modelo MOTOMAN GP8 que está instalado no OpenLab do SENAI SP, em São Caetano do Sul. O OpenLab é uma planta industrial aberta a pesquisas na qual diversas parcerias foram estabelecidas para estudos e implementações de inovações tecnológicas para a indústria nacional.

A robô marca YASKAWA modelo MOTOMAN GP8 tem seu projeto de construção compatível para as aplicações que exijam manuseio em alta velocidade e precisão. Este robô industrial é adequado para utilização em montagens e manuseio de peças, possui carga útil de até 8Kg, sendo encontrado em diversas áreas da manufatura industrial.

Nas figuras 12 e 13 estão apresentadas imagens da planta com foco no robô industrial utilizado nesse estudo.

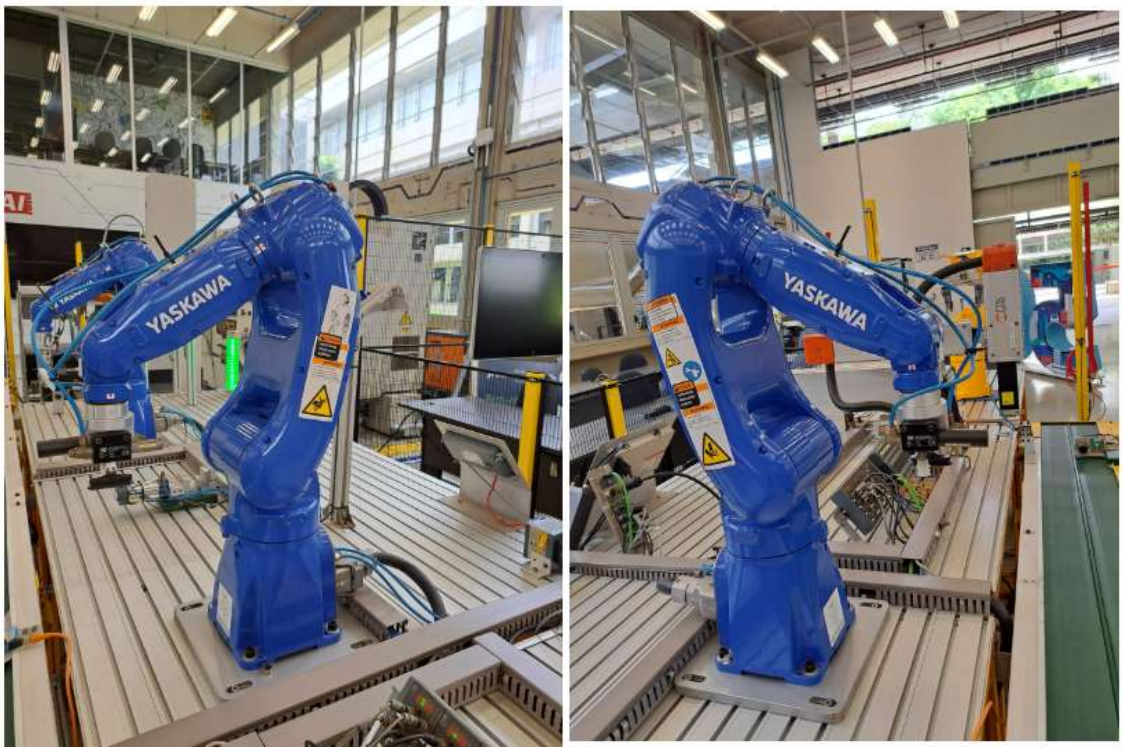


Figura 12 - OpenLab e robô YASKAWA modelo MOTOMAN GP8.

Figura 13 - OpenLab e robô YASKAWA modelo MOTOMAN GP8 com placa de dados de suas características.

A marca YASKAWA pertence a uma das três maiores fabricantes mundiais e tem aproximadamente 300.000 robôs instalados nos diferentes pátios fabris. O modelo escolhido utiliza no seu acionamento o controlador YRC1000micro que permite a interação com um *Smart Pendant* com tela sensível ao toque.

Na figura 14 é apresentado o robô utilizado no projeto e o seu respectivo *Teach Pendant* (TP) para o controle e a programação do posicionamento dos 6 eixos do robô pelo operador.

Figura 14 - Robô YASKAWA modelo MOTOMAN GP8 e seu *Teach Pendant*.

Segundo o seu fabricante, o robô GP8, desde que todos os eixos estejam em limite máximo, apresentará os *ranges* de ângulos e de velocidade para cada um dos 6 eixos, conforme mostrado na figura 15.

Axes	Maximum motion range		Maximum speed		Allowable moment		Allowable moment of inertia	
	degrees		°/sec		N·m		kg·m²	
	GP7	GP8	GP7	GP8	GP7	GP8	GP7	GP8
S	±170	±170	375	455	-	-	-	-
L	+145/-65	+145/-65	315	385	-	-	-	-
U	+190/-70	+190/-70	410	520	-	-	-	-
R	±190	±190	550	550	17	17	0.5	0.5
B	±135	±135	550	550	17	17	0.5	0.5
T	±360	±360	1000	1000	10	10	0.2	0.2

Figura 15 - *Range* com ângulos e pulsos/seg do robô MOTOMAM GP8.
Fonte: Manual YASKAWA MOTOMAN – GP8

Cada eixo do robô possui um *range* específico em ângulo e pulsos do seu *encoder* e como essa relação é proporcional, então é possível utilizar uma regra de três para determinar os valores correspondentes.

As variáveis θ e $Pulso$ representam o ângulo e o valor em pulso desejado.

Ângulo	**Pulso**
θ_{range}	$Pulso_{range}$
θ	$Pulso$

Onde, θ_{range} é o *range* angular do eixo do motor, ou seja, a diferença entre o maior e o menor alcance angular. Proporcionalmente, $Pulso_{range}$ é a diferença entre o maior e o menor pulso no *encoder* do motor do robô. Os valores dos *ranges* devem estar em módulo, pois o robô utiliza pulsos positivos e negativos no *encoder*, e seus valores correspondentes em ângulos também compreendem uma faixa positiva e negativa. Substituindo, fica:

Ângulo	**Pulso**
θ_{range}	$Pulso_{range}$
θ	$Pulso$

Como o ângulo mínimo de nenhum motor do robô inicia em 0, é então necessário acrescentá-lo (em módulo, pois o valor é negativo) ao ângulo desejado.

Ângulo	**Pulso**
$\lvert\theta_{max}\rvert + \lvert\theta_{min}\rvert$	$\lvert Pulso_{max}\rvert + \lvert Pulso_{min}\rvert$
θ	$Pulso$

Como o ângulo mínimo foi adicionado ao ângulo a ser convertido, então o valor correspondente em pulso deve estar subtraído do valor do pulso mínimo.

Ângulo	**Pulso**
$\lvert\theta_{max}\rvert + \lvert\theta_{min}\rvert$	$\lvert Pulso_{max}\rvert + \lvert Pulso_{min}\rvert$
$\theta + \lvert\theta_{min}\rvert$	$Pulso - Pulso_{min}$

Dessa forma:

$$(Pulso - Pulso_{min}).(|\theta_{max}| + |\theta_{min}|) = (|Pulso_{max}| + |Pulso_{min}|).(\theta + |\theta_{min}|)$$

$$Pulso - Pulse_{min} = \left(\frac{(|Pulso_{min}| + |Pulso_{max}|) \,.\, (\theta + |\theta_{min}|)}{|\theta_{max}| + |\theta_{min}|}\right)$$

Portanto, a relação entre ângulos e pulsos é regida pela equação final a seguir:

$$Pulse = \left(\frac{(|Pulse_{min}| + |Pulse_{max}|) \,.\, (\theta + |\theta_{min}|)}{|\theta_{max}| + |\theta_{min}|}\right) + Pulse_{min}$$

A fim de evitar possíveis colisões com o meio físico, foram utilizados os limites angulares e sua conversão em pulsos para os eixos que constam da tabela 1.

Tabela 2 - Limites em ângulos e pulsos de cada eixo do robô

EIXOS	LIMITES ANGULARES		LIMITES EM PULSOS	
	Máximo	Mínimo	Máximo	Mínimo
S	150	-150	153600	-153600
L	150	-40	157772	-72818
U	145	-60	131982	-54614
R	180	-180	153600	-153600
B	120	-120	87381	-87381
T	180	-180	83674	-83679

Portanto, os valores normalizados para aprendizagem e memorização da posição do robô inseridos no sistema ficaram dentro dos *ranges* pré-estabelecidos pelo fabricante evitando colisões com o ambiente da planta industrial no qual está inserido.

Na figura 16 estão indicados a posição dos 6 eixos do robô MOTOMAN GP8.

Figura 16 - Posição dos 6 eixos do robô MOTOMAM GP8.

Na figura 17 é apresentado o espaço de atuação dos 6 eixos do robô MOTOMAN GP8 com dados importantes de seus raios de ação para evitar colisões causando possíveis acidentes em máquinas e pessoas.

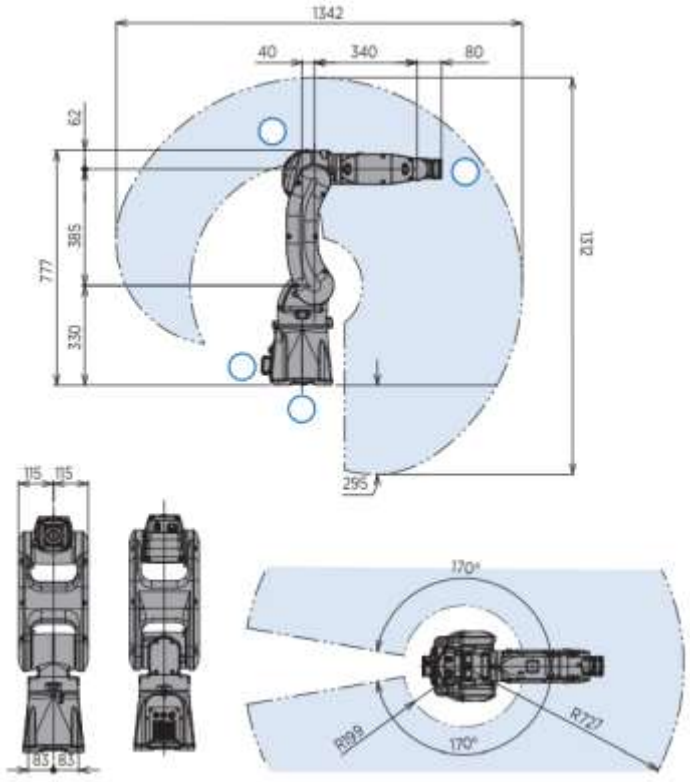


Figura 17 - Espaços de atuação do robô MOTOMAM GP8
Fonte: Manual YASKAWA MOTOMAN – GP8

O dispositivo utilizado na programação de robôs industriais é o *Teach Pendant*, que consiste em uma *interface* homem-máquina (IHM) permitindo a

programação e controle passo a passo de robôs industriais. Sendo composto por conectores, botões, membranas, cabos e circuitos eletrônicos, este conjunto tem a função de facilitar o controle e a programação dos movimentos de cada eixo de um robô. Como é utilizado por um operador especialista, seu tempo de programação depende efetivamente da habilidade do programador. Na sequência, as características e funções do *Smart Teach Pendant* integrado ao controlador YRC1000micro utilizado na aplicação estão sendo apresentado na figura 18.

Specifications Controller YRC1000micro	
Configuration	Open structure
Dimensions	425 (W) x 125 (H) x 280 (D) mm – excluding protrusions
Weight	10.5 kg (without top mount box)
Protection class	IP20
Cooling system	Direct cooling
Ambient temperature	During operation: 0°C to +40°C During storage: –10°C to +60°C
Relative humidity	90 % max. (non-condensing)
Power supply	Single-phase 200/230 VAC (+10 to –15%), 50/60 Hz (±2 %)[1]; three-phase 220/220 VAC (+10 to –15 %), 50/60 Hz (± 2%)
Digital I/Os	External IO (hardware, standard max): 8 inputs / 8 outputs (transistor 100 mA)[2] Standard signal allocation: General Purpose IO: 1 input / 1 output Specific IO: 7 inputs / 7 outputs
Programming capacity	JOB: 200,000 steps, 10,000 instructions; CIO ladder: 1,500 steps
Expansion slots	PCI express, 2 slots
LAN (Connections to host)	1 (10BASE-T/100BASE-TX)
Safety category	4, PLd

[1] Manipulators operated by single-phase power supply: MOTOMAN GP7, GP8, MotoMINI.
[2] If it is necessary to expand the number of points, a filed bus board is required

Specifications programming pendant	
Dimensions	152 (W) x 299 (H) x 53 (D) mm
Weight	0.730 kg
Material	Reinforced plastics
Operation device	Select keys, axis keys, numerical/application keys, mode selector switch with keys (mode: teach, play, and remote), emergency stop button, enable switch, SD card interface device, USB port (1 port)
Display	5.7-inch color LCD, touch panel 640 x 480 pixels
IECpProtection class	IP54

Figura 18 - Características do *Smart Pendant* do Controlador YRC1000micro.
Fonte: Manual YASKAWA MOTOMAN – GP8

2.2 Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v

O conjunto de *softwares* criados por diversos aplicativos para estabelecer o método de Aprendizado por demonstração (APD) no robô Industrial YASKAWA MOTOMAN – GP8 é construído através dos algoritmos da Lógica Paraconsistente Anotada, portanto, é denominado de Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v. As seções a seguir irão descrever os métodos para o desenvolvimento e sua construção.

2.2.1 Programação do Robô MOTOMAN GP8

Devido ao seu porte, a maneira de posicionar manualmente o Robô Industrial MOTOMAN GP8 para o aprendizado por demonstração, que é um método que exige repetições, se mostrou dificultoso. Dessa forma, para a aplicação desenvolvida na pesquisa foi necessário a criação no robô através do *seu Pendant,* os programas “CNAP”, para posicionamento do robô e o “ENVIA_POSICAO_CP_5_EIXOS_ROBO”. Este último é utilizado para envio dos dados do Controlador do MOTOMAN GP8 ao programa CNAPap desenvolvido no MATLAB, passando pela transição realizada pelo Controlador Programável (CP) Schneider M262.

O programa CNAP, cujo código completo está sendo apresentado no Apêndice C, possui as seguintes características principais seguindo o fluxograma apresentado na figura 19:

1) Inicia o robô em modo automático e o envia para a posição inicial com velocidade de movimentação de 5 rad/s;

2) Aguarda o envio pelo CP de um valor diferente de “0” para que o robô se desloque para a posição final (*set point*).

3) Quando é recebida a ordem vinda do CP, ou seja, um valor diferente de “0” nos 2 últimos bytes do eixo “T”, é realizada a movimentação do robô da sua posição

inicial para a posição final desejada carregada em “P002”, também em 5 rad/s.

4) Após a finalização da movimentação é realizada a chamada da função ENVIA_POSICAO_CP_5_EIXOS_ROBO.

5) No retorno do programa ENVIA_POSICAO_CP_5_EIXOS_ROBO o CNAP retorna a posição inicial com velocidade de 5 rad/s e fica aguardando um novo comando do CP.

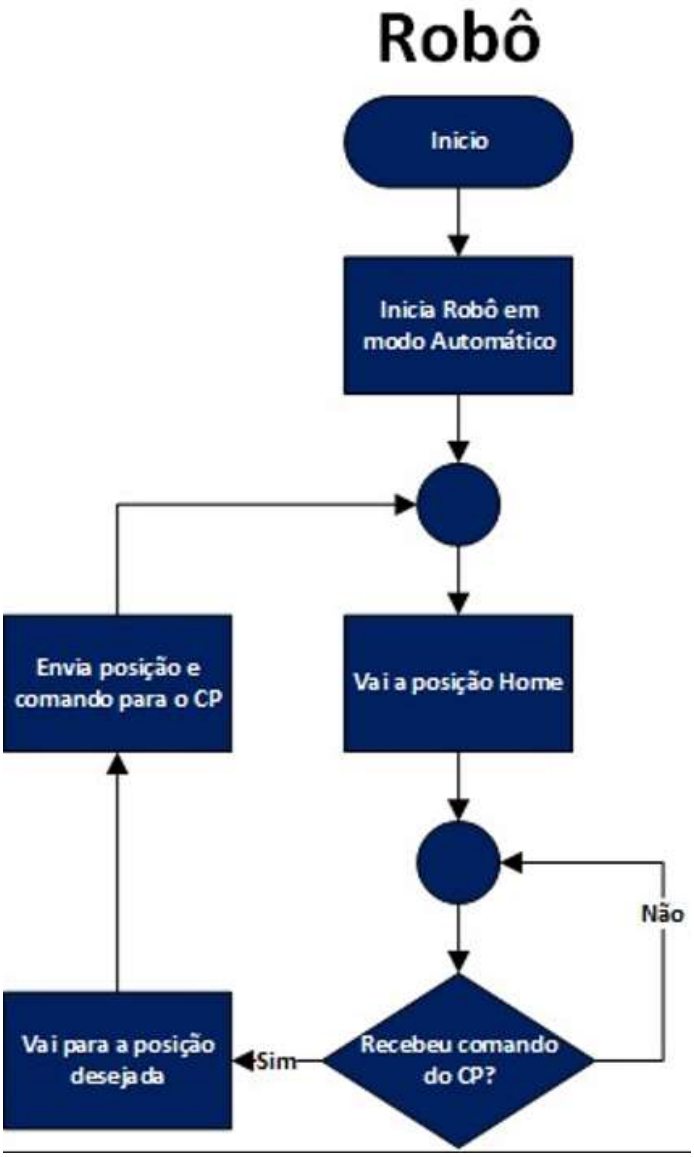


Figura 19 - Fluxograma dos programas CNAP e ENVIA_POSICAO_CP_5_EIXOS_ROBO.

Ao ser chamado pelo programa “CNAP” instalado no robô YASKAWA MOTOMAN – GP8, a função “ENVIA_POSICAO_CP_5_EIXOS_ROBO”, cujos códigos são encontrados no Apêndice D, executa os seguintes passos principais:

1) Após a movimentação do robô de sua posição inicial para a final realizada pelo programa “CNAP”, a função “ENVIA_POSICAO_CP_5_EIXOS_ROBO”, ao ser chamada, guarda os dados em pulsos de cada grau de liberdade, ou seja, dos eixos ”S”, “L”, “U”, “R”, “B” e “T”, respectivamente nas variáveis P000 (1) a P000(6).

2) Esses valores são respectivamente em número de pulsos: Eixos “S” com 96160 correspondente a 94°, “L” com 72818 correspondente a 30°, “U” com 54613 correspondente a 60°, “R” com 62817 correspondente a 80°, “B” com 50972 correspondente a 70° e “T” utilizado como confirmação com valor 1.

3) A esses dados em pulsos são somados os valores negativos mínimos de cada grau de liberdade (tabela 2), para que em nenhuma hipótese sejam transmitidos valores menores que “0”. Esses dados são alocados, de acordo com o programa, nas variáveis DWORD: D11, D12, D13, D14, D15 e D16, respectivamente para os eixos ”S”, “L”, “U”, “R”, “B” e “T”.

4) Nesse ponto as variáveis DWORD estão carregadas com os seguintes valores:
    - “S” (D11) com 96160 +153600 (valor negativo máximo) = 249760;
    - “L” (D12) com 72818 + 72818 (valor negativo máximo) = 145636;
    - “U” (D13) com 54613 + 54614 (valor negativo máximo) = 109227;
    - “R” (D14) com 62817 + 153600 (valor negativo máximo) = 221867;
    - “B” (D15) com 50972 + 83679 (valor negativo máximo) = 138353;
    - “T” (D16) utilizado como confirmação com valor 1.

5) Cada variável DWORD, D11, D12, D13, D14, D15 e D16 é então alocada em 4 variáveis do tipo Byte, B24 a B47, totalizando 24 bytes para os eixos ”S”, “L”, “U”, “R”, “B” e “T” a serem transmitidos, conforme a tabela 3 a seguir.

Tabela 3 – Alocação das variáveis B24 a B47 para os eixos do robô.

B027								B026								B025								B024							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0

B035								B034								B033								B032							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1

B039								B038								B037								B036							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	1	0	1	0	1	1

B043								B042								B041								B040							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1

B031								B030								B029								B028							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	1	0	0

B047								B046								B045								B044							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

6) Carregados os dados de posicionamento somados aos valores máximo negativos dos eixos ”S”, “L”, “U”, “R”, “B” e “T”, o programa “ENVIA_POSICAO_CP_5_EIXOS_ROBO” realiza a alocação de B24 a B47 nas Outputs General OG#(03) a OG#(26) como demonstrado no Apêndice D. Esses dados com o valor de cada eixo são enviados as CNAPap residentes no MATLAB através do CP Schneider M262.

7) Finalizado o envio, é contado 1 segundo e em seguida zerado B044 que é novamente enviada informando o término da operação.

8) Ao fim do processo, a função “ENVIA_POSICAO_CP_5_EIXOS_ROBO” retorna ao programa principal do robô, ou seja, retorna ao “CNAP”

Como o robô MOTOMAN GP8 utilizado na aplicação tem disponível apenas 48

bytes, sendo que são necessários para comunicação de Leitura 4 *bytes* por eixo e 1 *byte* de confirmação de finalização de comunicação, ou seja 25 no total e o mesmo processo se verifica para a escrita, nesse caso, optou-se pela utilização de 5 Eixos que foram os eixos ”S”, “L”, “U”, “R”, e “B”, ficando os *bytes* obrigatórios de transmissão do Eixo “T” utilizados como confirmação. Com mais detalhes os respectivos eixos são apresentados na figura 20.

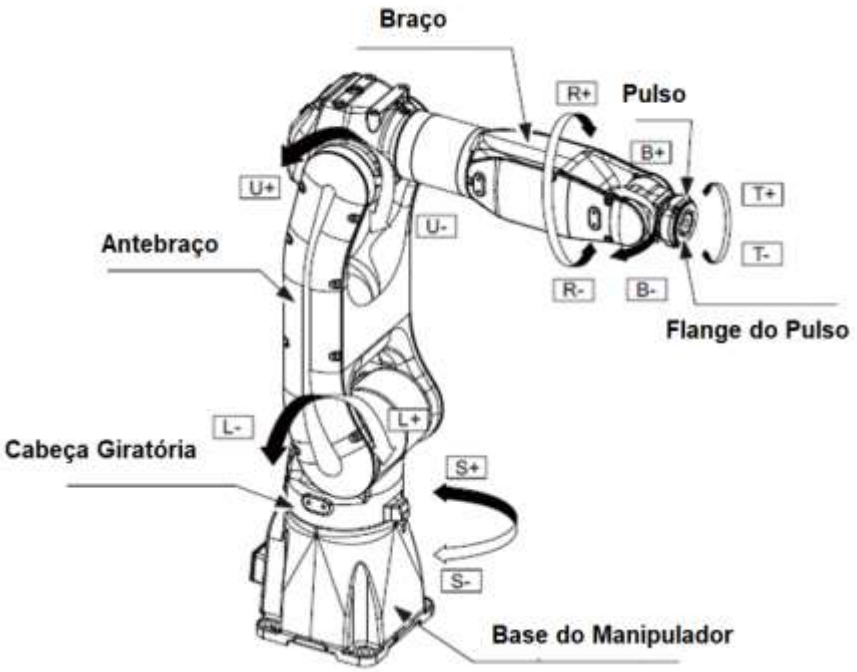


Figura 20 - Eixos do robô YASKAWA MOTOMAM GP8.

2.2.2 CNAPap criada no MATLAB

Nesta pesquisa, as CNAPap referentes aos eixos ”S”, “L”, “U”, “R”, “B” e “T” tiveram seus algoritmos implementados no *software* MATLAB cuja integração a um Controlador Programável Schneider modelo M262 através da Plataforma de Comunicação Aberta – Arquitetura Unificada (OPC UA) o que possibilitou a implementação de um Sistema de Aprendizagem por Demonstração (ApD), para aprendizagem e memorização de movimentações dos eixos de um robô industrial do fabricante YASKAWA modelo MOTOMAN - GP8.

Na figura 21 está apresentado o fluxograma do programa CNAPap que foi desenvolvido para essa finalidade.

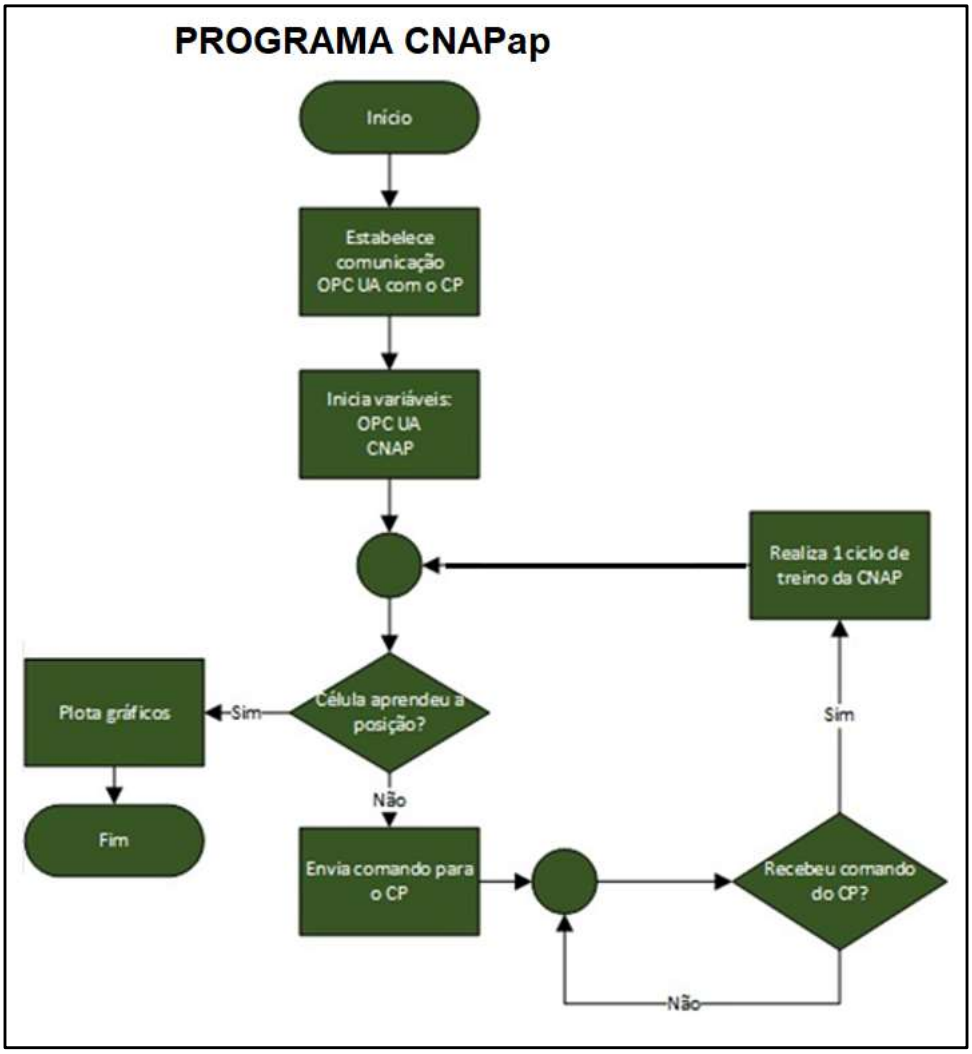


Figura 21 - Fluxograma do Programa CNAPap.

Os principais pontos da programação do programa CNAPap desenvolvido no MATLAB estão sendo apresentados a seguir com as respectivas observações, sendo que o programa completo pode ser encontrado no Anexo E.

1) Estabelece comunicação OPC UA, iniciando o IP (10.10.0.243) e a porta (4840) do notebook conectando a aplicação ao Controlador Programável Schneider M262.

2) Procura as variáveis de escrita e leitura OPC UA (Variaveis_CLP_wr e Variaveis_CLP_rd) provenientes do CP anexando-as a comunicação do programa.

3) Inicia as variáveis de escrita Robot_S_wr, Robot_L_wr , Robot_U_wr , Robot_R_wr , Robot_B_wr e Confirma_wr tendo seus respectivos valores zerados.

4) São atribuídos os *set points* de aprendizagem das CNAPap e pulsos mínimos correspondentes aos eixos ”S”, “L”, “U”, “R”, “B” e “T”.
   - Set_point_S = 96160; Valor_pulso_minimo_S = 153600;
   - Set_point_L = 72818; Valor_pulso_minimo_L = 72818;
   - Set_point_U = 54613; Valor_pulso_minimo_U = 54614;
   - Set_point_R = 68267; Valor_pulso_minimo_R = 153600;
   - Set_point_B = 50972; Valor_pulso_minimo_B = 87381;

5) É inserido o valor do Fator de Aprendizagem.

6) Entra com o padrão de entrada inicial para os eixos ”S”, “L”, “U”, “R”, “B” e “T”, variáveis ui_S; ui_L; ui_U; ui_R; ui_B;

7) Inicia o treino das CNAPap, enquanto a aprendizagem não é concluída, envia os dados de ordem de movimentação ao robô, zerando as variáveis de escrita Robot_S_wr, Robot_L_wr, Robot_U_wr, Robot_R_wr, Robot_B_wr e enviando o valor 65535 por Confirma_S_wr

8) Valores de escrita são enviados via OPCUA para o Controlador Programável Schneider M262 utilizando o IP e porta cadastrados.

9) Aguarda receber comando do Robô.

10) Lê as variáveis de posição enviadas pelo robô referente aos eixos ”S”, “L”, “U”, “R”, “B” e “T” via OPC UA do CP e converte para uma matriz de 5x1 (5 eixos), o eixo “T” chega como ordem de leitura de dados.

11) Converte a matriz [5,1] em número inteiro de 32 bits (4 bytes)

12) Inicia o Ciclo de Treino retirando o valor incrementado como valor máximo negativo no robô e armazena nas variáveis Robot_variaveis_rd
    - Robot_S_rd = int32(Valores_rd(4,1)) - Valor_pulso_minimo_S;
    - Robot_L_rd = int32(Valores_rd(2,1)) - Valor_pulso_minimo_L;
    - Robot_U_rd = int32(Valores_rd(6,1)) - Valor_pulso_minimo_U;
    - Robot_R_rd = int32(Valores_rd(3,1)) - Valor_pulso_minimo_R;
    - Robot_B_rd = int32(Valores_rd(1,1)) - Valor_pulso_minimo_B;

13) Retorna ao algoritmo das CNAPap para o eixos ”S”, “L”, “U”, “R”, “B” e “T”.

14) Envia pulso após a finalização do treino de aprendizagem para o Robô realizar uma nova movimentação, variável Confirma_wr enviada com valor 65535.

15) Carrega valores para escrita OPC UA.

16) Envia valores para o CP

17) Envia o valor Confirma_wr = 0;

18) Carrega valores para escrita OPC UA.

19) Envia valores para o CP

20) Enquanto as células não aprendem os valores padronizados, retorna ao passo 9. Se aprendem vão para o passo 21.

21) Plota gráficos de Aprendizado e modelos cinemáticos com suas respectivas posições.

22) Finaliza o programa e fica preparado para enviar os dados aprendidos ao robô para validação do aprendizado (imitação).

Nesse trabalho, os dados gerados para o aprendizado das CNAPaps dos graus de liberdade ”S”, “L”, “U”, “R”, “B” e “T”, são apresentados de acordo com o número de pulsos e ângulos a seguir:

a- CNAPap “S” – 94° e 96160 pulsos;
b- CNAPap “L” – 30° e 72818 pulsos;
c- CNAPap “S” – 60° e 54613pulsos;
d- CNAPap “S” – 80° e 62267 pulsos;
e- CNAPap “S” – 70° e 50972 pulsos;
f- CNAPap “S” – envio de confirmação.

2.2 Configuração do CP com os conceitos da IEC 61131

O Machine Expert Logic Builder é o *software* de programação do CP Schneider M262 utilizado nesse trabalho, o qual foi utilizado como ferramenta de comunicação entre o MATLAB e o CP via OPC UA, e entre o CP e o robô YASKAWA MOTOMAN GP8 através do protocolo Ethernet. O Machine Expert possibilita a utilização das cinco linguagens de programação descritas pela norma IEC 61131-3, ou seja a linguagem Sequenciamento Gráfico de Funções (SFC), usada para estruturar a organização interna do programa, e quatro linguagens de programação interoperáveis: Lista de Instruções (IL), Diagrama Ladder (LD), Diagrama de Blocos (FBD) e Texto Estruturado (ST). A estrutura do *software* foi elaborada atendendo as diretrizes da norma IEC 61131-3 e IEC 61499, utilizando os conceitos de decomposição em elementos lógicos, modularização e técnicas de *software* como programação orientada a objeto, onde permite que cada programa seja estruturado de forma a melhorar a sua reutilização, reduzindo erros e melhorando a programação e eficiência para o usuário. Para isso utiliza-se o processo de criação de blocos de programação denominados de “Unidades de Organização de Programas” (POUs) que segundo as normas são:

- Funções;
- Blocos funcionais;
- Programas.

Estas POUs são basicamente as formas de implementação de programas no

CP, através da associação de variáveis e instruções, podendo uma POU ser parte integrante de outra POU. Segundo a norma IEC 61131-3, deve ser estabelecida uma hierarquia entre as POUs seguindo os seguintes princípios:

- Funções somente podem conter Funções;
- Blocos Funcionais podem conter outros Blocos Funcionais e Funções;
- Programas podem conter qualquer tipo de POU.

O conceito de função, segundo a norma IEC 61131-3 deve apresentar em sua forma organizacional as seguintes características fundamentais:

- Podem ser utilizados como elementos comuns para a definição de novas POUs.
- São elementos reutilizáveis de *software*.
- Só existem em tempo de execução.
- Deve ser feita a declaração do tipo antes do uso.
- As respectivas chamadas em linguagens textuais devem ser feitas pelo nome e em linguagens gráficas pelo bloco.

A linguagem de Blocos Funcionais é baseada no item da norma que se refere à reutilização de código e modularidade. Dentre os seus fundamentos tem como características principais:

- Ser um elemento encapsulado de *software* que deve apresentar a possibilidade de reutilização.
- Somente tipos de dados definidos podem ser utilizados por interfaces externas.
- Valores das variáveis são mantidos entre uma execução e outra.
- Deve apresentar variáveis de entrada e saída além de poderem apresentar em seu código a utilização de variáveis internas.
- Para cada instância declarada é reservada uma área de memória.
- Uma instância de um Bloco Funcional pode ser utilizada na declaração de outro Bloco Funcional ou Programa.

Os Programas são considerados a maior forma de construção de uma POU, podendo nele conter agrupamentos de Funções e Blocos Funcionais. Um Programa segundo a norma, pode ser o elemento responsável por executar uma função de controle, ou ser declarado a nível de recurso para ser utilizado por um outro programa

principal, como por exemplo, a criação de uma biblioteca que será manipulada por um programa com a função de controle de uma variável de processo.

O Machine Expert Logic Builder, utilizado nesse trabalho, é uma ferramenta de programação projetada para configurar, desenvolver e comissionar programas para Controladores Programáveis (CP) utilizando esses conceitos, acrescidos pelos seguintes detalhes lógicos:

a- Projeto: contém detalhes sobre a configuração do CP e módulos de expansão associados ao projeto, o código fonte de um programa, símbolos, comentários, documentação e outras informações relacionadas.
b- Aplicativo: Contém as partes do projeto que são baixadas para o controlador lógico, incluindo o programa compilado, informações de configuração de *hardware* e dados que não são do programa (propriedades, símbolos e comentários do projeto).
c- Programa: O código fonte compilado que é executado no CP.
d- POU (Unidade de Organização do Programa): O objeto reutilizável que contém uma declaração de variável e um conjunto de instruções usadas em um programa.

Na figura 22 é apresentado o diagrama com os estágios típicos do desenvolvimento de um projeto com a utilização do Machine Expert Logic Builder tendo como base a IEC 61131-3, com as respectivas guias de Configuração, Programação e início da execução.

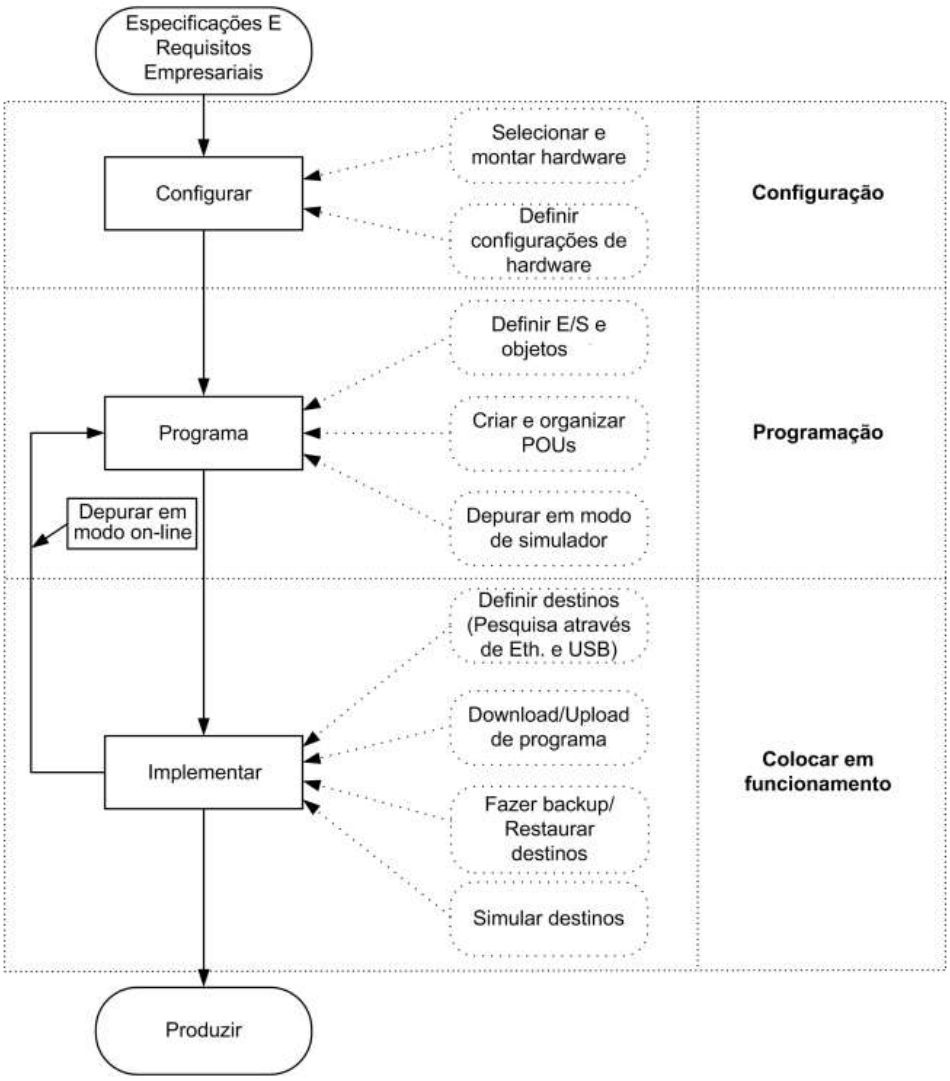


Figura 22 - Fluxograma do Machine Expert Logic Builder.
Fonte: Manual Schneider

O *software* Machine Expert usa os seguintes termos em sua estrutura:

a- Projeto: Um projeto EcoStruxure Machine Expert - Basic contém detalhes sobre o a configuração do CP e seus respectivos módulos de expansão associados ao projeto, o código fonte de um programa, símbolos, comentários, documentação entre outras informações.

b- Aplicativo: Contém as partes do projeto que são baixadas para o CP, incluindo o programa compilado, informações de configuração de *hardware* e dados que não são do programa (propriedades, símbolos e comentários do projeto).

c- Programa: O código fonte compilado que é executado no CP.

d- POU: O objeto reutilizável que contém uma declaração de variável e um conjunto

de instruções usadas em um programa.

Como elemento principal de reutilização de código seguindo as diretrizes da IEC 61131-3. Através dos comandos *Application* e *Add Object* é possível ser inserida uma Unidade de Organização de Programa (POU), como demonstrado na figura 23.

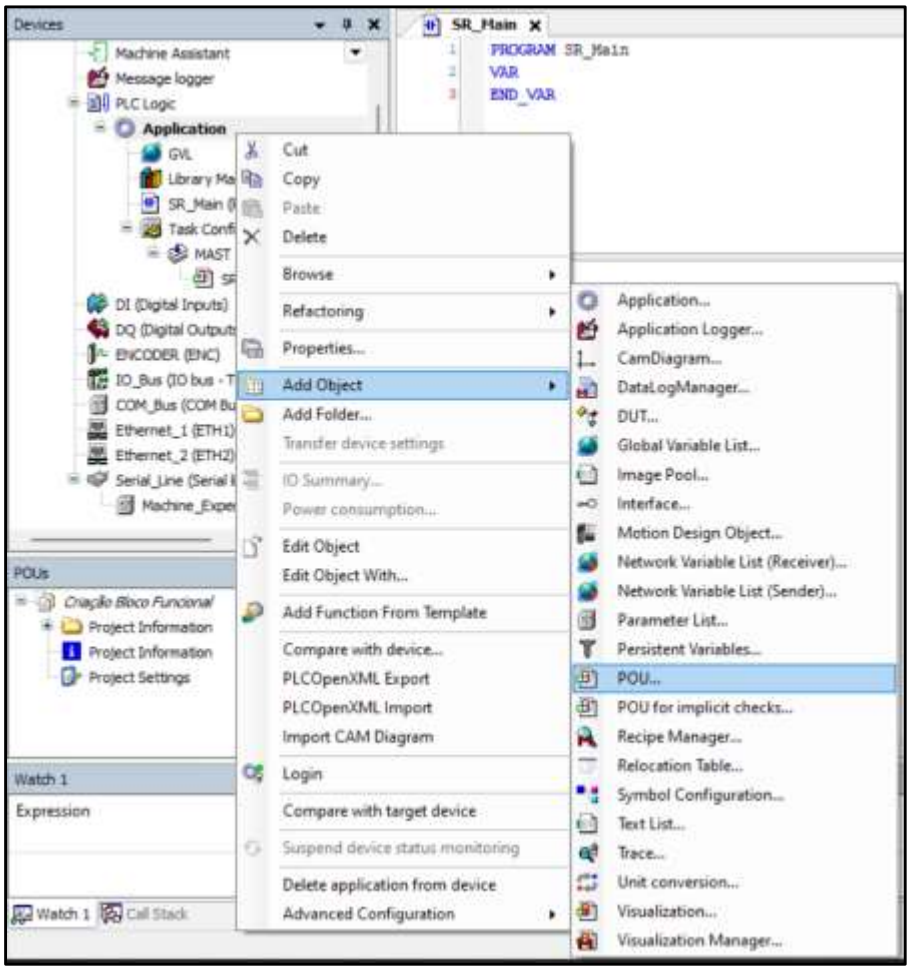


Figura 23 - Tela de criação de uma POU.

Após a criação da POU, esta deve ser referenciada através da atribuição de um nome ao Bloco Funcional, conforme a figura 24.

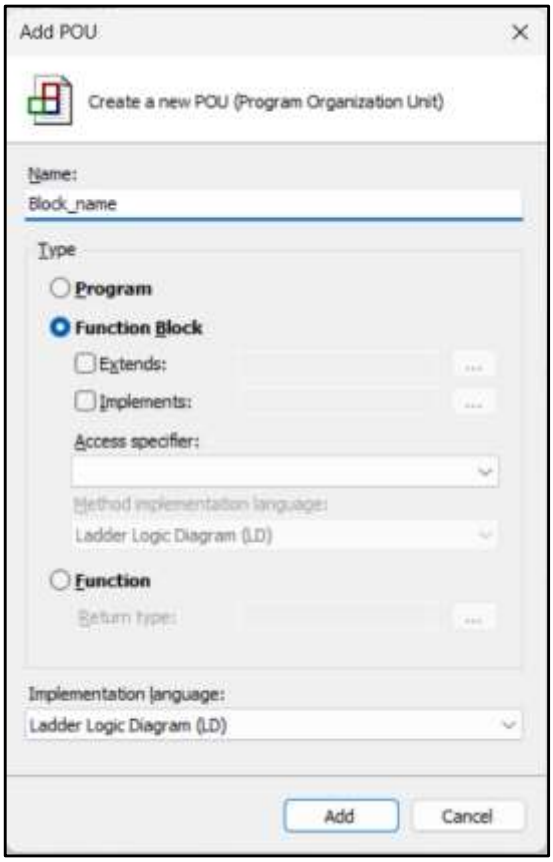


Figura 24 - Tela atribuição de nome a um Bloco Funcional.

Ao ser adicionada uma POU ao programa principal se faz necessário a declaração das variáveis internas, de entrada e saída do Bloco Funcional (BF), conforme o modelo de tela apresentado na figura 25 a seguir.

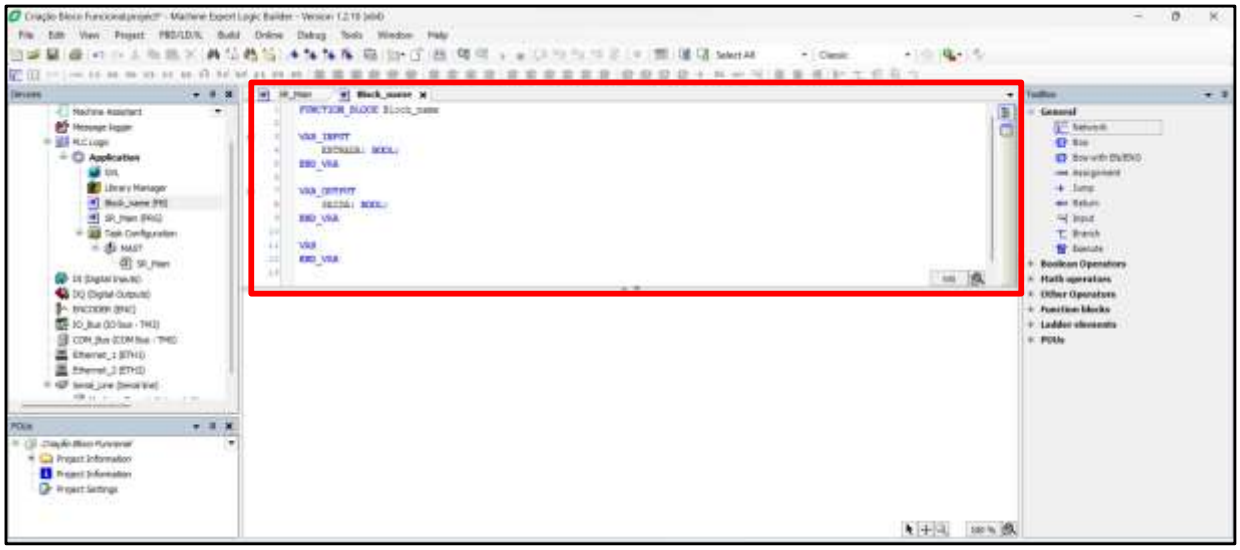


Figura 25 - Declaração das variáveis de um Bloco Funcional.

A seguir, na figura 26 é apresentado o programa principal com a criação de uma *Network* em linguagem *Ladder* na qual é realizado o chamamento do bloco criado.

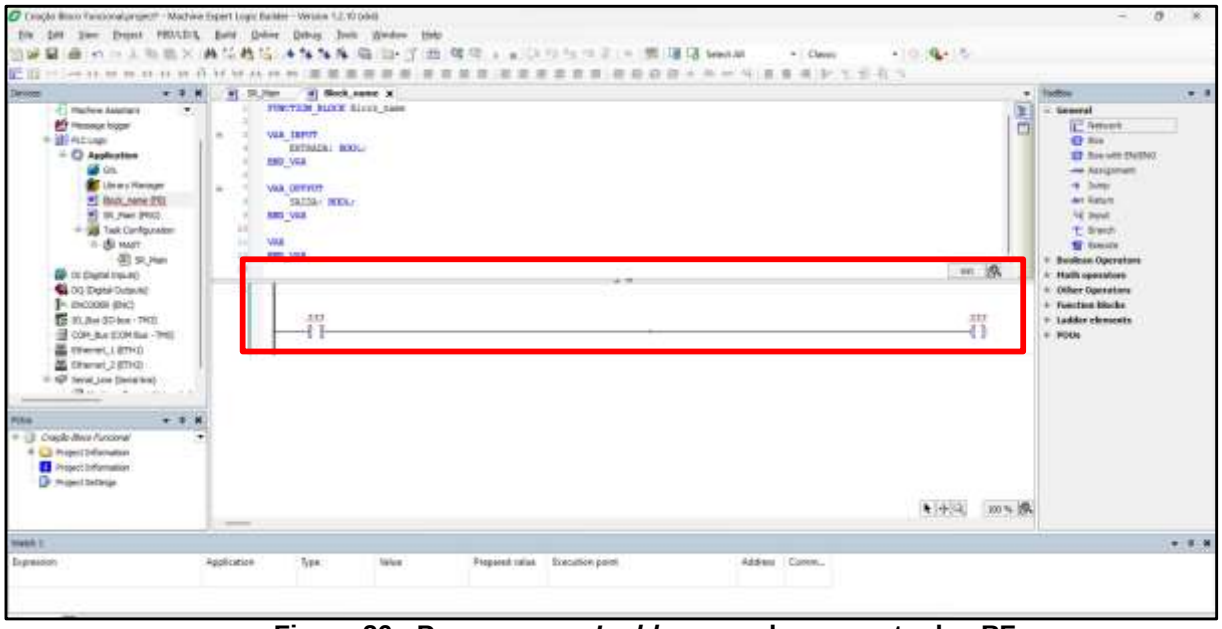


Figura 26 - Programa em *Ladder* para chamamento dos BFs.

Na figura 27 é apresentada a declaração dos blocos funcionais chamados pelo programa principal (MAIN) na aplicação desenvolvida.

```
1       PROGRAM  SR_Main
2       VAR
3           FB_DWORD_TO_2WORDS_S :  FB_DWORD_TO_2WORDS ;
4           FB_DWORD_TO_2WORDS_L :  FB_DWORD_TO_2WORDS ;
5           FB_DWORD_TO_2WORDS_U :  FB_DWORD_TO_2WORDS ;
6           FB_DWORD_TO_2WORDS_R :  FB_DWORD_TO_2WORDS ;
7           FB_DWORD_TO_2WORDS_B :  FB_DWORD_TO_2WORDS ;
8           FB_DWORD_TO_2WORDS_T :  FB_DWORD_TO_2WORDS ;
9           FB_2WORD_TO_DWORD_S :  FB_2WORD_TO_DWORD ;
10          FB_2WORD_TO_DWORD_L :  FB_2WORD_TO_DWORD ;
11          FB_2WORD_TO_DWORD_U :  FB_2WORD_TO_DWORD ;
12          FB_2WORD_TO_DWORD_R :  FB_2WORD_TO_DWORD ;
13          FB_2WORD_TO_DWORD_B :  FB_2WORD_TO_DWORD ;
14          FB_2WORD_TO_DWORD_T :  FB_2WORD_TO_DWORD ;
15      END_VAR
16
```

Figura 27 - Declaração dos BFs utilizados no programa principal.

Para a comunicação entre o robô YASKAWA MOTOMAN GP8 e o Controlador Programável Schneider M262 foram criados os blocos “S”, “L”, “U”, “R”, “B” e “T”, que têm a função de transformar 2 WORDS de 16 bits em uma DWORD de 32 bits.

A figura 28 apresenta o bloco funcional construído para o grau de liberdade “S”, o FB_2WORD_TO_DWORD_S, os demais são encontrados no Apêndice H.

Figura 28 - FB_2WORD_TO_DWORD eixo “S”.

A declaração das variáveis de entrada em WORD e saída em DWORD da POU construída a partir do bloco FB_2WORD_TO_DWORD, bem como as suas respectivas variáveis internas em bits, estão sendo apresentadas no Apêndice F, CP - Bloco Funcional DWORDS TO 2WORD.

Cada bloco FB_2WORD_TO_DWORD_X, onde X representa os 6 eixos ou graus de liberdade do robô, encapsula 2 Blocos do tipo WORD_AS_BIT1 e WORD_AS_BIT2 que convertem variáveis de entrada WORD em saídas em bits. Esses bits são alocados como entradas de um bloco do tipo BIT_AS_DWORD. A conversão direta entre WORDS e DWORDS não é possível tendo, portanto a necessidade de utilizar tal artifício. A figura 29 demonstra essa conversão.

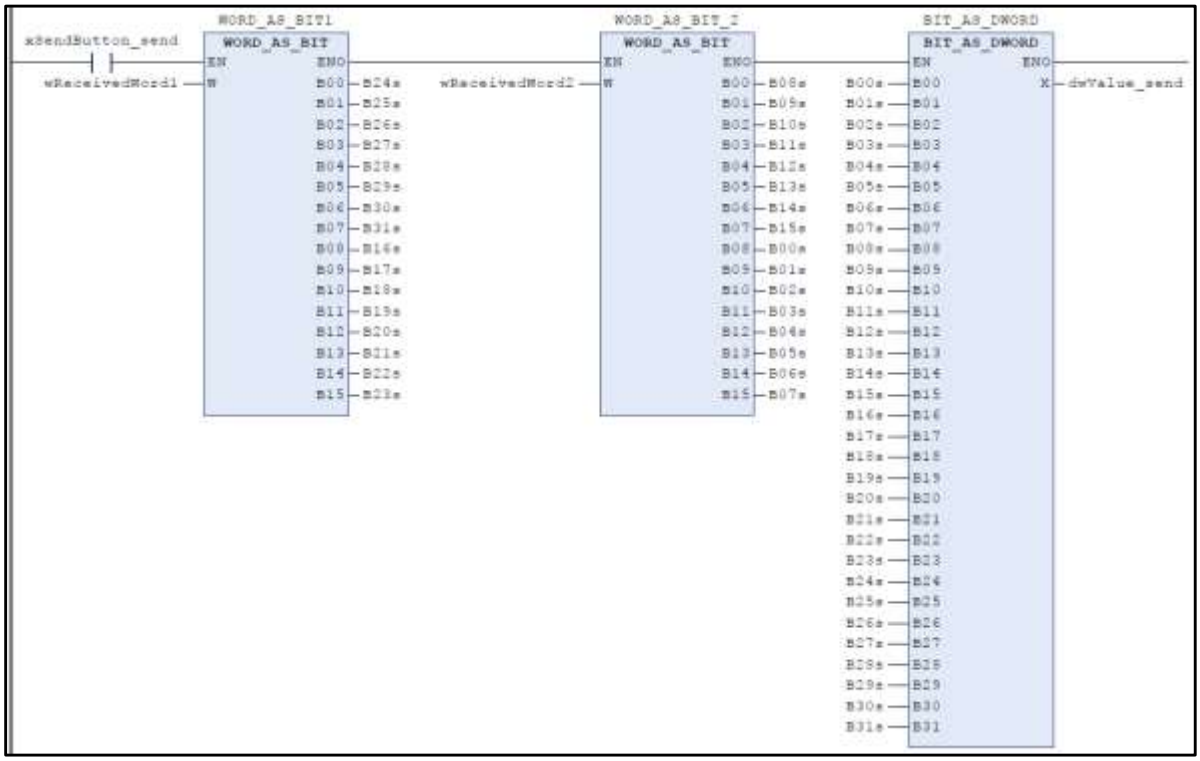


Figura 29 - Conversão de 2 WORDS em DWORD para o FB_2WORD_TO_DWORD.

Da mesma forma, para a comunicação em sentido contrário, ou seja, entre o CP Schneider M262 e o robô MOTOMAN GP8, foram criados os blocos “S”, “L”, “U”, “R”, “B” e “T” que tem a função de transformar uma DWORD de 32 bits em 2 DWORD de 16 bits.

A figura 30 apresenta o bloco funcional do tipo FB_DWORD_TO_2WORD_S. Os demais blocos criados para completar os 6 eixos do robô podem ser encontrados no Apêndice H.

Figura 30 - FB_DWORD_TO_2WORD_S.

A declaração das variáveis de entrada em DWORD e saída em WORD do bloco FB_DWORD_TO_2WORD, bem como as suas respectivas variáveis internas

em bits podem ser encontradas no Apêndice G.

O bloco FB_DWORD_TO_2WORD_X, também encapsula 2 Blocos do tipo DWORD_AS_BIT e que converte a variáveis de entrada do tipo DWORD em saídas em bits. Esses bits são alocados como entradas de 2 blocos do tipo BIT_AS_2WORD. A figura 31 apresenta o bloco.

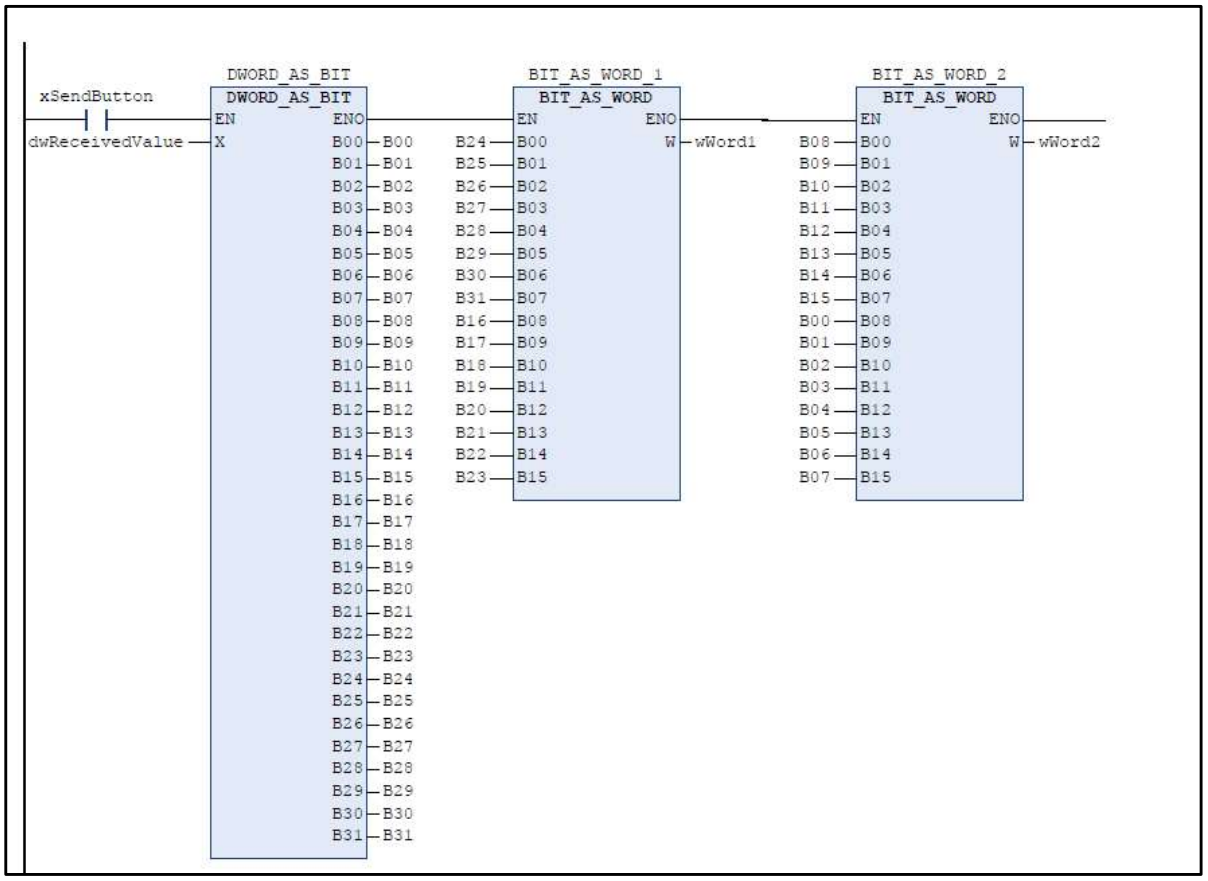


Figura 31 - Conversão DWORD em 2WORD para o FB_DWORD_TO_2WORD.

2.2.1 Configuração para Comunicação do CP Schneider M262

O Controlador Programável Schneider M262, para ser utilizado como elo de comunicação entre o MATLAB e Robô MOTOMAN GP8, necessitou de uma configuração prévia através da ferramenta Machine Expert. Inicialmente foi necessário definir o IP do CP.

A figura 32 apresenta essa configuração através de uma captura de tela.

Figura 32 - Configuração do IP do Controlador Programável.

Na figura 33 pode ser observada a configuração de IP do controlador YRC1000micro pertencente ao robô YASKAWA MOTOMAN GP8 realizada no programa do CP para estabelecimento de uma comunicação através da rede Ethernet.

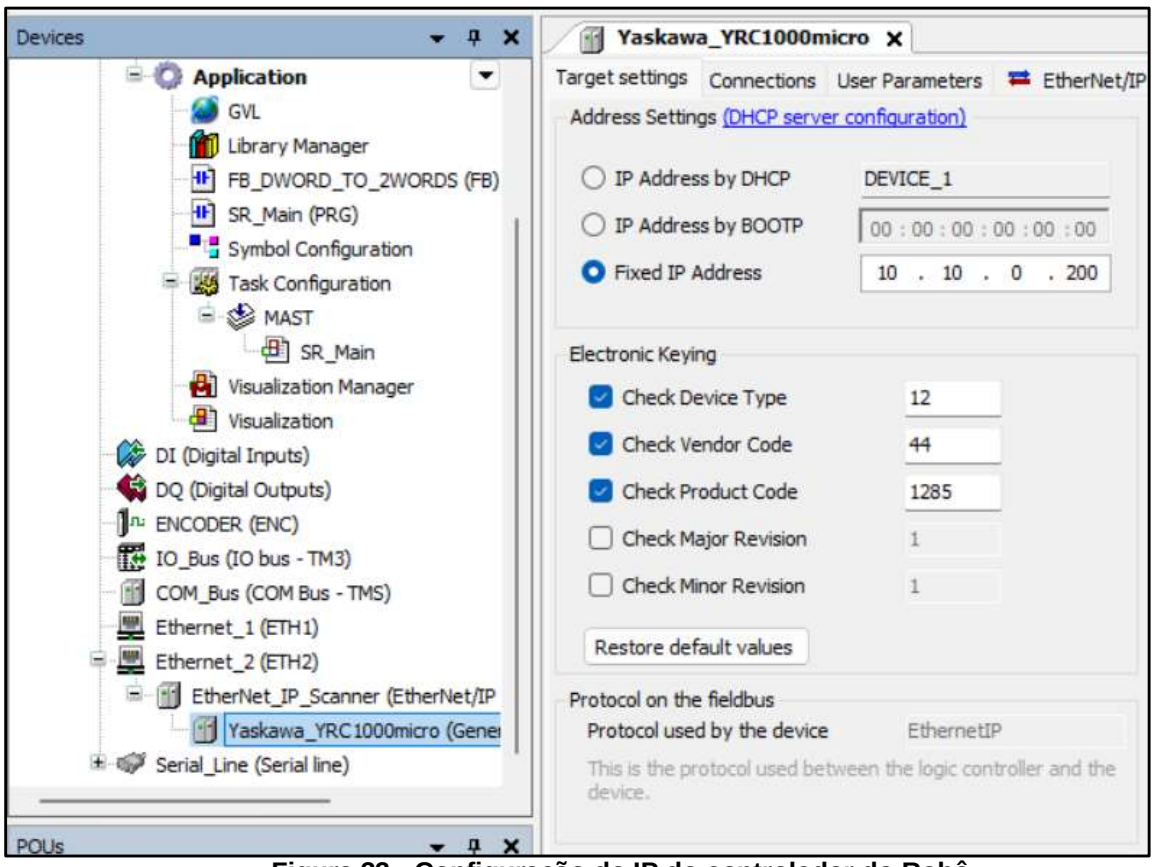


Figura 33 - Configuração do IP do controlador do Robô.

2.3 Metodologia da Programação do Sistema de Aprendizado por Demonstração

Nesse trabalho, devido a disponibilidade de memória do robô YASKAWA MOTOMAN GP8, foram utilizados os bytes disponíveis para movimentação dos eixos ”S”, “L”, “U”, “R”, e “B” do robô, ficando o eixo “T” para comando de início e termino de ação. Nesse caso, como são utilizados 4 bytes por eixo, foram utilizados 24 bytes de entrada e 24 de saída totalizando os 48 bytes disponíveis do robô.

Para o desenvolvimento da aplicação foi inicialmente criado e configurado o caminho entre as CNAPaps inseridas no software MATLAB e o robô YASKAWA MOTOMAN GP8 passando pela ponte de comunicação realizada pelo Controlador Programável Schneider M262. Para tal, uma configuração inicial do MATLAB foi realizada através da seleção das variáveis OPC UA de saída que foram escritas no Controlador Programável. A figura 34 a seguir identifica no fluxograma CNAPap essa ação.

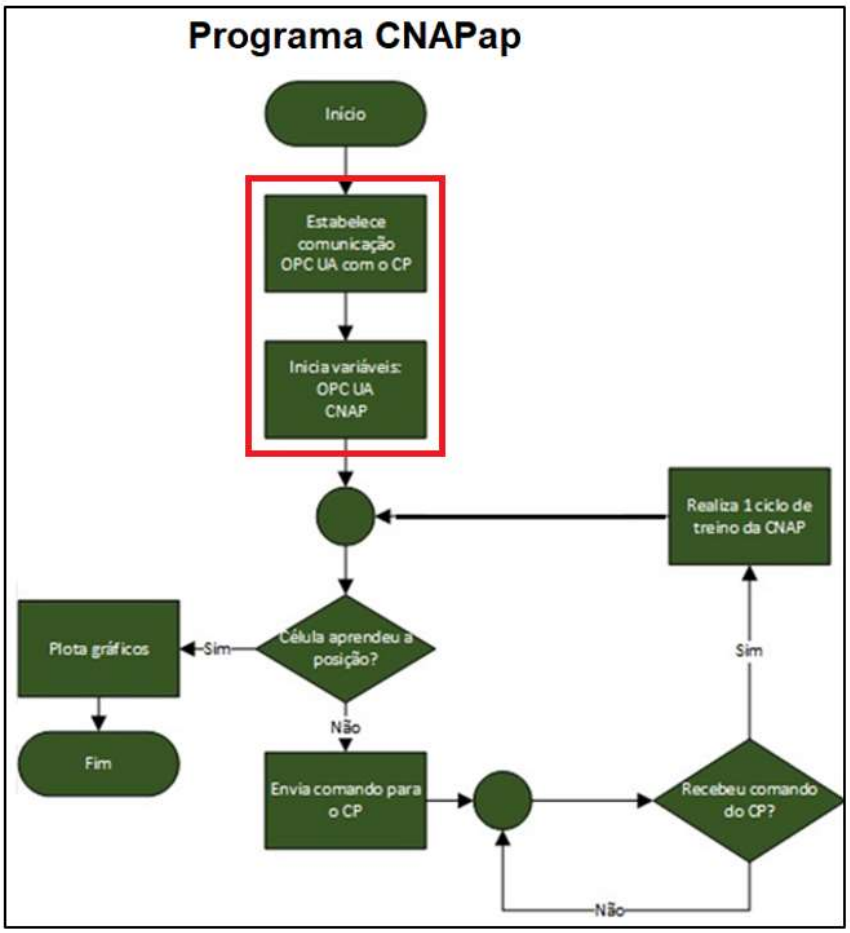


Figura 34 - CNAPap – Fluxograma com inicialização das variáveis OPC UA.

Após a seleção das variáveis OPC UA, foi necessário o envio, para cada eixo ou grau de liberdade do robô, um dado numérico inteiro e positivo, sendo fixado o valor 65535, nível “1” aplicado aos 2 últimos bytes do eixo T. Esse valor tem a finalidade de enviar um comando que resulta em uma mudança de posicionamento do robô, da posição inicial para a desejada. Para isso, o MATLAB estabelece a comunicação e envia o comando para o Controlador Programável, como pode ser observado através dos pontos de marcação nos fluxogramas da figura 35.

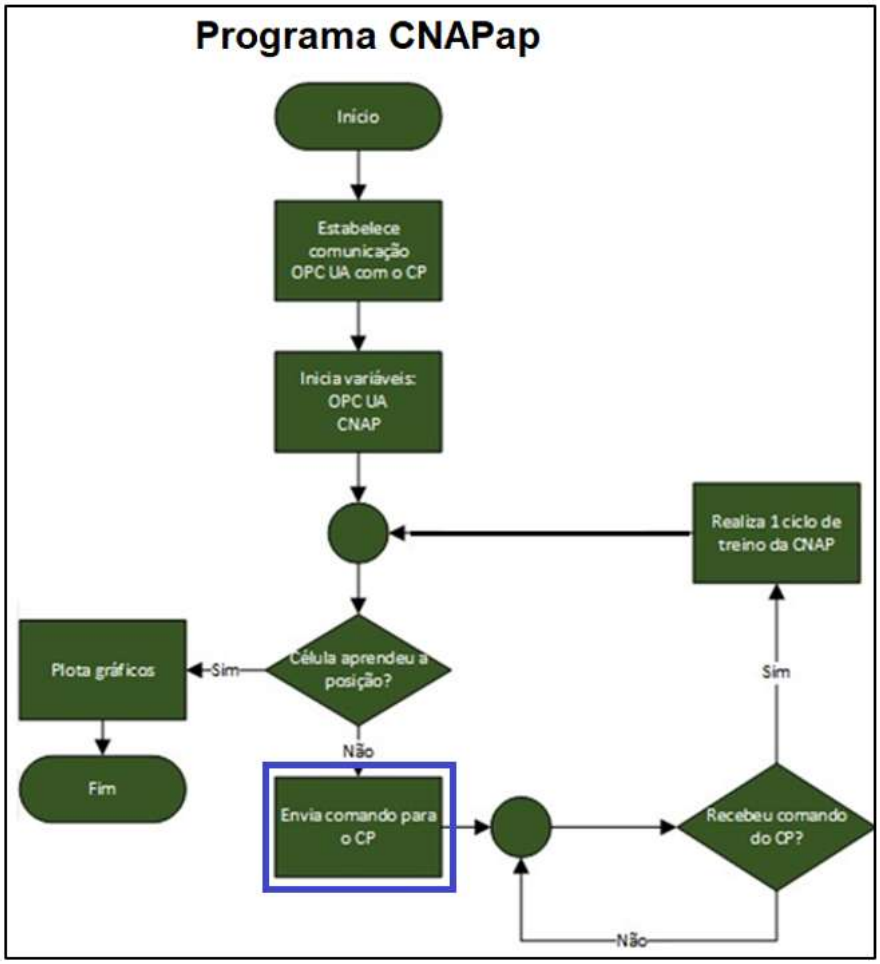


Figura 35 - MATLAB – Fluxograma com envio de comando para o CP.

Esse dado é enviado pelo programa CNAPap através de uma variável do tipo DOUBLE e recebido pelo CP, sendo armazenado em uma variável do tipo DOUBLE WORD, declarada como dwDwordRobot_X_send, onde X representa 1 dos 6 eixos do robô (S, L, U, R, B e T).

A seguir é apresentada a figura 36 com o mapa de memória do eixo “T” como exemplo dessa operação.

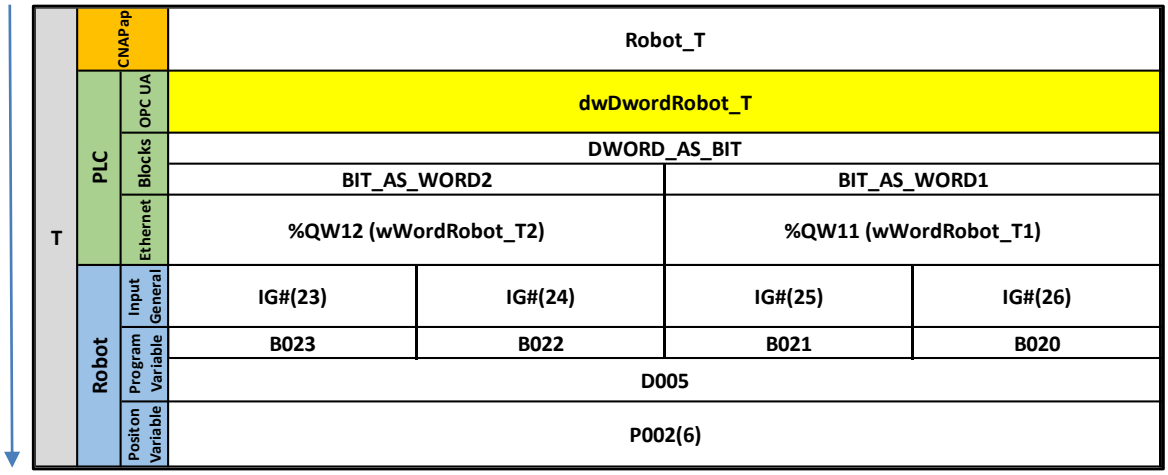


Figura 36 - Mapa de Memória com variável dwDwordRobot_T.

A declaração e o procedimento de associação das variáveis OPC UA de entrada do CP com as do tipo DWORD, dwDwordRobot_S_send, dwDwordRobot_L_send, dwDwordRobot_U_send, dwDwordRobot_R_send, dwDwordRobot_B_send e dwDwordRobot_T_send estão apresentadas na figura 37.

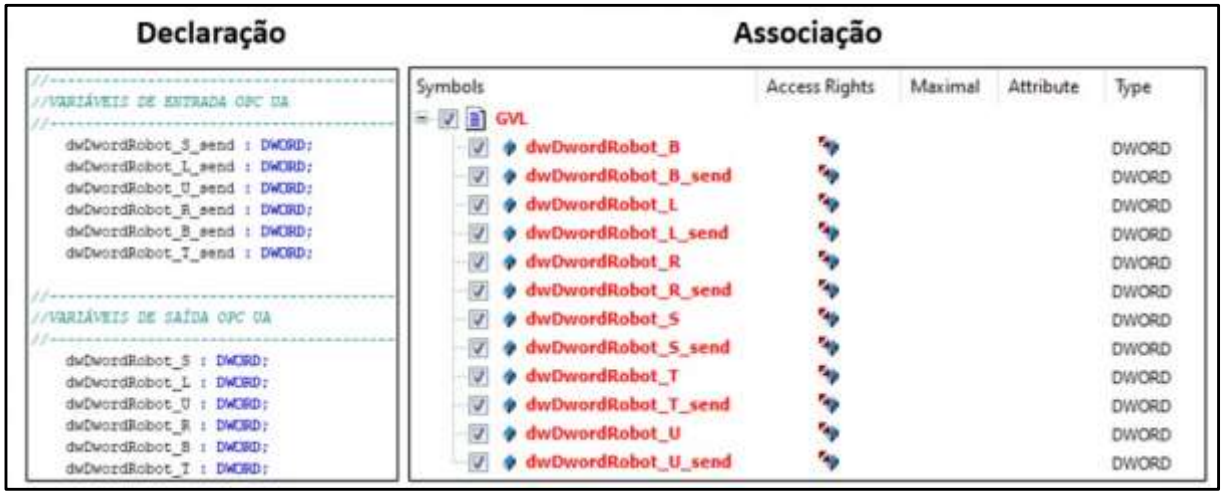


Figura 37 - Declaração e associação de variáveis de entrada OPC – CNAPap/CP.

O dado recebido pelo Controlador Programável é enviado ao bloco funcional denominado FB_DWORD_TO_2WORDS_X, desenvolvido para converter o dado recebido no formato DOUBLE WORD em 2 dados do tipo WORD. Para isso, esse dado é escrito em um bloco funcional denominado DWORD_AS_BIT, que converte, como o próprio nome já indica, uma DWORD em uma saída de 32 bits. Em seguida, são utilizados 2 blocos funcionais do tipo BIT_AS_WORD, denominados

BIT_AS_WORD_1 e BIT_AS_WORD_2, que armazenam os 8 bits mais significativos e os 8 bits menos significativos, respectivamente, por meio das variáveis wWordRobot_X1, wWordRobot_X2. Essa conversão foi necessária tendo em vista que não existe uma passagem direta de DWORD para WORD disponível no CP. Uma imagem dessa combinação de blocos está apresentada na figura 38.

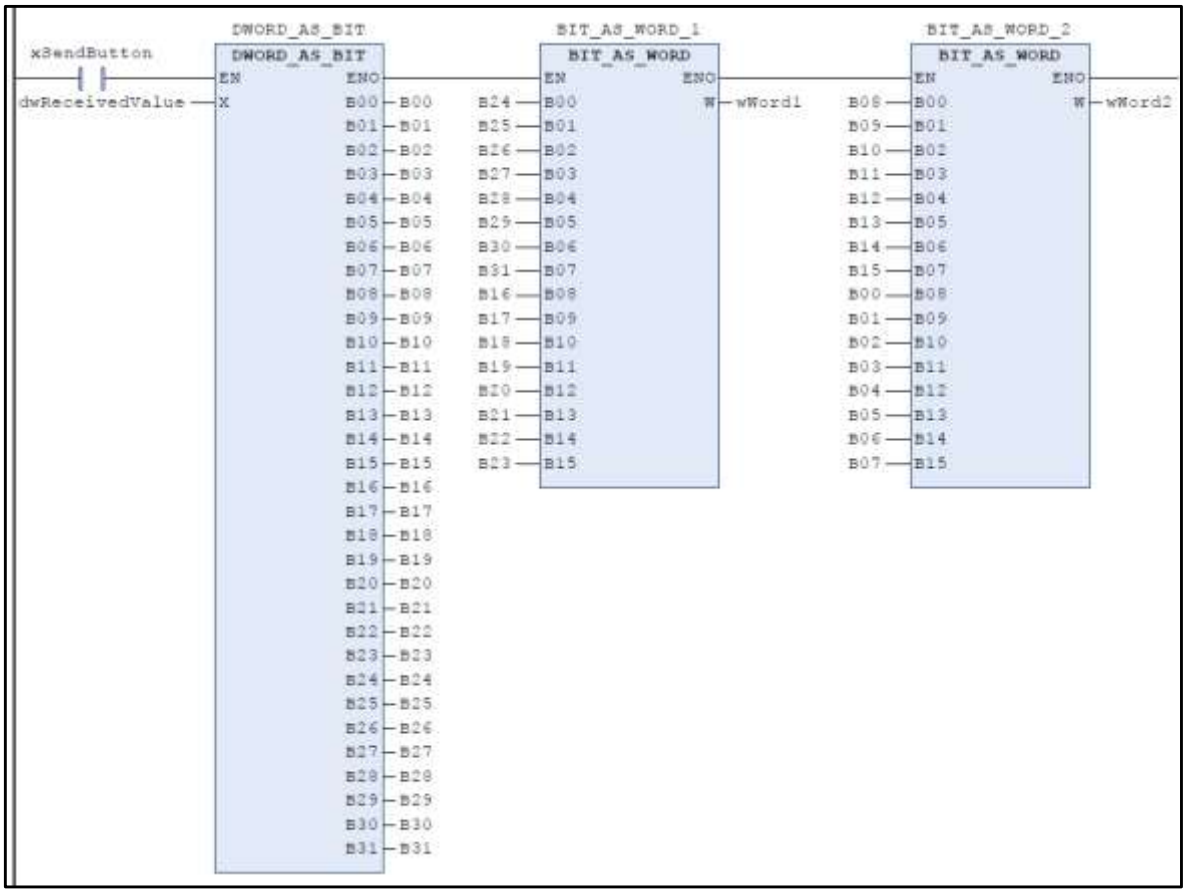


Figura 38 - BF DWORD_AS_BIT e BIT_AS_WORD_1/ BIT_AS_WORD_2.

A movimentação no mapa de memória é apresentada na figura 39, no qual em amarelo está a DWORD_AS_BIT e, na saída em verde, estão as saídas BIT_AS_WORD1 e BIT_AS_WORD2.

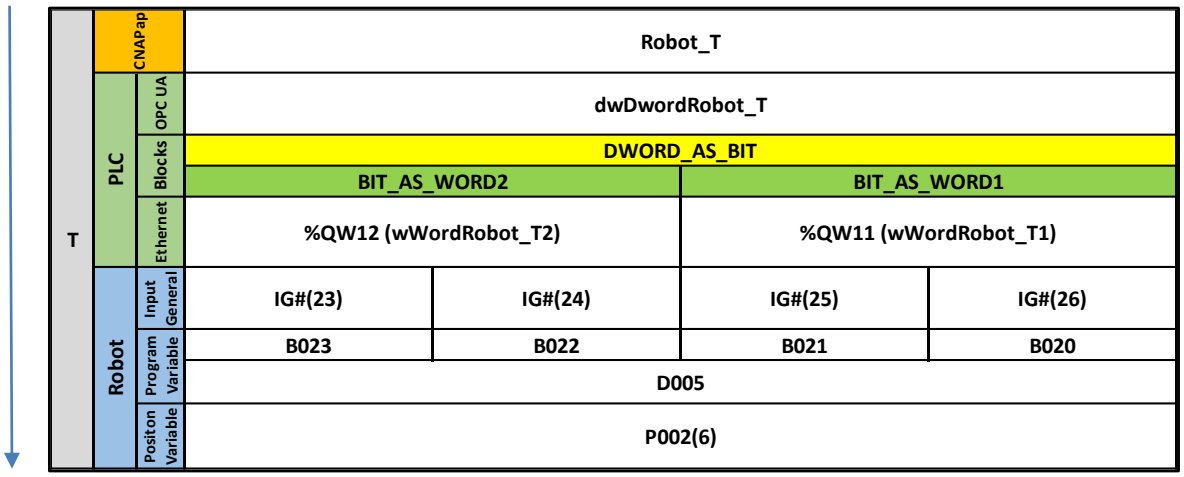


Figura 39 - Mapa de Memória com variáveis D_WORD_AS_BIT, BIT_AS_WORD1 e BIT_AS_WORD2.

Como premissa da IEC 61131-3 os blocos funcionais DWORD_AS_BIT, BIT_AS_WORD_1 e BIT_AS_WORD_2 estão encapsulados no bloco FB_DWORD_TO_2WORDS_X. Na figura 40 é apresentado em linguagem *Ladder,* como exemplo, o bloco “T” utilizado no programa principal do CP.

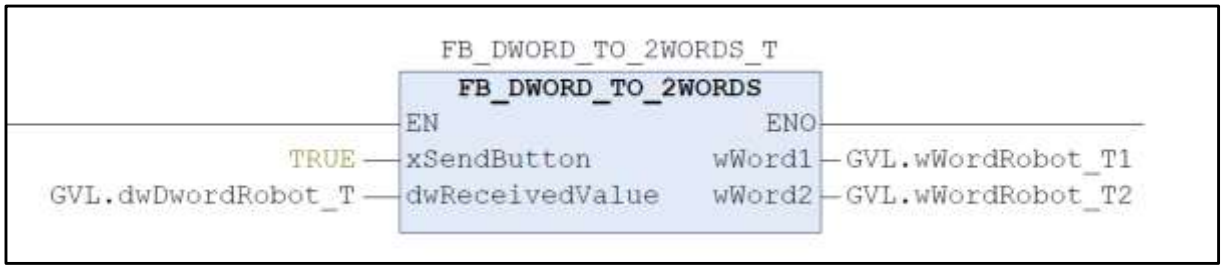


Figura 40 - FB_DWORD_TO_2WORDS_T.

As variáveis wWordRobot_X1, wWordRobot_X2 são associadas às variáveis %QW da rede Ethernet e então são enviadas para o robô. Na figura 41 é apresentado o exemplo dessa associação com a utilização do eixo “T”.

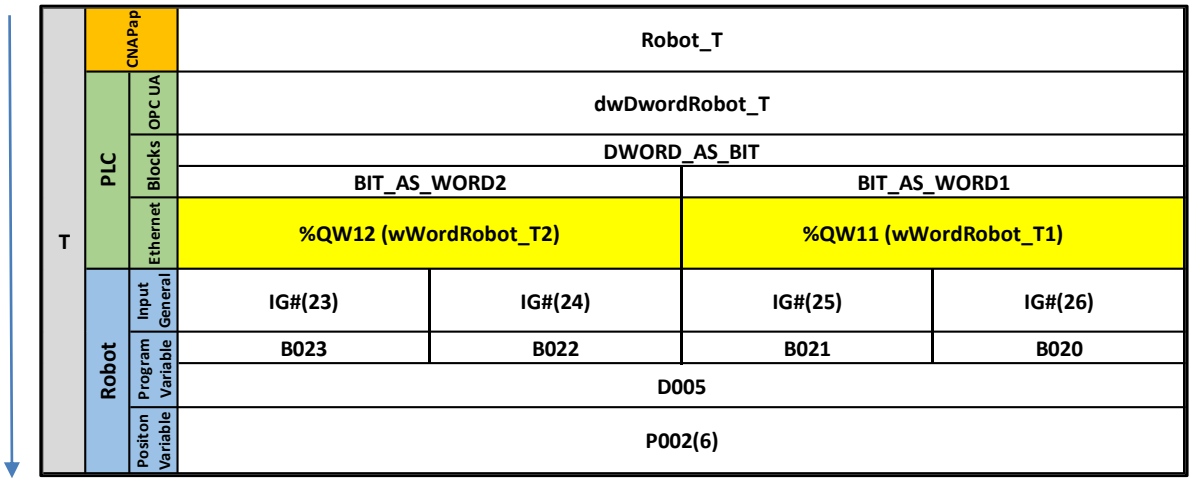


Figura 41 - Mapa de Memória - Associação entre variáveis wWordRobot_T e %QW.

Na figura 42 está apresentada uma imagem com as áreas do Machine Expert (software do CP), onde são declaradas e associadas as variáveis Ethernet de saída que serão enviadas ao robô MOTOMAN GP8 pelo Controlador Programável.

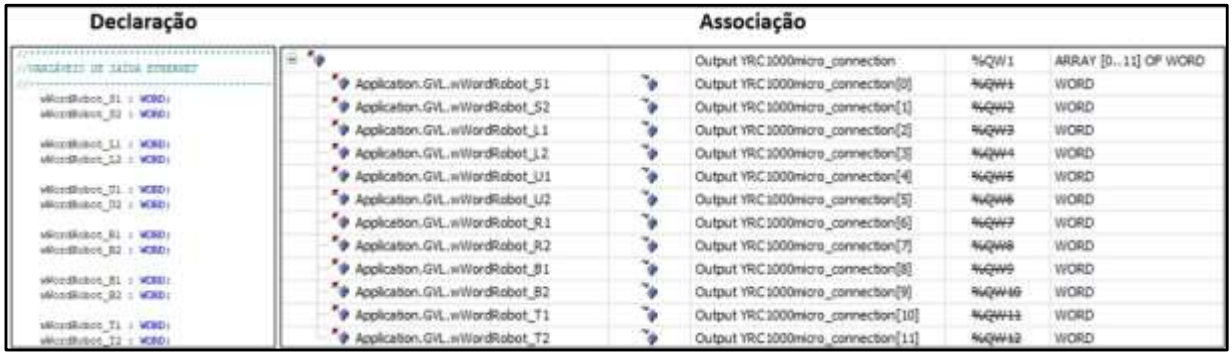


Figura 42 - Declaração e associação de variáveis Ethernet CP/GP8.

Os dados no formato WORD são recebidos pelo robô e em seguida armazenados em 2 variáveis do tipo BYTE, alocadas nas INPUT GENERAL (IG#), a partir do endereço 3. Uma figura com o mapa de memória indicando o posicionamento das variáveis no eixo “T” é mostrada na figura 43 onde os bytes 25 e 26 estão evidenciados em cor verde. Estes são os únicos não zerados que chegam ao controlador do robô utilizando o valor 65535 que serve de código de comando para a alteração do robô da posição inicial para a posição desejada.

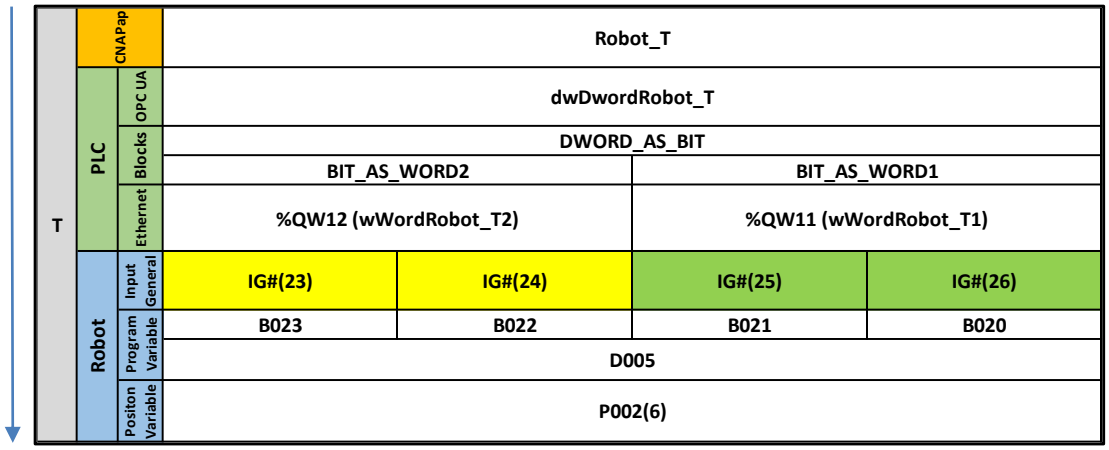


Figura 43 - Mapa de Memória – Variáveis Input General – IG#.

Essa alocação ocorre de forma automática, ou seja, ao chegarem 16 bits, o controlador do robô os aloca na primeira e na segunda variável disponível a partir do endereço 3. Como consequência, são destinadas 24 variáveis das INPUT GENERAL para receber os dados provenientes do CLP (4 bytes para cada um dos 6 eixos do robô), partindo do endereço 3 e ocupando até o endereço 26 de acordo com o eixo, conforme sequência a seguir:

a- IG#(3), IG#(4), IG#(5), IG#(6) para o eixo “S”.

b- IG#(7), IG#(8), IG#(9), IG#(10) para o eixo “L”.

c- IG#(11), IG#(12), IG#(13), IG#(14) para o eixo “U”.

d- IG#(15), IG#(16), IG#(17), IG#(18) para o eixo “R”.

e- IG#(19), IG#(20), IG#(21), IG#(22) para o eixo “B”.

f- IG#(23), IG#(24), IG#(25), IG#(26) para o eixo “T”.

Dessa forma chegam no robô os dados apresentados para os eixos “S”, “L”, “U”, “R”, “B”, e “T” inseridos na tabela 4.

Tabela 4 – Dados enviados pelo CP ao robô MOTOMAN – GP8.

EIXO S																															
IG#(3)								**IG#(4)**								**IG#(5)**								**IG#(6)**							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EIXO L																															
IG#(7)								**IG#(8)**								**IG#(9)**								**IG#(10)**							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EIXO U																															

Continuação da Tabela 4

IG#(11)								IG#(12)								IG#(13)								IG#(14)							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EIXO R																															
IG#(15)								IG#(16)								IG#(17)								IG#(18)							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EIXO B																															
IG#(19)								IG#(20)								IG#(21)								IG#(22)							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EIXO T (COMANDO PARA O ROBÔ)																															
IG#(23)								IG#(24)								IG#(25)								IG#(26)							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Na figura 44 estão apresentados os eixos do robô na qual a rotação da garra (J6) não foi utilizado na movimentação devido a necessidade da utilização desses bytes para comando de alteração de posição do robô. Com o comando recebido por **IG#(25) e IG#(26)** o MOTOMAN GP8 se desloca da sua posição inicial (home) até a posição final desejada.

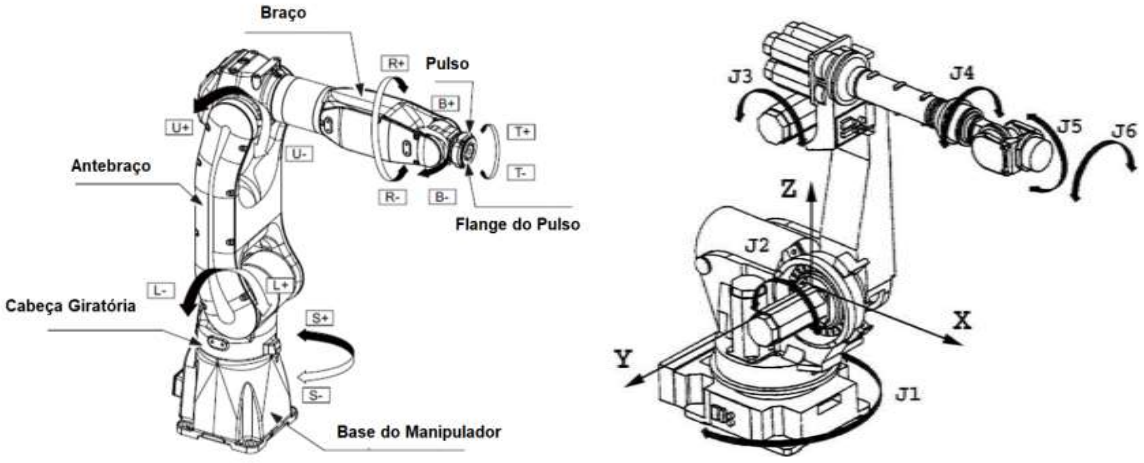


Figura 44 - Eixos do robô MOTOMAN GP8 utilizáveis.

Para melhor entendimento da dinâmica de envio de mudança de posição do robô é apresentado na figura 45 um fluxograma indicando a sequência do *status* atual da ação após o comando enviado pelo Controlador Programável.

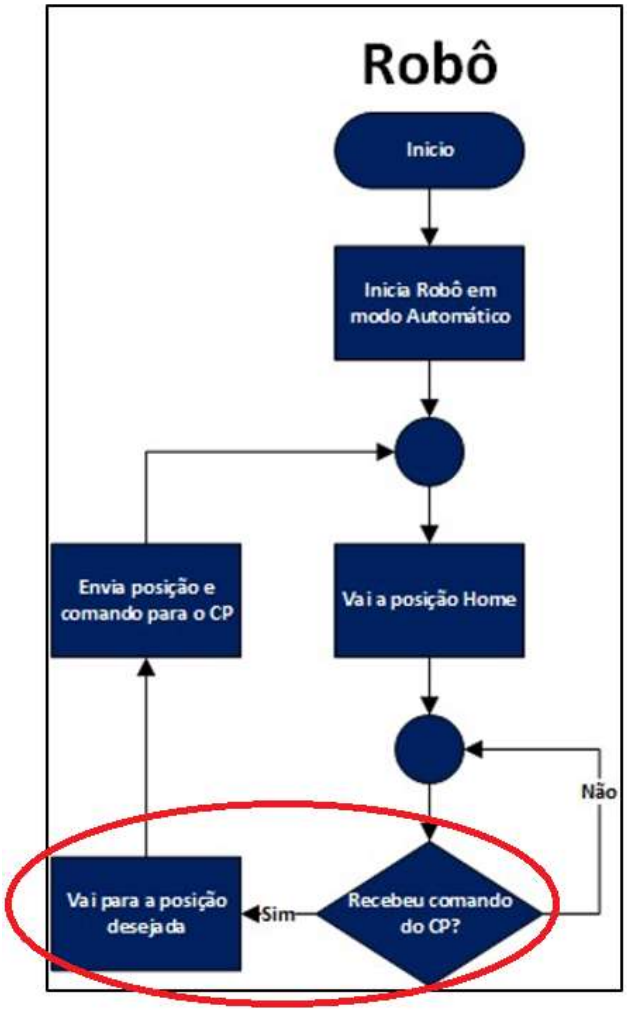


Figura 45 - Fluxograma de Ação do MOTOMAN GP8 – Posição desejada.

Após finalizado o deslocamento do robô até a sua posição final, se faz necessário o envio dos dados armazenados pelos seus eixos ”S”, “L”, “U”, “R”, e “B”, bem como mais uma vez o eixo “T” utilizado para comando de início e termino de ação do MOTOMAN GP8, até sua recepção pelo *software* MATLAB, sendo portanto, necessária a criação de um caminho de retorno passando mais uma vez pela ponte de comunicação realizada pelo CP Schneider M262. Essa sequência é indicada no fluxograma da figura 46.

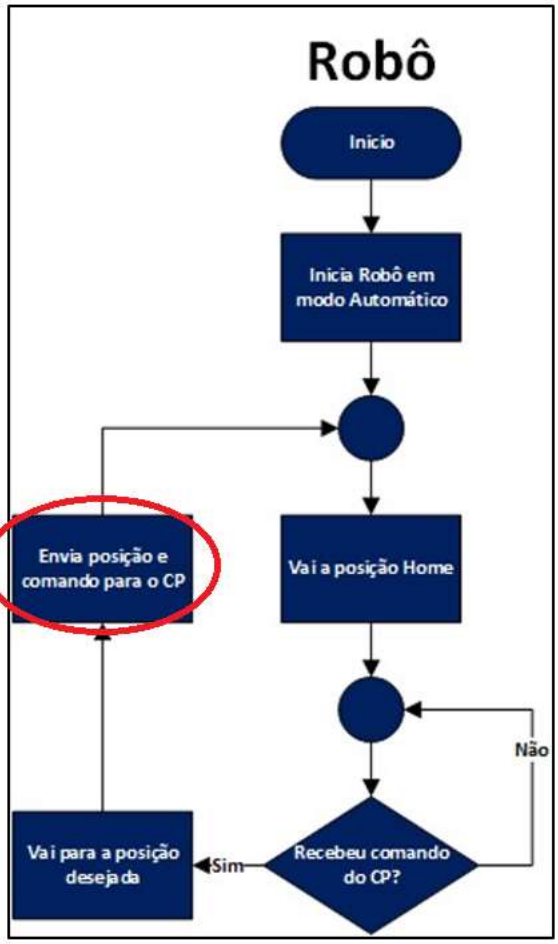


Figura 46 - Fluxograma de Ação do MOTOMAN GP8 – Posição desejada.

Como cada eixo do robô possui um *range* angular e um valor proporcional de deslocamento em pulsos, no seu primeiro elemento fica alocado o valor de posição do eixo S e, sucessivamente, até o elemento 6, que armazena o valor do eixo T.

a- P000(1) para o eixo “S”
b- P000(2) para o eixo “L”
c- P000(3) para o eixo “U”
d- P000(4) para o eixo “R”
e- P000(5) para o eixo “B”
f- P000(6) para o eixo “T”

Os dados dos eixos do robô armazenados na sua posição desejada final, são apresentados com os seus respectivos valores em Pulsos na figura 47 conforme capturada do seu *Teach Pendant* virtual.

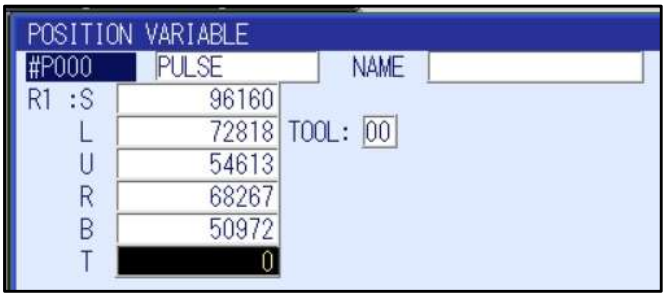


Figura 47 – Posições dos Eixos do Robô MOTOMAN GP8 mostradas no *Teach Pendant* virtual.

Como exemplo, é apresentado na figura 48 o mapa de memória do retorno do robô até sua chegada ao programa CNAPap, indicando a posição atual do eixo “S” em sua P000(1). As posições de cada eixo seguem a seguinte sequência:

a- P000(1) para o eixo “S”

b- P000(2) para o eixo “L”

c- P000(3) para o eixo “U”

d- P000(4) para o eixo “R”

e- P000(5) para o eixo “B”

f- P000(6) para o eixo “T”

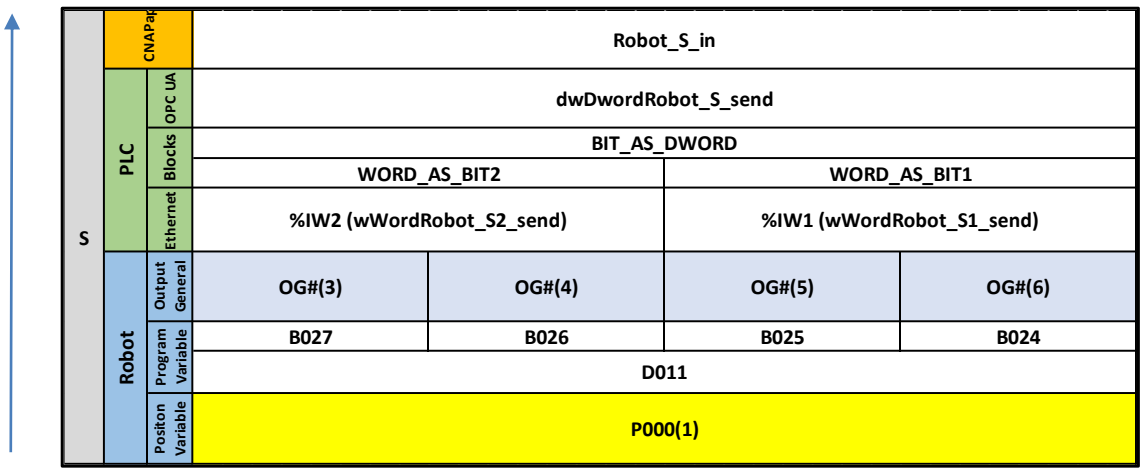


Figura 48 - Mapa de Memória – Variável de Posição eixo “S” – P000(1).

Nesse momento é enviado por P000(1) o valor correspondente a posição do eixo “S”, ou seja, o valor em pulsos 96160. Na sequência, é armazenado em D011, o valor da posição do eixo “S” somado ao seu valor máximo negativo (153600), que assim garante sempre um número positivo.

Os valores de cada eixo alocados nos seus respectivos elementos da variável de posição P000(1) a P000(6) são, portanto, copiados para variáveis do tipo DOUBLE, com a seguinte sequência:

a- D011 para o eixo “S”
b- D012 para o eixo “L”
c- D013 para o eixo “U”
d- D014 para o eixo “R”
e- D015 para o eixo “B”
f- D016 para o eixo “T”

Para acompanhamento da fase atual, na figura 49 é apresentado o mapa de memória do eixo “S” com a variável D011 em evidência.

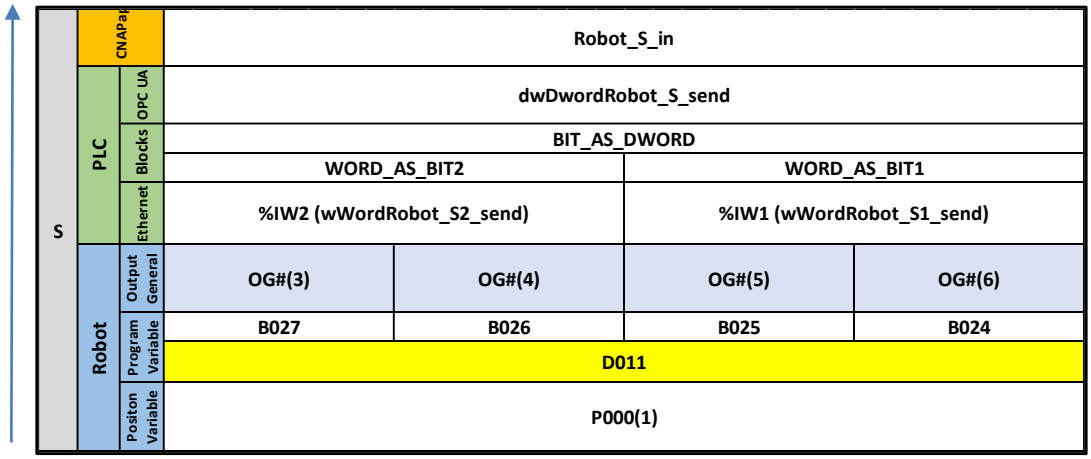


Figura 49 - Mapa de Memória – Variável de Programa eixo “S”.

Nesse ponto do programa do robô, os dados alocados nas variáveis DOUBLE são convertidos para 4 variáveis do tipo BYTE. A seguir, na tabela 5 é apresentado o resultado da conversão do número 249760 e sua respectiva alocação nas variáveis do eixo “S”.

O método com a programação completa pode ser encontrado no Apêndice D.

Tabela 5 – Conversão para Variáveis de Programa

B027								B026								B025								B024							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0

O valor final correspondente ao número 249760 é a somatória da posição do eixo “S” do robô (96160) com o seu valor mínimo (153600). Nessa fase, cada eixo ocupa as variáveis de acordo com a seguinte sequência:

a- B027, B026, B025, B024 para o eixo “S”.
b- B031, B030, B029, B028 para o eixo “L”.
c- B035, B034, B033, B032 para o eixo “U”.
d- B039, B038, B037, B036 para o eixo “R”.
e- B043, B042, B041, B040 para o eixo “B”.
f- B047, B046, B045, B044 para o eixo “T”.

Para o acompanhamento do *status* atual do programa, os dados do Mapa de memória do eixo “S” são apresentados, como exemplo, na figura 50.

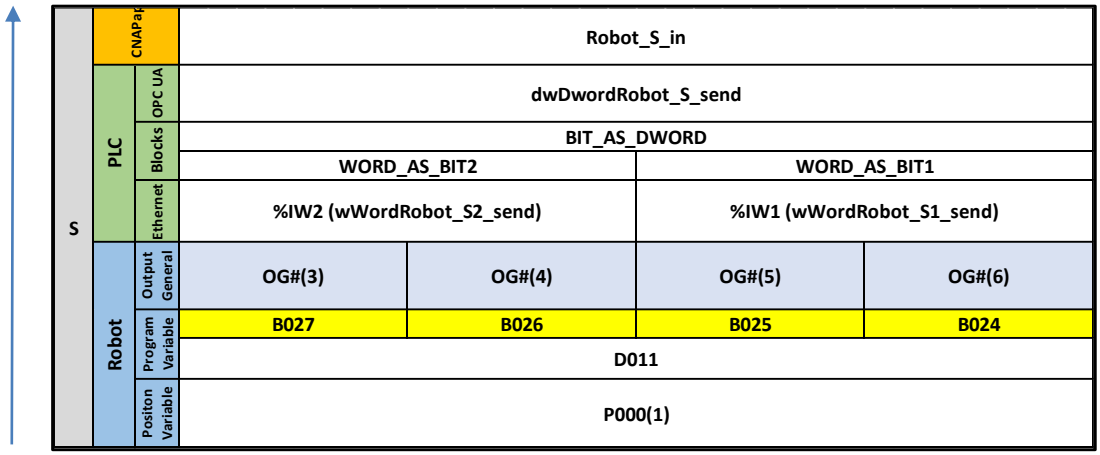


Figura 50 - Mapa de Memória – Variável de Programa eixo “S” – D011.

Os dados no formato BYTE são copiados pelo robô e alocadas nas OUTPUT GENERAL (OG#), a partir do endereço 3. Essa alocação é indicada na programação que se encontra completa no Apêndice D. Como consequência, são destinadas 24 variáveis das OUTPUT GENERAL para enviar os dados provenientes do robô para o CP através do protocolo de conexão Ethernet, sendo 4 bytes para cada um dos 6 eixos do robô, partindo do endereço 3 até o endereço 26, de acordo com o eixo.

a- OG#(3), OG#(4), OG#(5), OG#(6) para o eixo “S”.
b- OG#(7), OG#(8), OG#(9), OG#(10) para o eixo “L”.
c- OG#(11), OG#(12), OG#(13), OG#(14) para o eixo “U”.
d- OG#(15), OG#(16), OG#(17), OG#(18) para o eixo “R”.

e- OG#(19), OG#(20), OG#(21), OG#(22) para o eixo “B”.

f- OG#(23), OG#(24), OG#(25), OG#(26) para o eixo “T”.

A seguir, pode-se acompanhar na figura 51 o *status* atual do programa do eixo “S” no mapa de memória ocupando as variáveis OG#(3), OG#(4), OG#(5) e OG#(6).

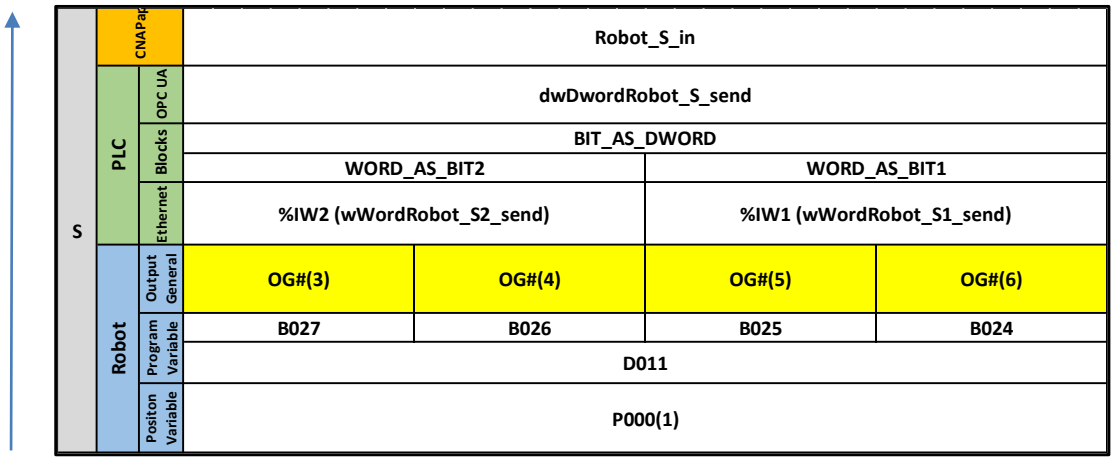


Figura 51 - Mapa de Memória – Output General eixo “S”.

Nessa fase da comunicação os dados provenientes das OG# são escritos no CP Schneider M262. A partir desse ponto o *status* atual e as ações que serão necessárias para disponibilizar os dados das posições dos eixos ”S”, “L”, “U”, “R”, “B” e “T” serão descritos.

Na figura 52 pode ser observado no fluxograma de operações do controlador, a área que será detalhada no transcorrer das ações até o envio dos dados ao MATLAB.

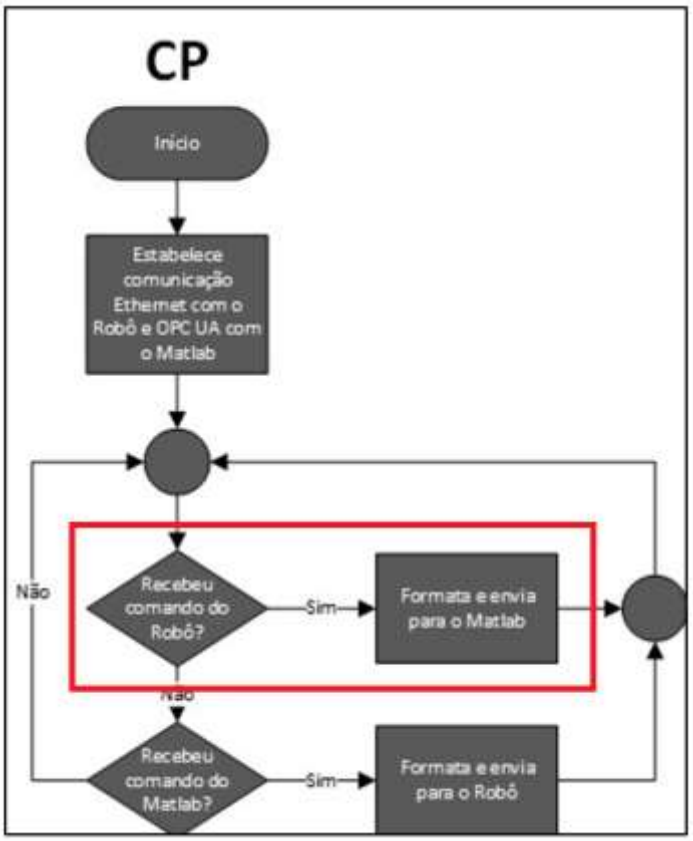


Figura 52 - Fluxograma de Ação do CP Schneider M262 – Robô/MATLAB.

Em uma sequência de fases, no Machine Expert são declaradas e associadas as variáveis Ethernet de entrada, que serão recebidas pelo CP e, em procedimentos sequenciais enviadas ao MATLAB. Uma sequência da declaração e associação das variáveis Ethernet, está apresentada na figura 53.

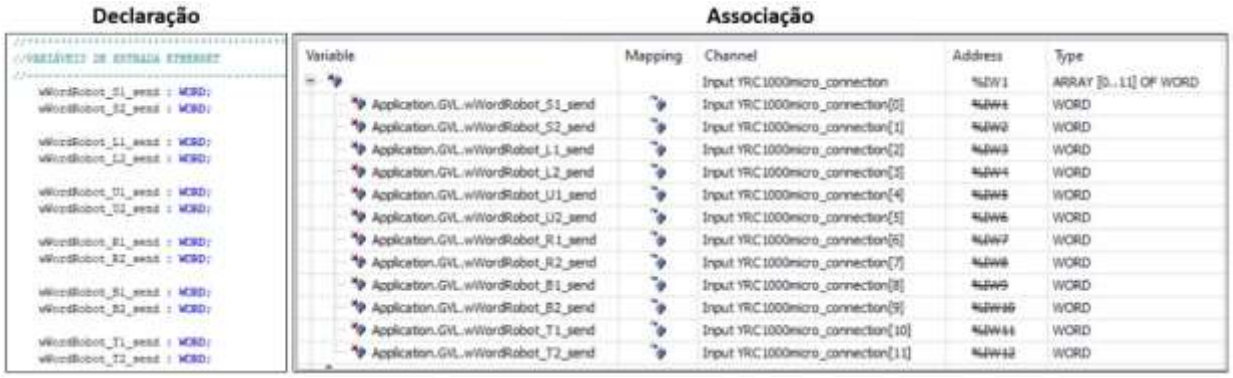


Figura 53 - Declaração e associação de variáveis Ethernet GP8/CP.

Cada variável do tipo OUTPUT GENERAL é então escrita na WORD wWordRobot_X_send declaradas no CP, onde “X” representa o eixo do robô. Uma imagem do mapa de memória do eixo “S” com essa ação pode ser observada na figura 54.

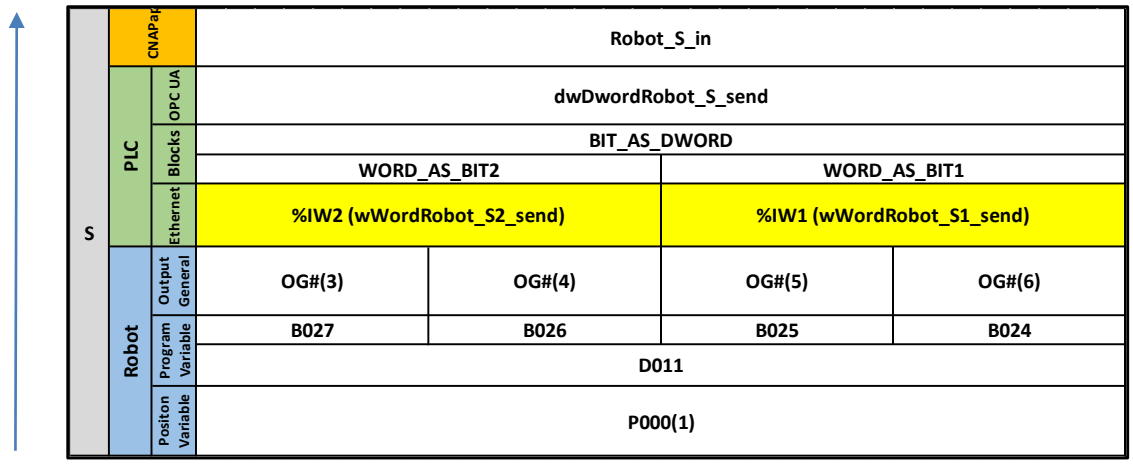


Figura 54 - Mapa de Memória – %IWX (wWordRobot_X_send).

O dado recebido pelo Controlador Programável é enviado ao bloco funcional denominado FB_2WORD_TO_DWORDS_X, desenvolvido para converter 2 dados recebidos no formato WORD em um dado do tipo DWORD. Para isso, esse dado é escrito em dois blocos funcionais do tipo WORD_AS_BIT, denominados WORD_AS_BIT2 e WORD_AS_BIT1 que convertem uma WORD em uma saída de 16 bits. Como conexão é utilizado um BF do tipo BIT_AS_DWORD, que armazena respectivamente os 16 bits mais e menos significativos.

As saídas dos Blocos funcionais são escritas em 6 variáveis do tipo DWORD, denominadas:

a- dwDwordRobot_S_send
b- dwDwordRobot_L_send
c- dwDwordRobot_U_send
d- dwDwordRobot_R_send
e- dwDwordRobot_B_send
f- dwDwordRobot_T_send

Um exemplo dessa interação, utilizando o eixo “S” como exemplo, está apresentada na figura 55. O valor de saída do bloco carrega a posição do eixo “S” do

robô somado ao valor mínimo que garante um dado positivo, ou seja, o valor 249760, que foi enviado pelo robô.

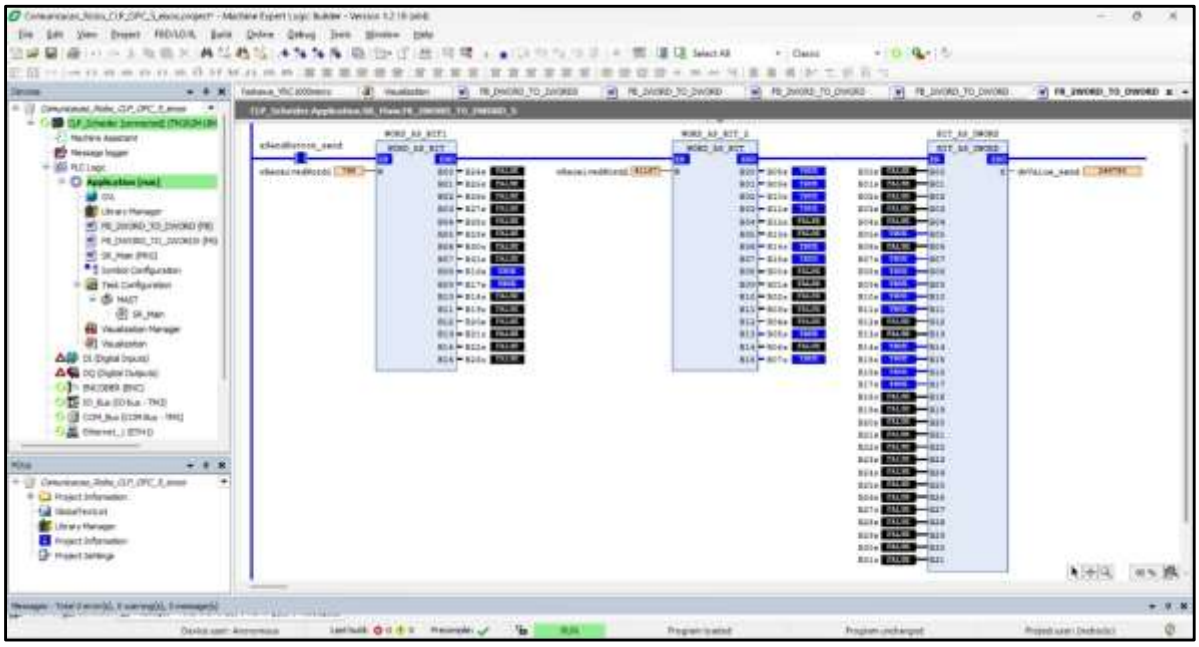


Figura 55 - BF do eixo “S” - WORD_AS_BIT1 e WORD_AS_BIT 2 para BIT_AS_DWORD.

Os blocos funcionais do tipo WORD, WORD_AS_BIT2 WORD_AS_BIT1 e BIT_AS_WORD_2, juntamente com o BF BIT_AS_DWORD estão encapsulados no FB_2WORD_TO_DWORDS_X. Na figura 56 é apresentado como exemplo, o bloco “S” utilizado no programa do CP.

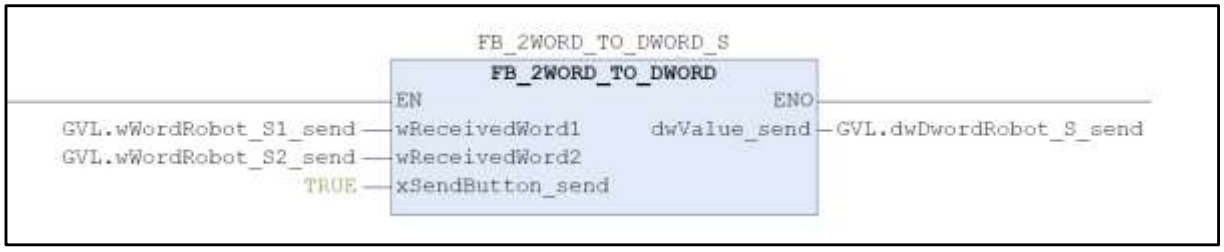


Figura 56 - Bloco Funcional FB_2WORD_TO_DWORDS_X.

O posicionamento atual no mapa de memória do eixo “S” pode ser observado a seguir, na figura 57.

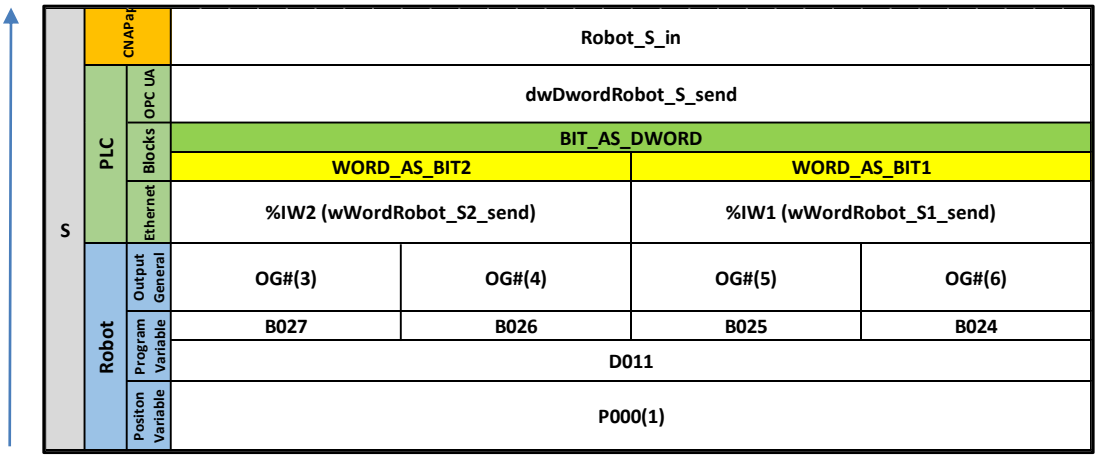


Figura 57 - Mapa de Memória com variáveis WORD_AS_BIT1, WORD_AS_BIT2, e BIT_AS_DWORD.

Os valores com as respectivas conversões enviados pelos blocos funcionais FB_2WORD_TO_DWORDS_X dos eixos “L”, “U”, “R”, “B” e “T” são apresentados respectivamente nas figuras 58, 59, 60, 61 e 62. Os valores com as suas respectivas posições, valores mínimos e valor somado que serão enviados ao MATLAB, estão expostos a seguir:

a- Eixo “L” = valor **72818** (pulso mínimo) + **72818 (set point) = 145636 (DWORD)**

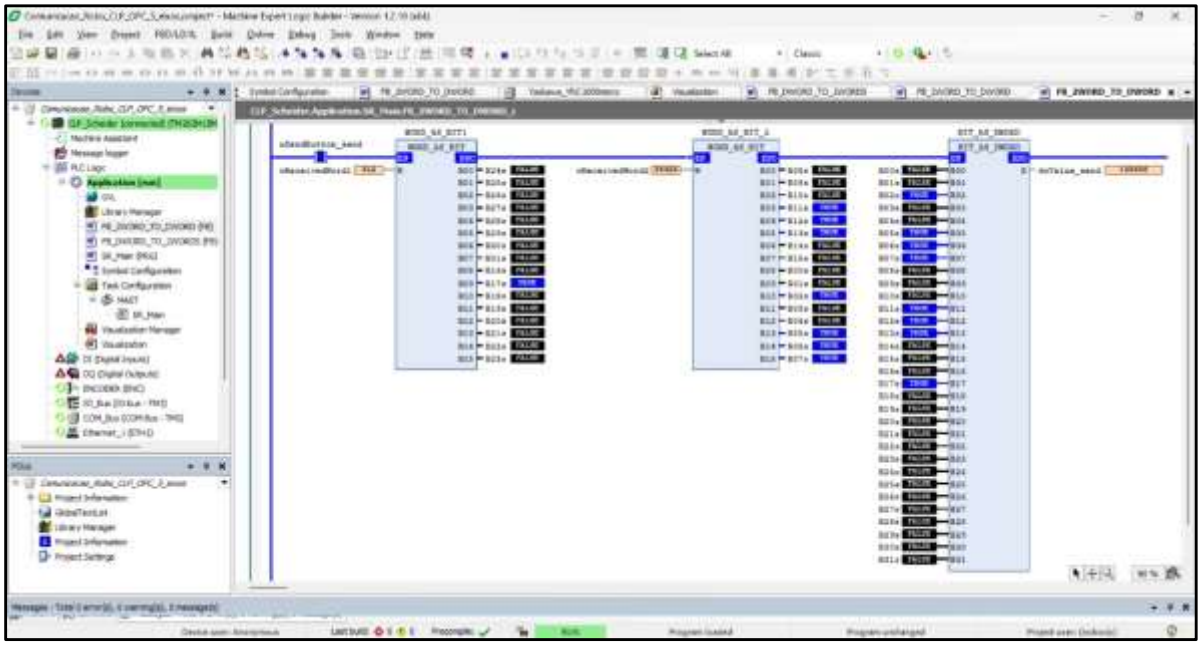


Figura 58 - BF do eixo “L” - WORD_AS_BIT1 e WORD_AS_BIT 2 para BIT_AS_DWORD.

b- Eixo “U” = valor **54614** (pulso mínimo) + **54613 (set point) = 109227 (DWORD)**

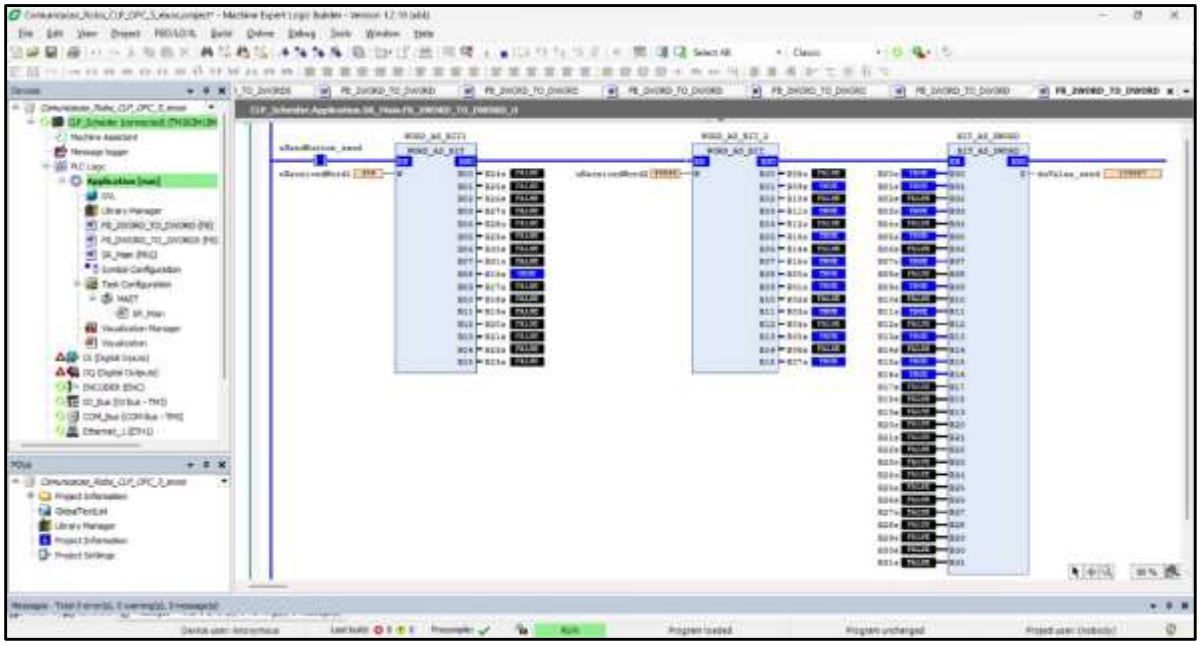


Figura 59 - BF do eixo “U” - WORD_AS_BIT1 e WORD_AS_BIT 2 para BIT_AS_DWORD.

c- Eixo “R” = valor **153600**(pulso mínimo) + **68267 (set point) = 221867 (DWORD)**

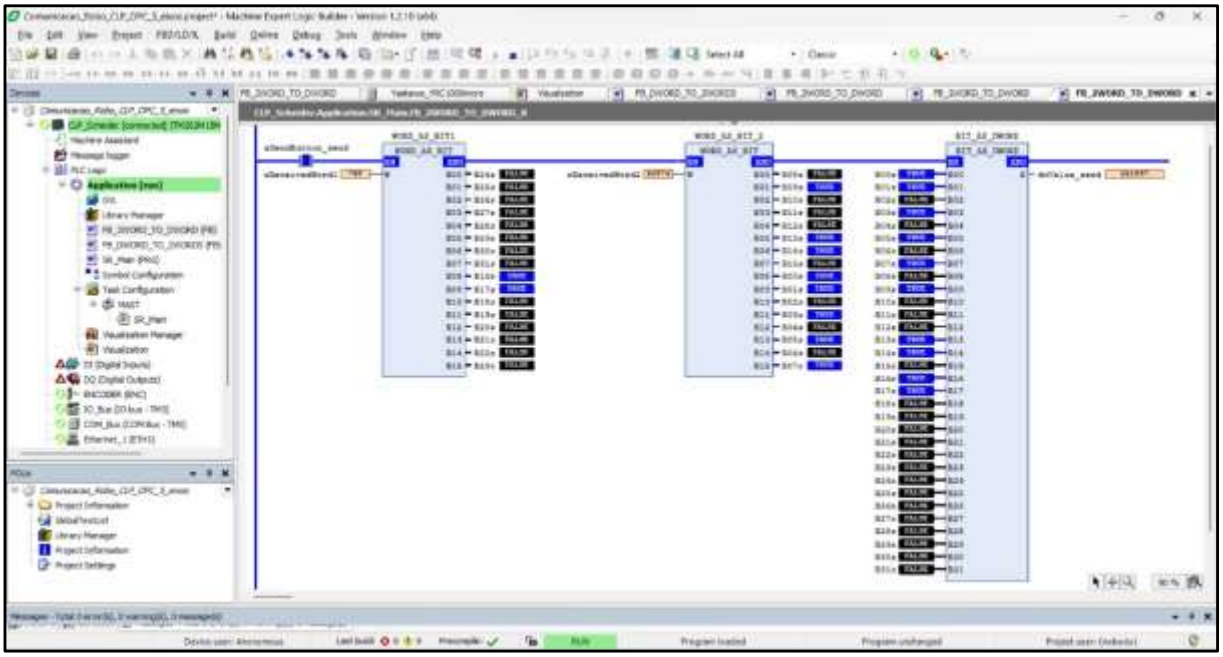


Figura 60 - BF do eixo “R” - WORD_AS_BIT1 e WORD_AS_BIT 2 para BIT_AS_DWORD.

d- Eixo “B” = valor **87381** (pulso mínimo) + **50972 (set point) = 138353 (DWORD)**

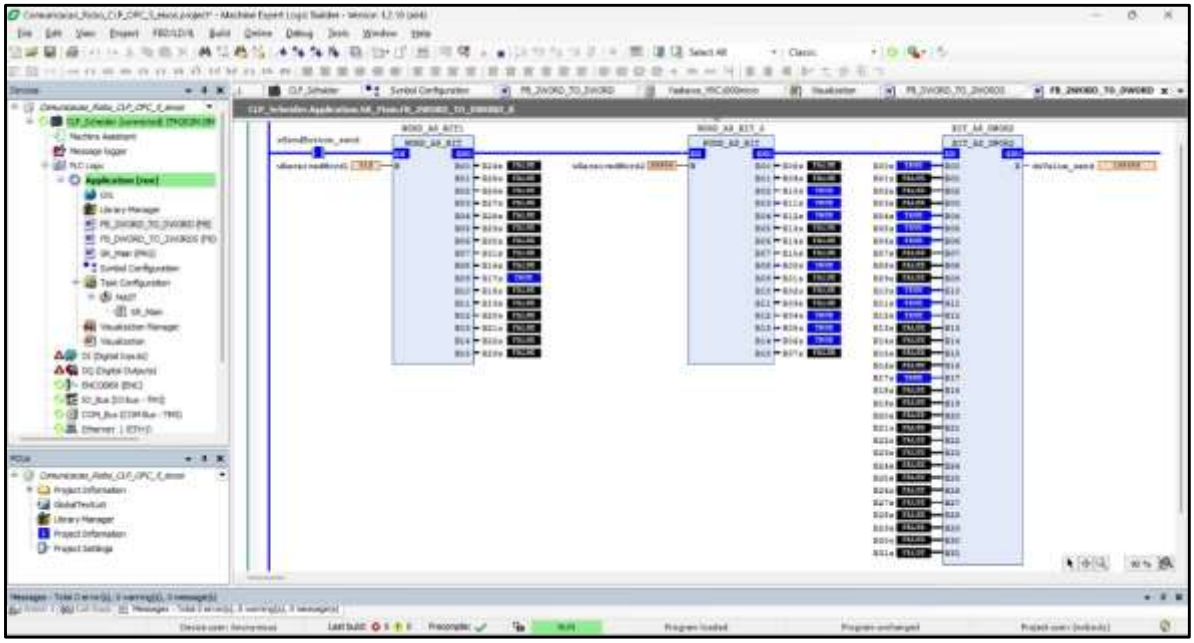


Figura 61 - BF do eixo “B” - WORD_AS_BIT1 e WORD_AS_BIT 2 para BIT_AS_DWORD.

e- Eixo “T” = valor **0** (pulso mínimo) + **0 (set point) = 0 (DWORD)**

Eixo utilizado como sinalização de envio de dados.

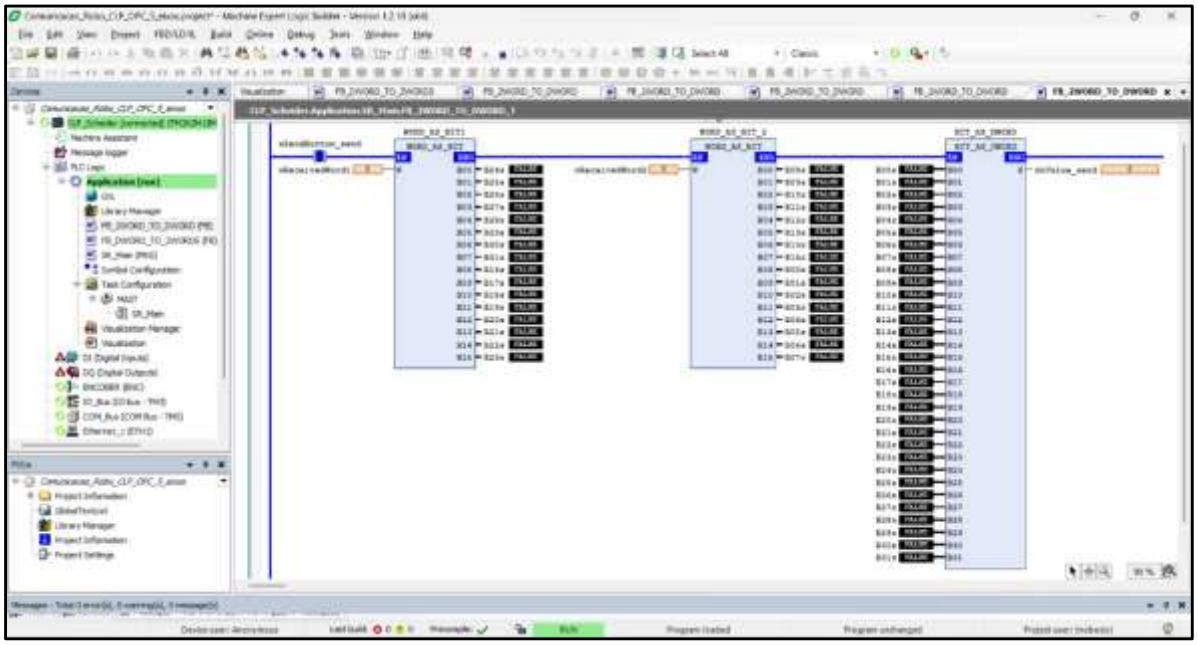


Figura 62 - BF do eixo “T” - WORD_AS_BIT1 e WORD_AS_BIT 2 para BIT_AS_DWORD.

Os dados enviados pelos blocos funcionais FB_2WORD_TO_DWORDS dos eixos “S”, “L”, “U”, “R”, “B” e “T” são então associados as variáveis OPC UA de saída. Isto pode ser observado na figura 63 através de uma captura de tela do Machine Expert utilizado pelo Schneider M262.

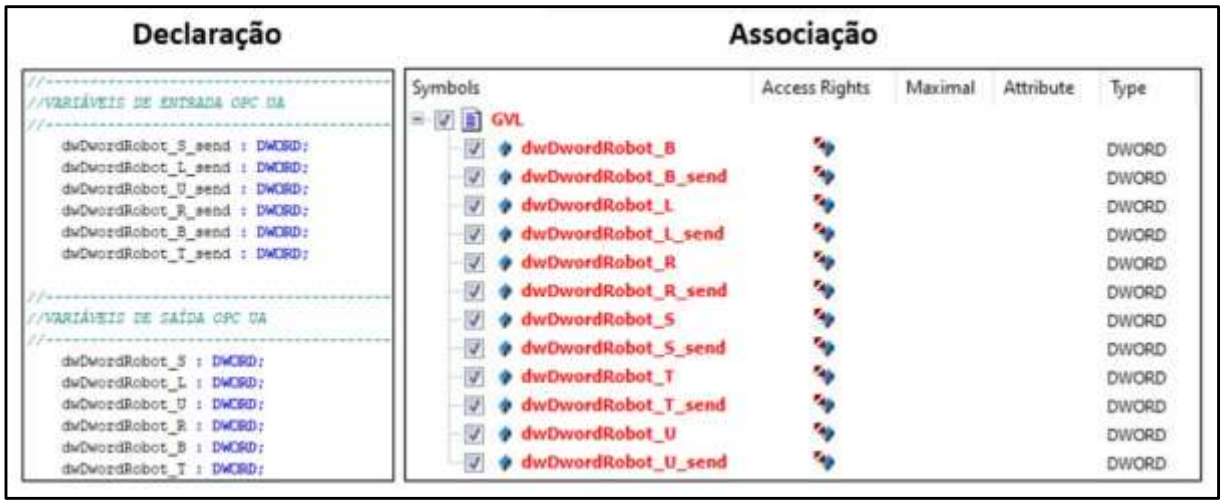


Figura 63 - Declaração e associação de variáveis OPC CP/MATLAB.

Um exemplo dessa associação, para posterior envio ao MATLAB através de variáveis OPC UA, pode ser observada através da figura 64 que apresenta o mapa de memória do eixo “S”.

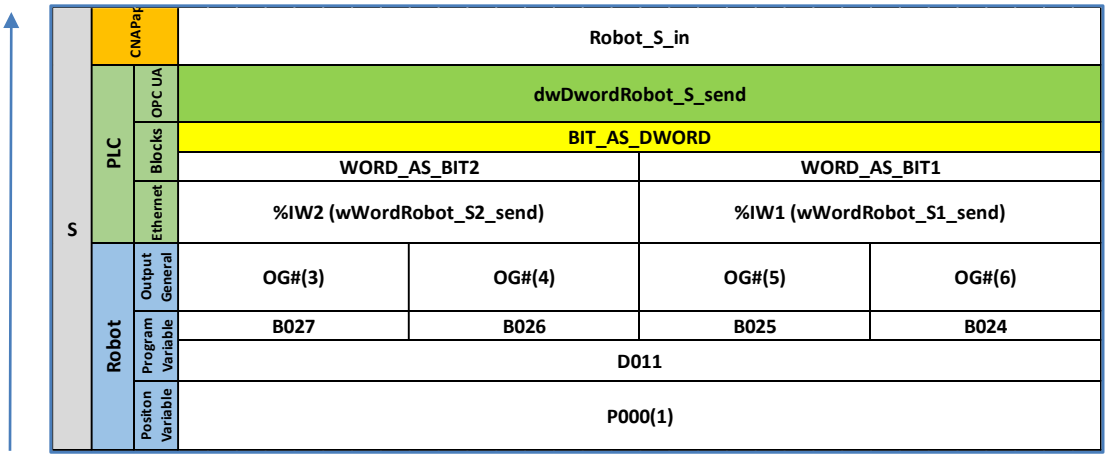


Figura 64 - Mapa de Memória com associação entre variáveis BIT_AS_DWORD e OPC UA.

Os dados são então recebidos pelo programa CNAPap que lê as variáveis OPC UA e subtrai os pulsos mínimos inseridos no algoritmo do robô MOTOMAN – GP8,

resultando nos valores de pulsos a serem aprendidos pelas CNAPap dos eixos “S”, “L”, “U”, “R” “B”. No Apêndice E pode ser encontrado o programa completo. Os dados depurados são então inseridos nas respectivas células de aprendizagem que compõem a configuração característica mostrada na figura 65.

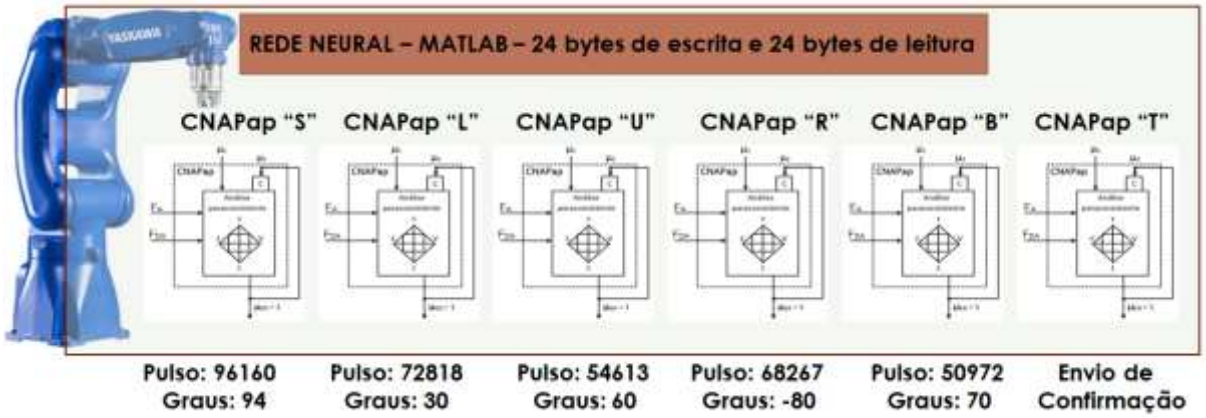


Figura 65 – Dados para aprendizagem das CNAPap dos eixos “S”, “L”, “U”, “R” “B”.

Ao ser finalizado o treinamento, o algoritmo plota os gráficos dos eixos “S”, “L”, “U”, “R” e “B” assim como o modelo cinemático da posição de cada eixo. Essa programação também se encontra no Apêndice E.

O fluxograma do programa CNAPap com essa ação está apresentado na figura 66.

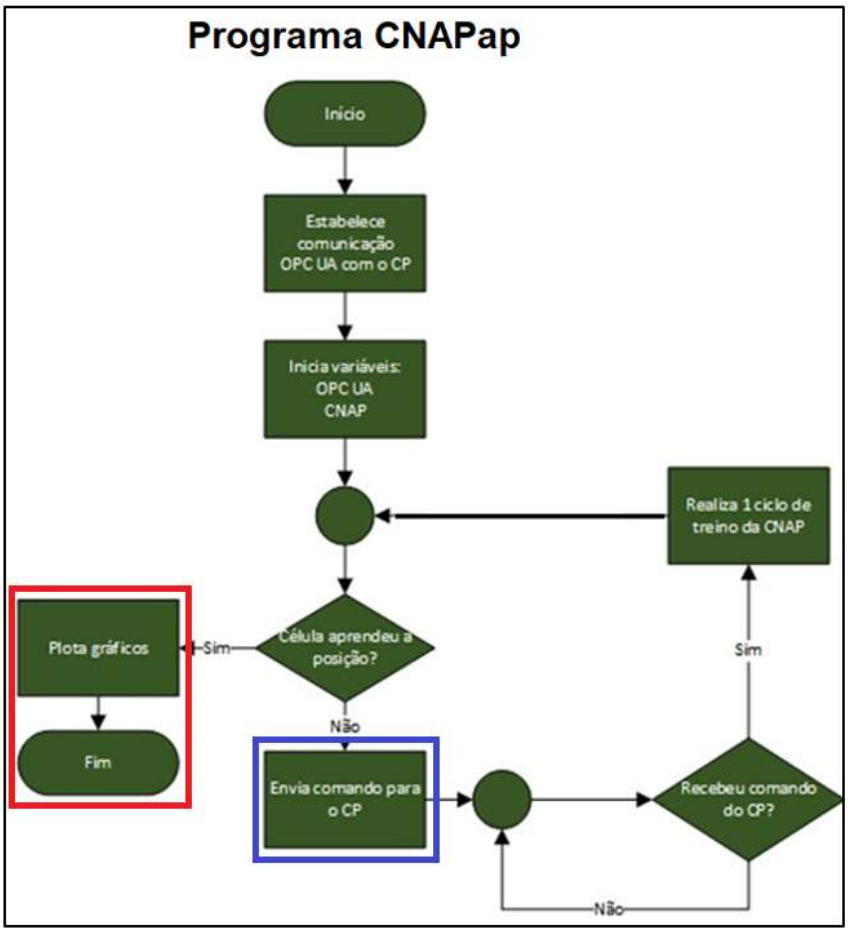


Figura 66 - CNAPap - Fluxograma com a finalização da aprendizagem.

2.4 Procedimentos dos Testes

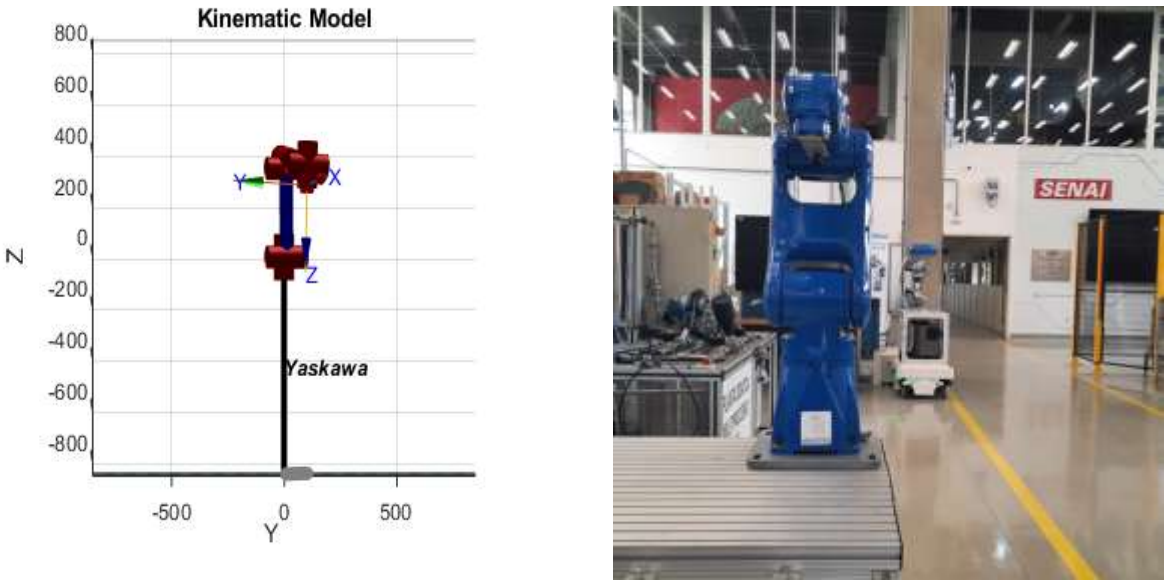
Vários testes para verificação da eficiência do aprendizado e respostas através da imitação, foram efetuados com o Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v aplicado ao Robô Industrial MOTOMAN GP8. Para demonstrar a efetividade do SPApD-LPA2v em um robô industrial de seis graus de liberdade foram selecionados os resultados obtidos com três tipos de testes. Os tipos de testes foram classificados pelo número de iterações utilizados no processo de aprendizagem em uma ação de movimentação dos seis eixos simultaneamente. Portanto, os resultados mostrados no capitulo posterior serão correspondentes aos processos de aprendizado por demonstração que utilizaram o conjunto de células aplicando 10 iterações no teste tipo 1, 11 iterações no teste tipo 2 e 15 iterações no teste tipo 3.

3 RESULTADOS E DISCUSSÃO

Os resultados práticos relacionados a movimentação dos eixos do Robô Industrial MOTOMAN – GP8 a partir do método aplicado pelo SPApD-LPA2v (Sistema Paraconsistente de Aprendizagem por Demonstração) são mostrados a seguir.

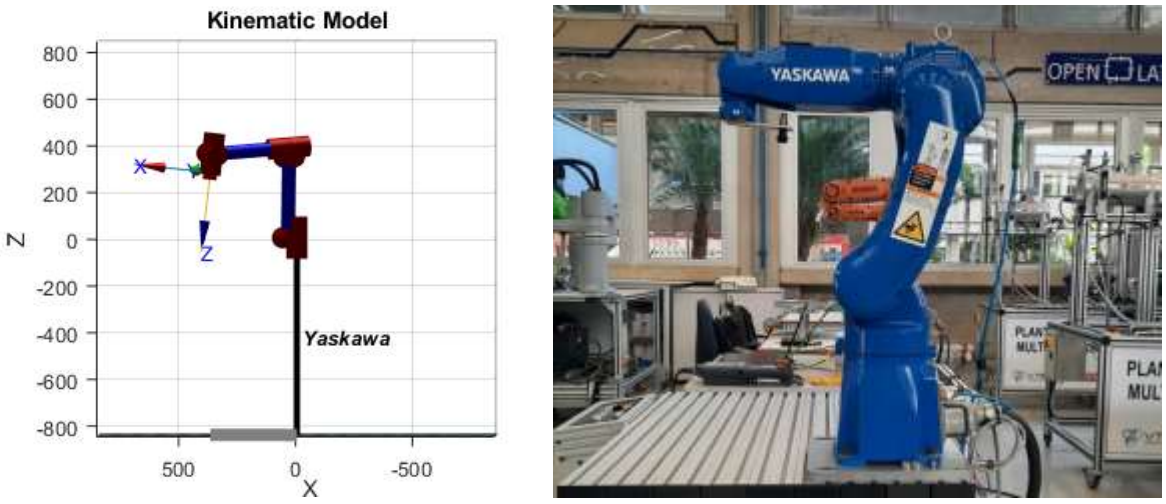
Conforme exposto no capítulo anterior os resultados mostrados serão correspondentes aos testes de aprendizado e de imitação que utilizaram 10 iterações (tipo 1), 11 iterações (tipo 2) e 15 iterações (tipo 3). Portanto, os valores correspondentes ao processo de aprendizagem e de resposta no processo de imitação foram medidos e verificados em cada um dos seis eixos do Robô Industrial para cada um dos três tipos classificados pela quantidade de iterações no aprendizado.

Inicialmente nas figuras 67, 68 e 69 são mostrados o ambiente dos testes com detalhes para as posições iniciais do robô com as respectivas vistas frontal, lateral e isométrica plotadas pelo MATLAB e fotografados na planta real.



(a) ***(b)***

Figura 67 - Vista frontal inicial.
a) Vista frontal inicial plotada pelo MATLAB.
b) Vista frontal inicial fotografada no ambiente real.



(a) *(b)*

Figura 68 – Vista lateral inicial.
a) Vista lateral inicial plotada pelo MATLAB.
b) Vista lateral inicial fotografada no ambiente real.

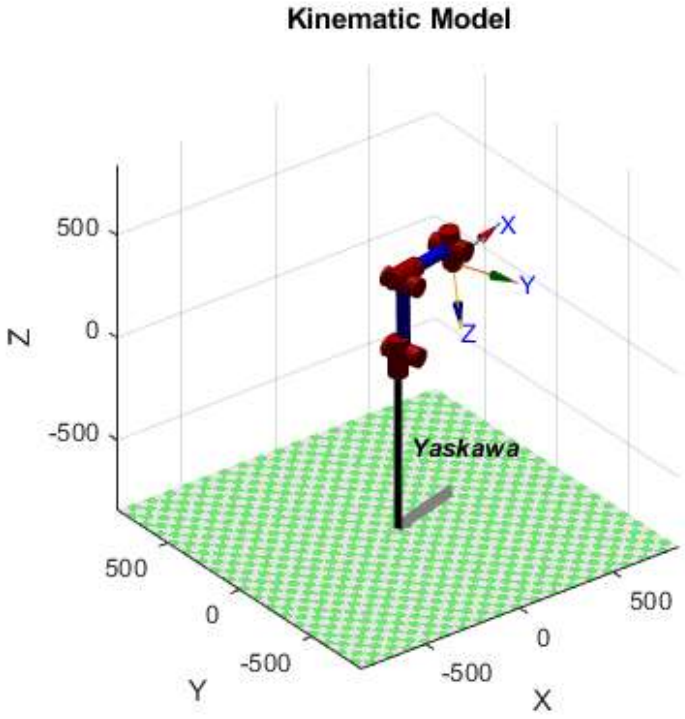
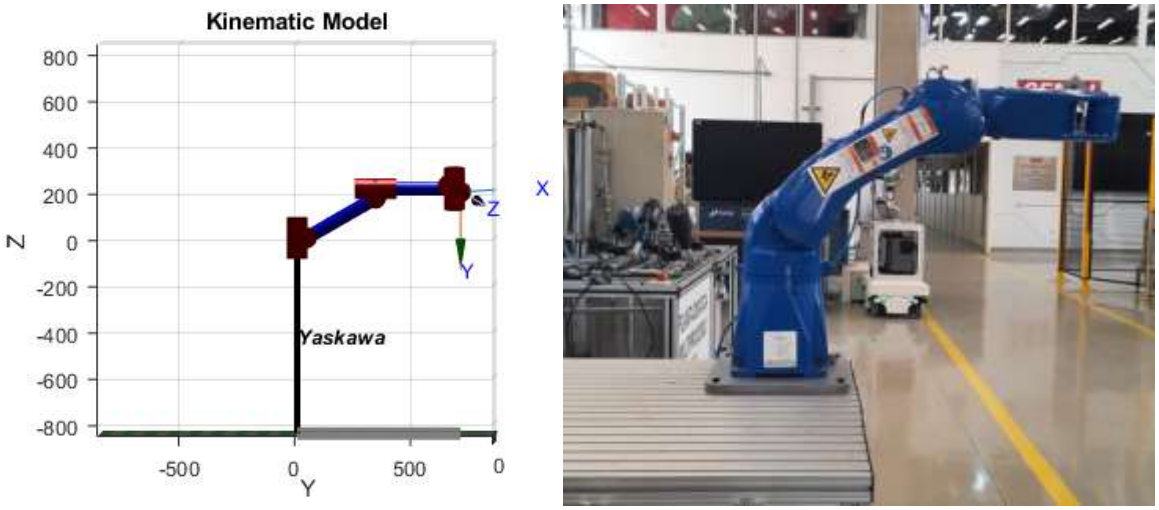


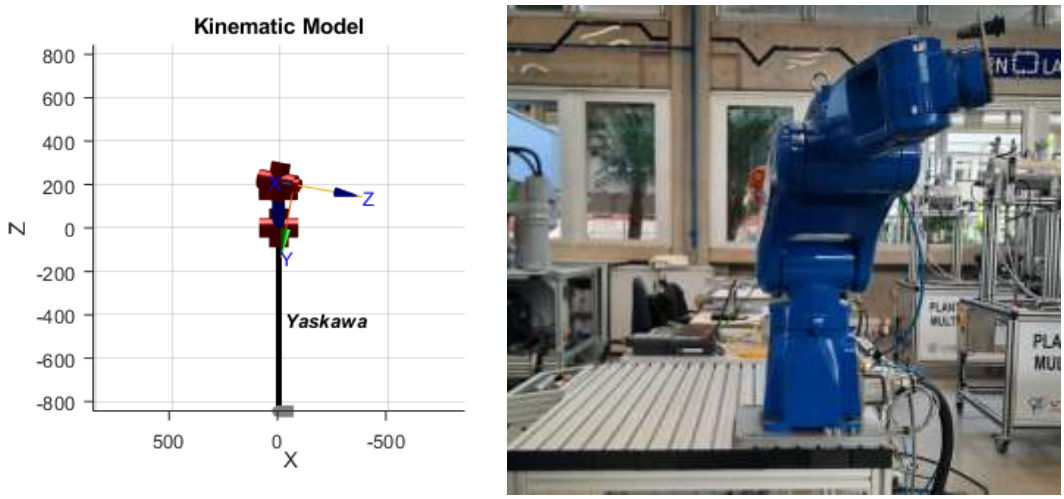
Figura 69 – Vista isométrica inicial.

A posição final do robô com as respectivas vistas frontal, lateral e isométrica alteradas, e aprendidas pelas CNAPs dos eixos “S”, “L”, “U”, “R” e “B”, estão apresentadas nas figuras 70, 71 e 72.



(a) *(b)*

Figura 70 – Vista frontal final.
a) Vista frontal final plotada pelo MATLAB.
b) Vista frontal final fotografada no ambiente real.



(a) *(b)*

Figura 71 – Vista lateral final.
a) Vista lateral final plotada pelo MATLAB.
b) Vista lateral final fotografada no ambiente real.

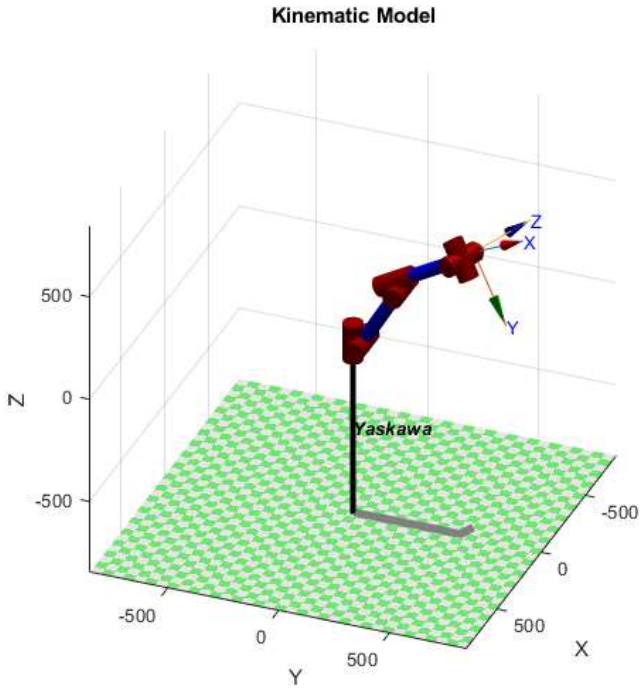


Figura 72 – Vista isométrica final.

A seguir, são apresentados os resultados obtidos na aplicação do Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v no Robô Industrial MOTOMAN – GP8. Os resultados referem-se aos valores em cada eixo com a utilização de três diferentes tipos de testes correspondentes a 10, 11 e 15 iterações.

Inicialmente, o processo de Aprendizagem por Demonstração do Eixo “S”, com ângulo de 94° correspondente a 96160 pulsos, tem seus dados completos apresentados na tabela 6.

Tabela 6 - Valores do processo completo de aprendizagem Eixo "S".

Interação	Padrão μ_1	Estados Lógicos Paraconsistentes G_c	G_{ct}	Saída $\mu_{E\,(K+1)}$	% de Erro Estado Final	Observações
0	96160	0,000000000	-1,000000000	48081		**Início Aprendizagem**
1	96160	0,500010399	0,499989601	72121		
2	96160	0,750010399	0,249989601	84141		
3	96160	0,875010399	0,124989601	90151		
4	96160	0,937510399	0,062489601	93156		
5	96160	0,968760399	0,031239601	94658		
6	96160	0,984380200	0,015619800	95409		
7	96160	0,992190100	0,007809900	95785		
8	96160	0,996100250	0,003899750	95973		
9	96160	0,998055324	0,001944676	96067		
10	**96160**	**0,999032862**	**0,000967138**	**96114**	**0,048357**	**Resultado 10 iterações (teste 1)**
11	**96160**	**0,999521631**	**0,000478369**	**96137**	**0,023918**	**Resultado 11 iterações (teste 2)**
12	96160	0,999760816	0,000239185	96149		
13	96160	0,999880408	0,000119592	96155		
14	96160	0,999940204	0,000059796	96158		
15	**96160**	**0,999970102**	**0,000029898**	**96159**	**0,001040**	**Resultado 15 iterações (teste 3)**

Os resultados gráficos da aprendizagem do eixo "S" com a respectiva porcentagem de erro para as 10, 11 e 15 iterações são apresentados a seguir.

Teste 1 – 10 iterações no aprendizado

a- Aprendizagem e erro resultante após os 10 passos, estão apresentados nos gráficos das figuras 73 e 74, respectivamente.

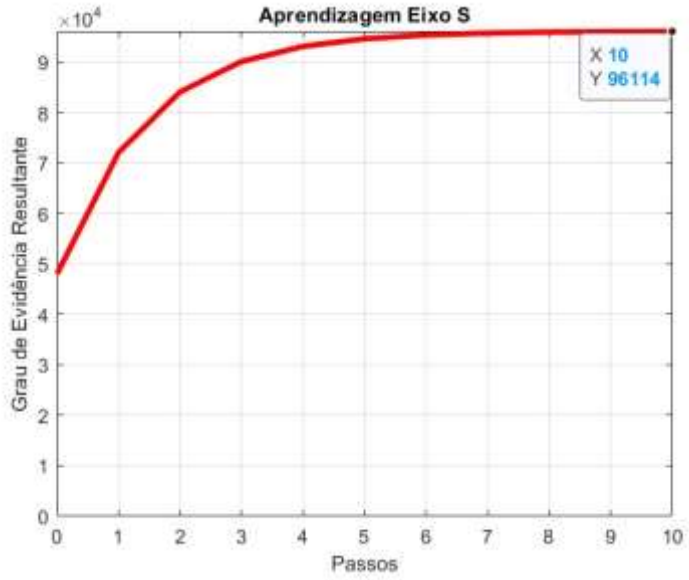


Figura 73 – Resultados gráficos do Aprendizado com 10 passos para o eixo S.

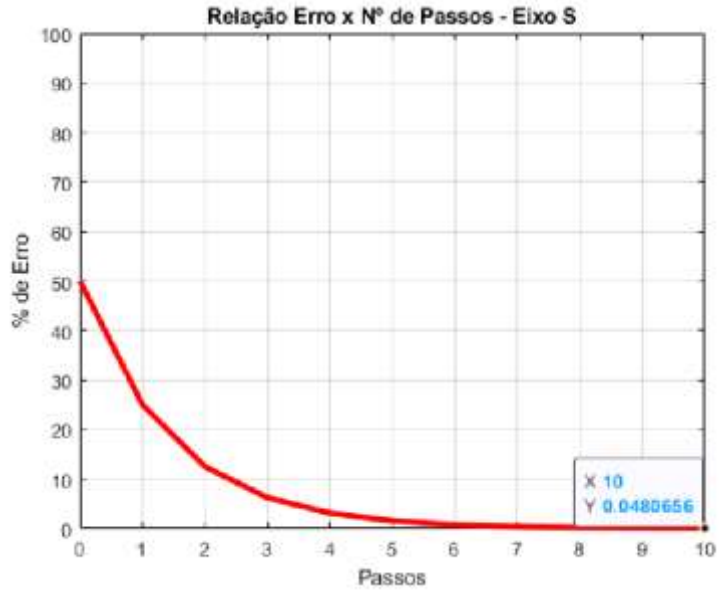


Figura 74 – Porcentagem de erro do Aprendizado com 10 passos para o eixo S.

Teste 2 – 11 iterações no aprendizado

b- Aprendizagem e erro resultante após os 11 passos, estão apresentados nos gráficos das figuras 75 e 76, respectivamente.

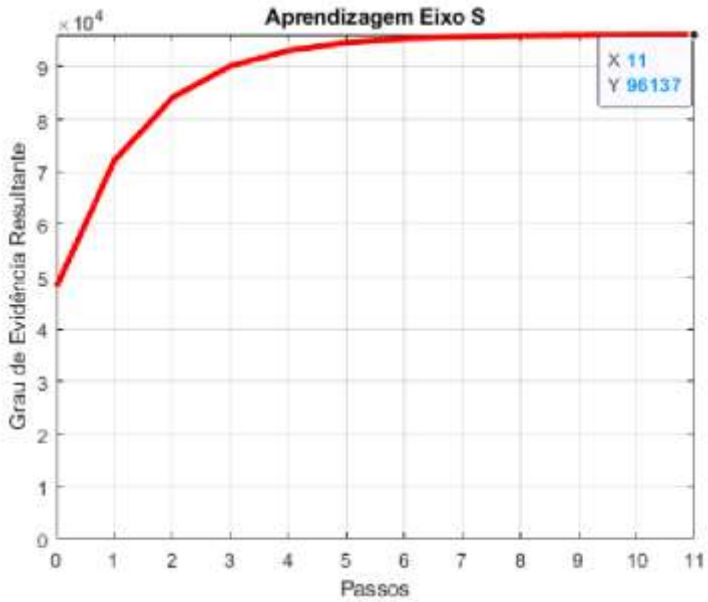


Figura 75 – Resultados gráficos do Aprendizado com 11 passos para o eixo S.

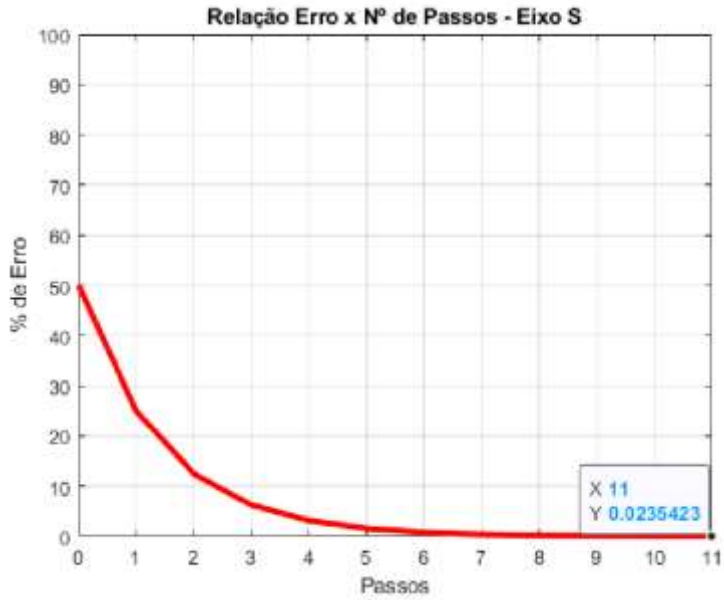


Figura 76 – Porcentagem de erro do Aprendizado com 11 passos para o eixo S.

Teste 3 – 15 iterações no aprendizado

c- Aprendizagem e erro resultante após os 15 passos, estão apresentados nos gráficos das figuras 77 e 78, respectivamente.

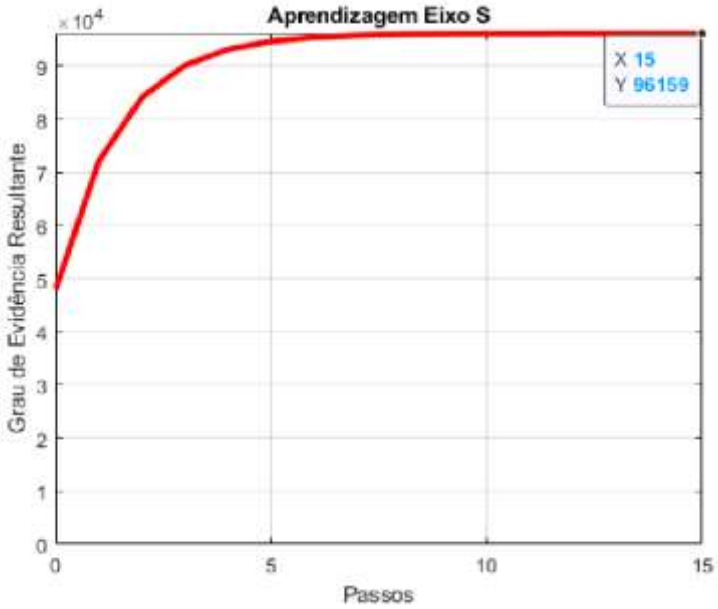


Figura 77 – Resultados gráficos do Aprendizado com 15 passos para o eixo S.

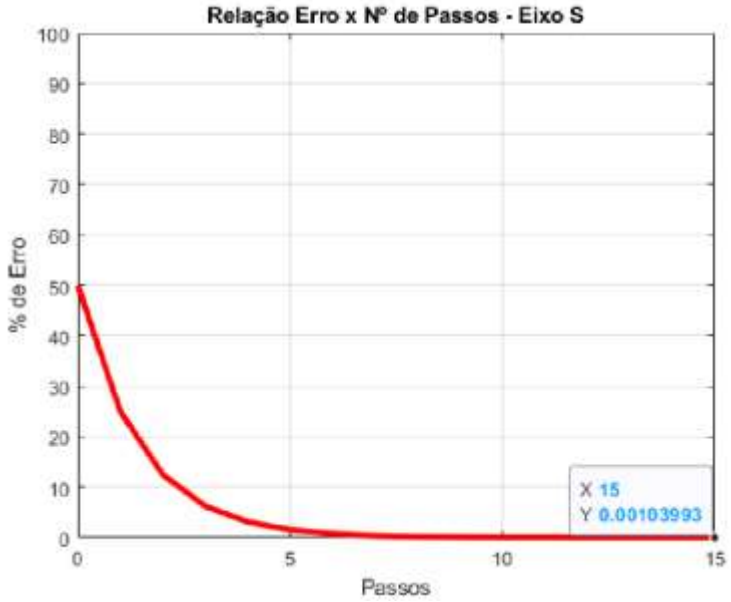


Figura 78 – Porcentagem de erro do Aprendizado com 15 passos para o eixo S.

O Aprendizagem do Eixo “L”, com ângulo de 30° correspondente a 72818 pulsos, tem seus dados completos apresentados na tabela 7.

Tabela 7 - Valores do processo completo de aprendizagem Eixo "L".

Interação	Padrão μ_1	Estados Lógicos Paraconsistentes G_c	G_{ct}	Saída $\mu_{E\,(K+1)}$	% de Erro Estado Final	Observações
0	72818	0,000000000	-1,000000000	36410		**Início Aprendizagem**
1	72818	0,500013733	0,499986267	54614		
2	72818	0,750006866	0,249993134	63716		
3	72818	0,875003433	0,124996567	68267		
4	72818	0,937501717	0,062498283	70543		
5	72818	0,968757725	0,031242275	71681		
6	72818	0,984385729	0,015614271	72250		
7	72818	0,992199731	0,007800269	72534		
8	72818	0,996099865	0,003900135	72676		
9	72818	0,998049933	0,001950067	72747		
10	**72818**	**0,999024966**	**0,000975034**	**72783**	**0,048752**	**Resultado 10 iterações (teste 1)**
11	**72818**	**0,999519350**	**0,000480650**	**72801**	**0,024033**	**Resultado 11 iterações (teste 2)**
12	72818	0,999759675	0,000240325	72810		
13	72818	0,999879838	0,000120163	72814		
14	72818	0,999939919	0.000060081	72816		
15	**72818**	**0,999969959**	**0.0000300406**	**72817**	**0,001373**	**Resultado 15 iterações (teste 3)**

Os resultados gráficos da aprendizagem do eixo "L", com a respectiva porcentagem de erro para as 10, 11 e 15 iterações são apresentados a seguir.

Teste 1 – 10 iterações no aprendizado

a- Aprendizagem e erro resultante do eixo "L" após os 10 passos, estão nos gráficos das figuras 79 e 80, respectivamente.

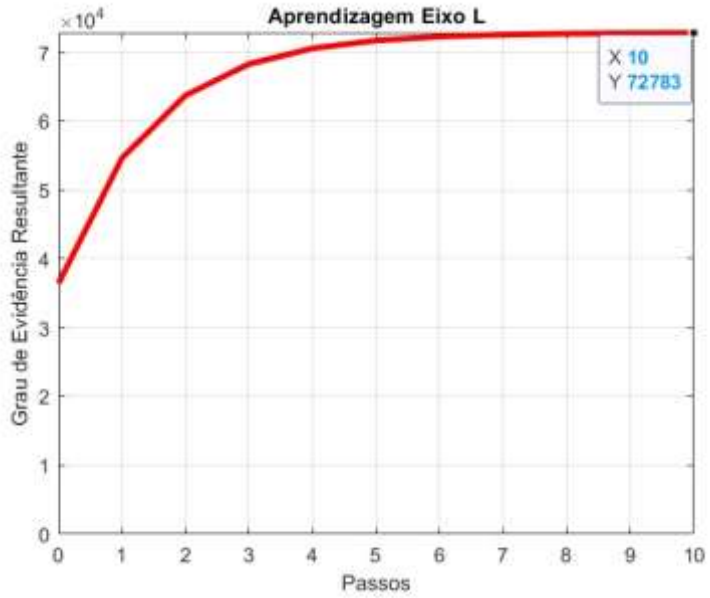


Figura 79 – Resultados gráficos do Aprendizado com 10 passos para o eixo L.

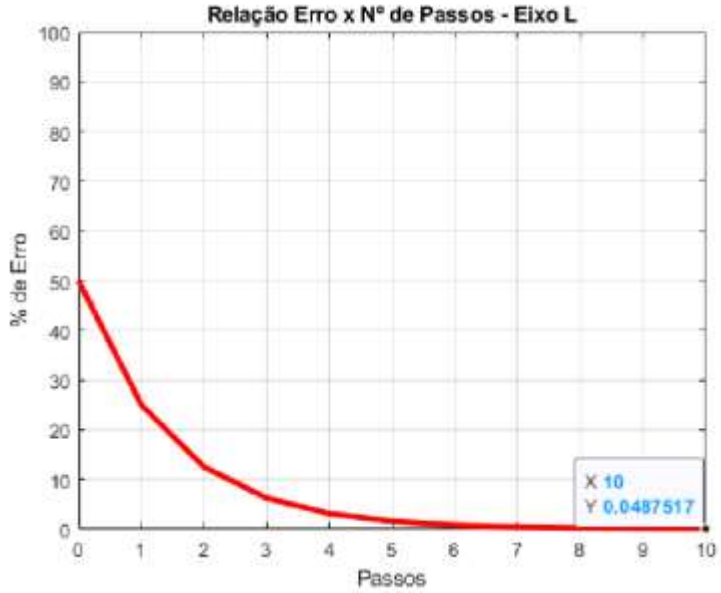


Figura 80 – Porcentagem de erro do Aprendizado com 10 passos para o eixo L.

Teste 2 – 11 iterações no aprendizado

b- Aprendizagem e erro resultante do eixo “L” após os 11 passos, estão nos gráficos das figuras 81 e 82, respectivamente.

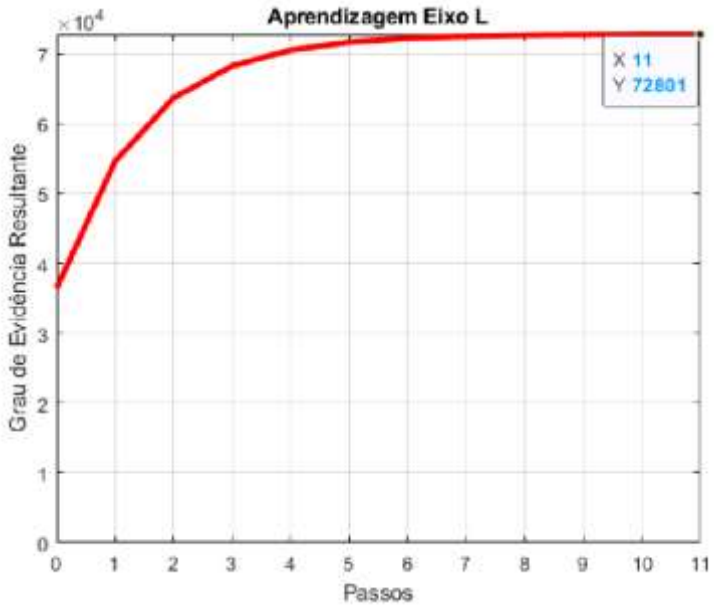


Figura 81 – Resultados gráficos do Aprendizado com 11 passos para o eixo L.

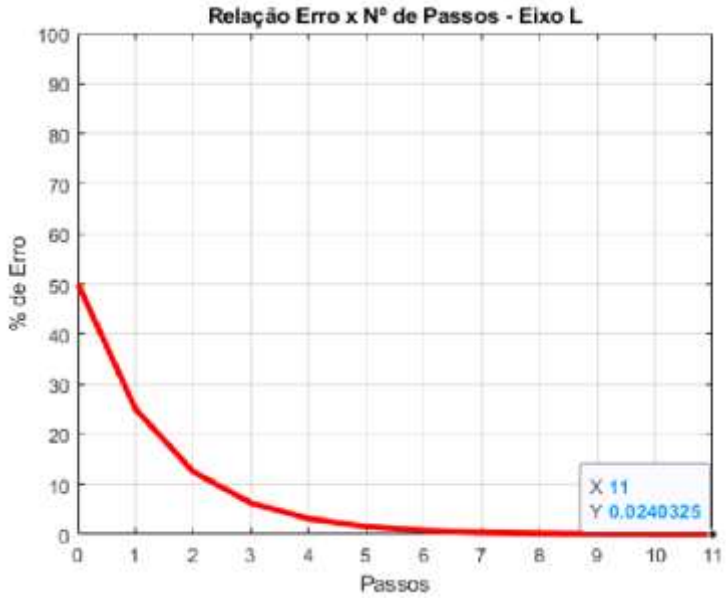


Figura 82 – Porcentagem de erro do Aprendizado com 11 passos para o eixo L.

Teste 3 – 15 iterações no aprendizado

c- Aprendizagem e erro resultante do eixo “L” após os 15 passos, estão nos gráficos das figuras 83 e 84, respectivamente.

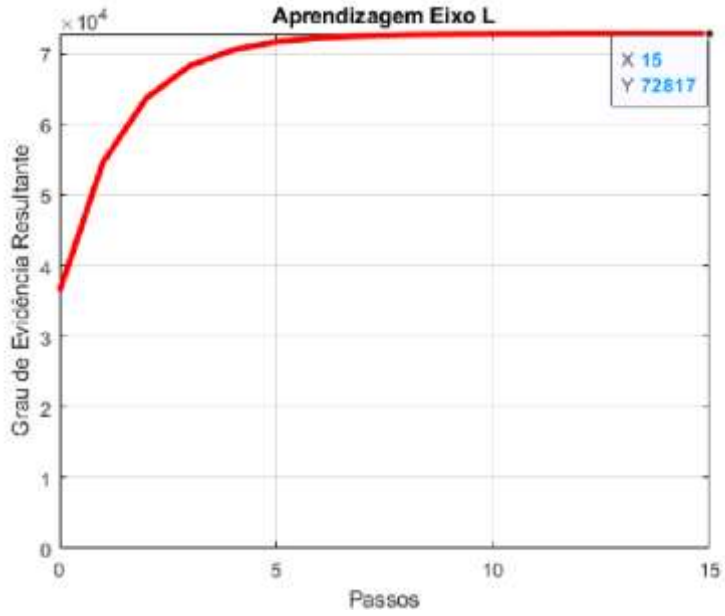


Figura 83 – Resultados gráficos do Aprendizado com 15 passos para o eixo L.

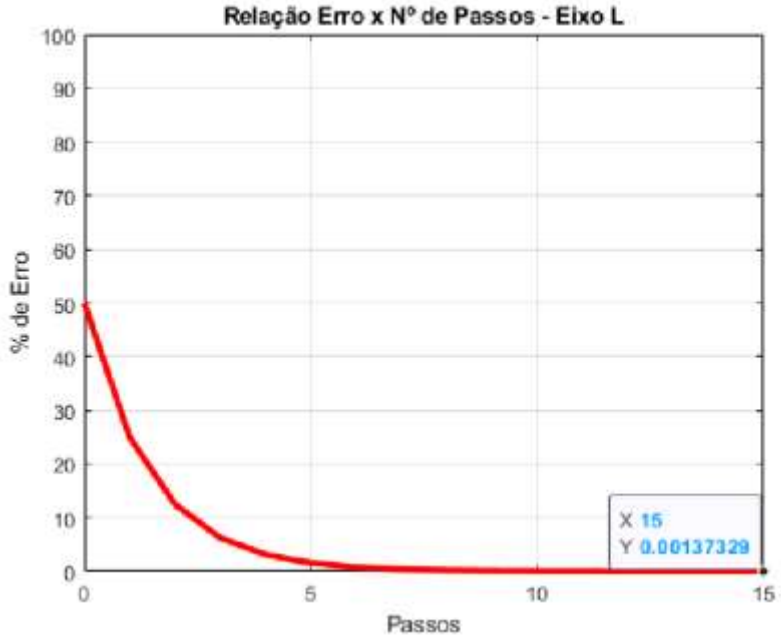


Figura 84 – Porcentagem de erro do Aprendizado com 15 passos para o eixo L.

A seguir, o Aprendizagem do Eixo “U”, com ângulo de 60° correspondente a 54613 pulsos, tem seus dados completos apresentados na tabela 8.

Tabela 8 - Valores do processo completo de aprendizagem Eixo "U".

Interação	Padrão μ_1	Estados Lógicos Paraconsistentes G_c	G_{ct}	Saída $\mu_{E\,(K+1)}$	% de Erro Estado Final	Observações
0	54613	0,000000000	-1,000000000	27307		**Início Aprendizagem**
1	54613	0,500009155	0,499990845	40960		
2	54613	0,750004578	0,249995422	47787		
3	54613	0,875011444	0,124988556	51200		
4	54613	0,937505722	0,062494278	52907		
5	54613	0,968762016	0,031237984	53760		
6	54613	0,984381008	0,015618992	54187		
7	54613	0,992199659	0,007800341	54400		
8	54613	0,996099830	0,003900170	54507		
9	54613	0,998059070	0,001940930	54560		
10	**54613**	**0,999029535**	**0,000970465**	**54587**	**0,048523**	**Resultado 10 iterações (teste 1)**
11	**54613**	**0,999523923**	**0,000476077**	**54600**	**0,023804**	**Resultado 11 iterações (teste 2)**
12	54613	0,999514768	0,000485233	54607		
13	54613	0,999757384	0,000242616	54610		
14	54613	0,999878692	0,000121308	54612		
15	**54613**	**0,999939346**	**0,0000606541**	**54613**	**0,000916**	**Resultado 15 iterações (teste 3)**

Os resultados gráficos da aprendizagem do eixo "U", com a respectiva porcentagem de erro para as 10, 11 e 15 iterações são apresentados a seguir.

Teste 1 – 10 iterações no aprendizado

a- Aprendizagem e erro resultante do eixo "U" após os 10 passos podem ser observados nos gráficos das figuras 85 e 86, respectivamente.

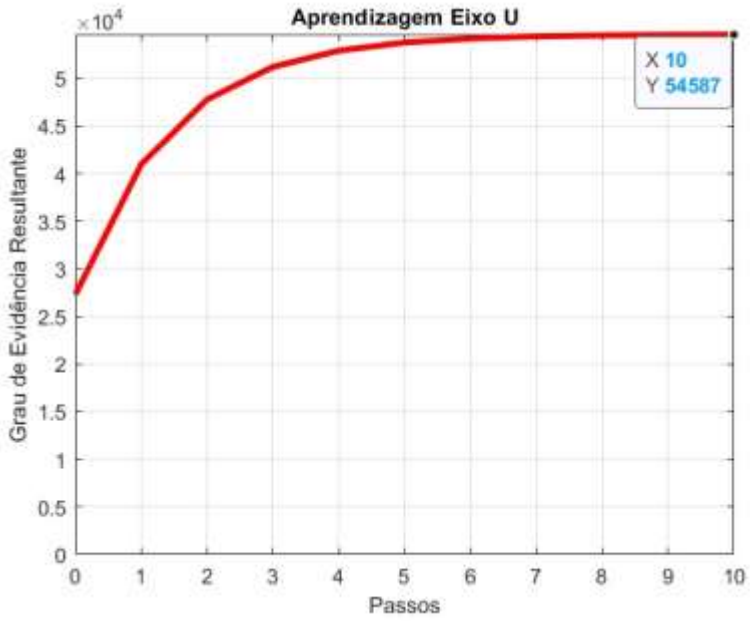


Figura 85 – Resultados gráficos do Aprendizado com 10 passos para o eixo U.

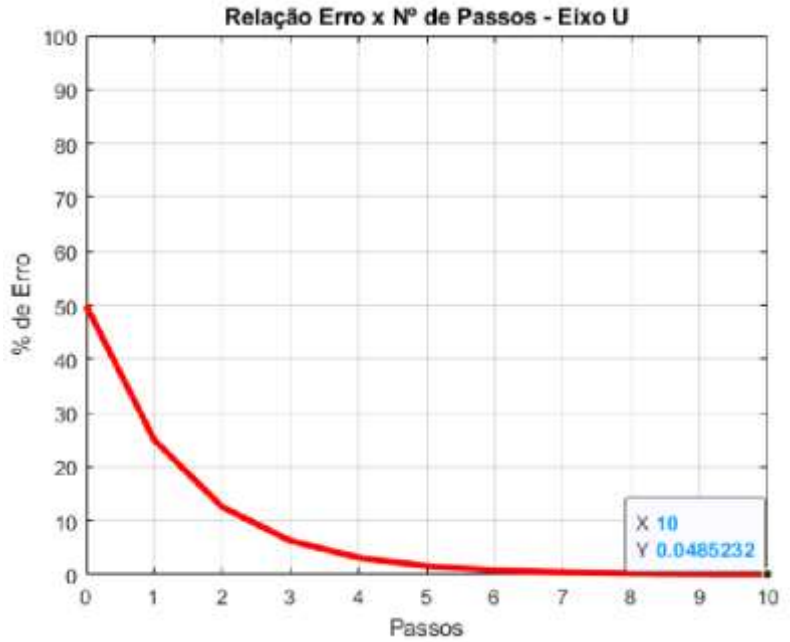


Figura 86 – Porcentagem de erro do Aprendizado com 10 passos para o eixo U.

Teste 2 – 11 iterações no aprendizado

b- Aprendizagem e erro resultante do eixo “U” após os 11 passos podem ser observados nos gráficos das figuras 87 e 88, respectivamente.

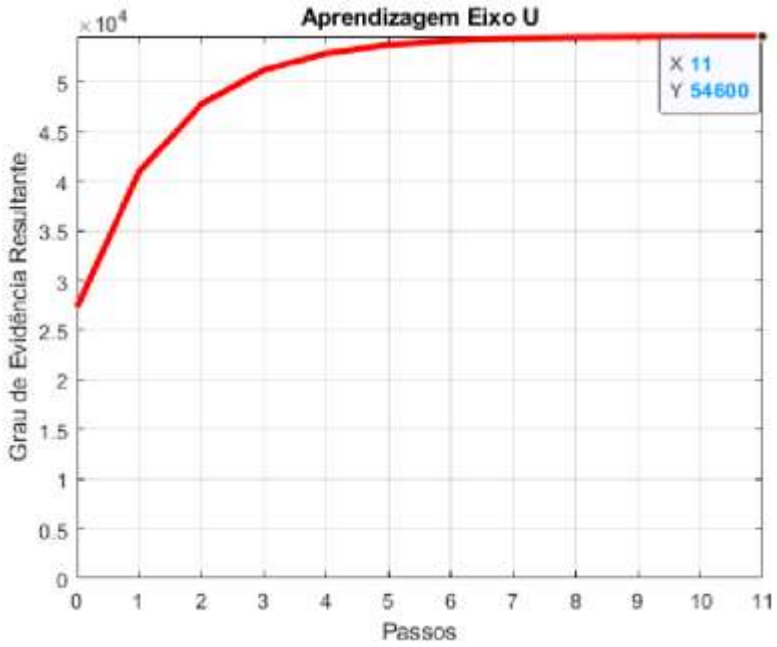


Figura 87 – Resultados gráficos do Aprendizado com 11 passos para o eixo U.

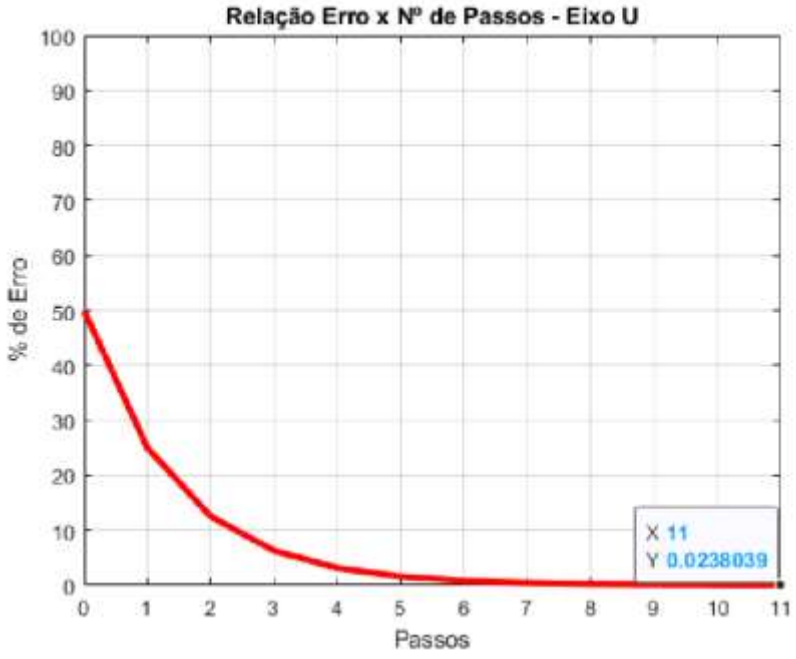


Figura 88 – Porcentagem de erro do Aprendizado com 11 passos para o eixo U.

Teste 3 – 15 iterações no aprendizado

c- Aprendizagem e erro resultante do eixo “U” após os 15 passos podem ser observados nos gráficos das figuras 89 e 90, respectivamente.

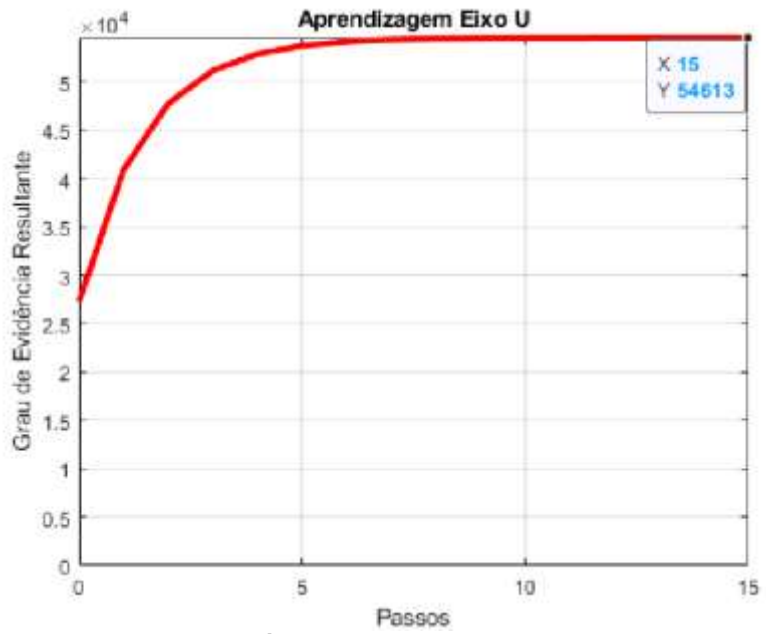


Figura 89 – Resultados gráficos do Aprendizado com 15 passos para o eixo U.

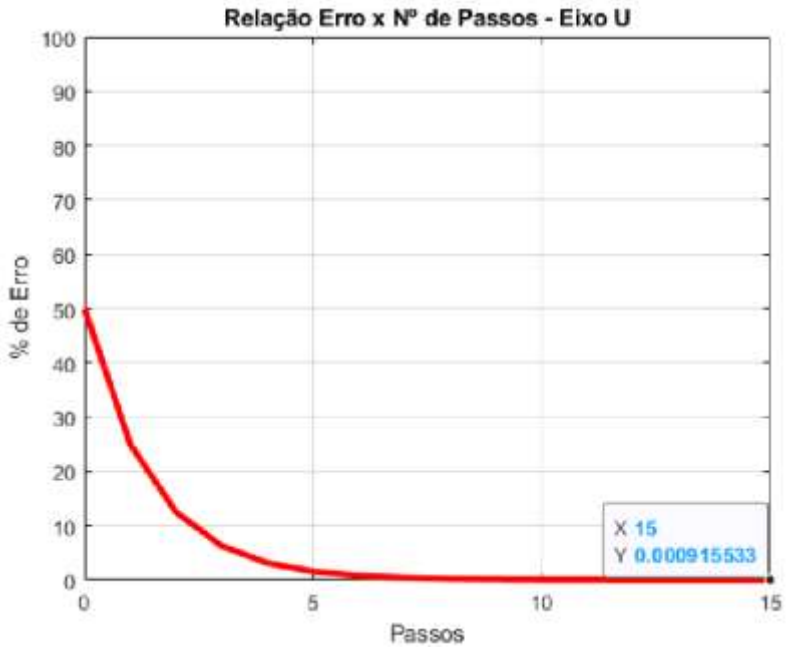


Figura 90 – Porcentagem de erro do Aprendizado com 15 passos para o eixo U.

A seguir, o Aprendizagem do Eixo “R”, com ângulo de 80° correspondente a 68267 pulsos, tem seus dados completos apresentados na tabela 9.

Tabela 9 - Valores do processo completo de aprendizagem Eixo "R".

Interação	Padrão μ_1	Estados Lógicos Paraconsistentes		Saída $\mu_{E\,(K+1)}$	% de Erro Estado Final	Observações
		G_c	G_{ct}			
0	68267	0,000000000	-1,000000000	34134		**Início Aprendizagem**
1	68267	0,500007324	0,499992676	51201		
2	68267	0,750010986	0,249989014	59734		
3	68267	0,875005493	0,124994507	64001		
4	68267	0,937510071	0,062489929	66134		
5	68267	0,968755035	0,031244965	67201		
6	68267	0,984384842	0,015615158	67734		
7	68267	0,992192421	0,007807579	68001		
8	68267	0,996103535	0,003896465	68134		
9	68267	0,998051767	0,001948233	68201		
10	**68267**	**0,999033208**	**0,000966792**	**68234**	**0,048340**	**Resultado 10 iterações (teste 1)**
11	**68267**	**0,999516604**	**0,000483396**	**68251**	**0,024170**	**Resultado 10 iterações (teste 2)**
12	68267	0,999758302	0,000241698	68259		
13	68267	0,999879151	0,000120849	68263		
14	68267	0,999939576	0,000060424	68265		
15	**68267**	**0,999969788**	**0,000030212**	**68266**	**0,001465**	**Resultado 10 iterações (teste 3)**

Os resultados gráficos da aprendizagem do eixo "R", com a respectiva porcentagem de erro para as 10, 11 e 15 iterações são apresentados a seguir.

Teste 1 – 10 iterações no aprendizado

a- Aprendizagem e erro resultante do eixo "R" após os 10 passos podem ser observados nos gráficos das figuras 91 e 92, respectivamente.

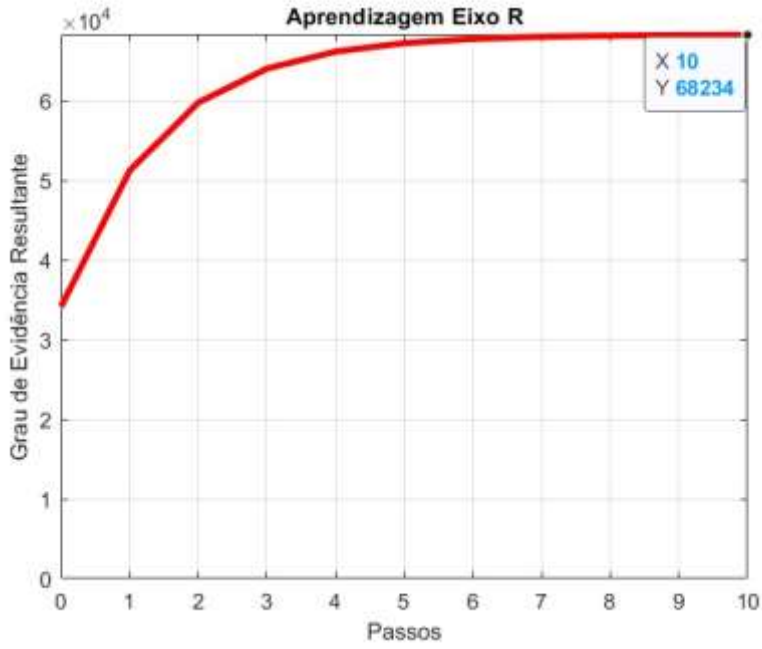


Figura 91 – Resultados gráficos do Aprendizado com 10 passos para o eixo R.

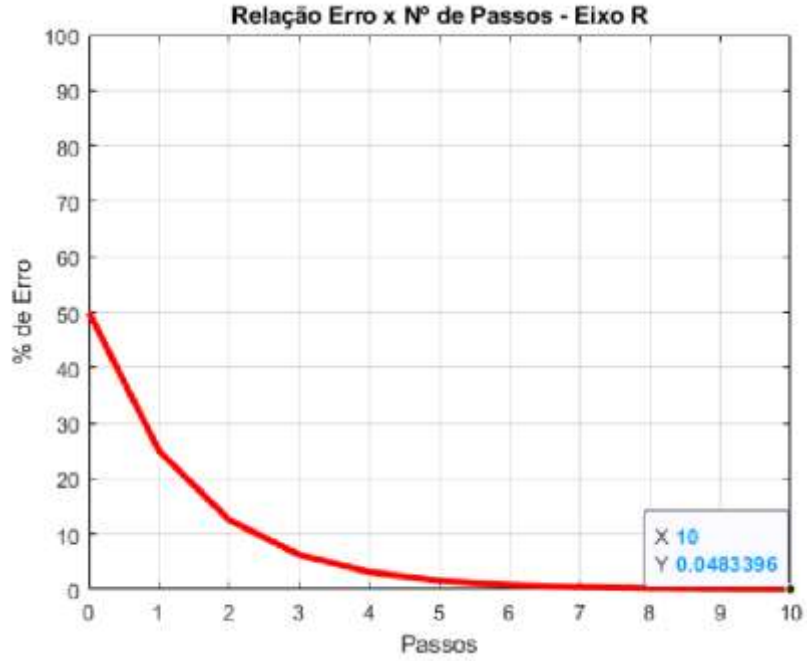


Figura 92 – Porcentagem de erro do Aprendizado com 10 passos para o eixo R.

Teste 2 – 11 iterações no aprendizado

b- Aprendizagem e erro resultante do eixo “R” após os 11 passos podem ser observados nos gráficos das figuras 93 e 94, respectivamente.

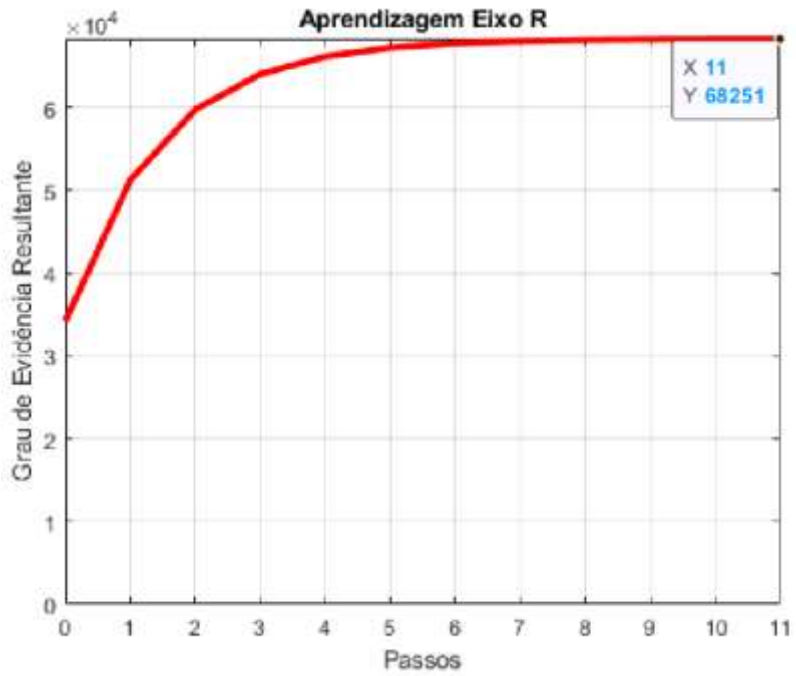


Figura 93 – Resultados gráficos do Aprendizado com 11 passos para o eixo R.

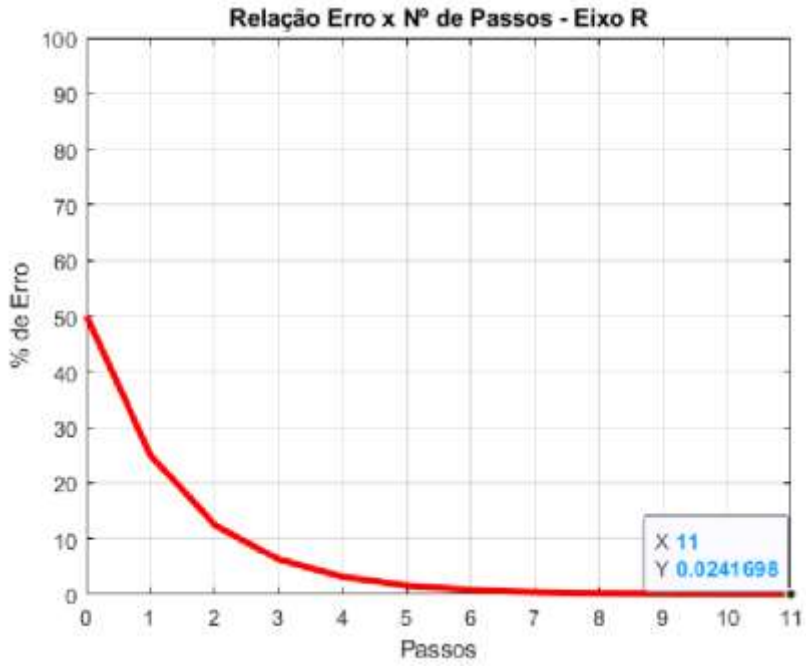


Figura 94 – Porcentagem de erro do Aprendizado com 11 passos para o eixo R.

Teste 3 – 15 iterações no aprendizado

c- Aprendizagem e erro resultante do eixo “R” após os 15 passos podem ser observados nos gráficos das figuras 95 e 96, respectivamente.

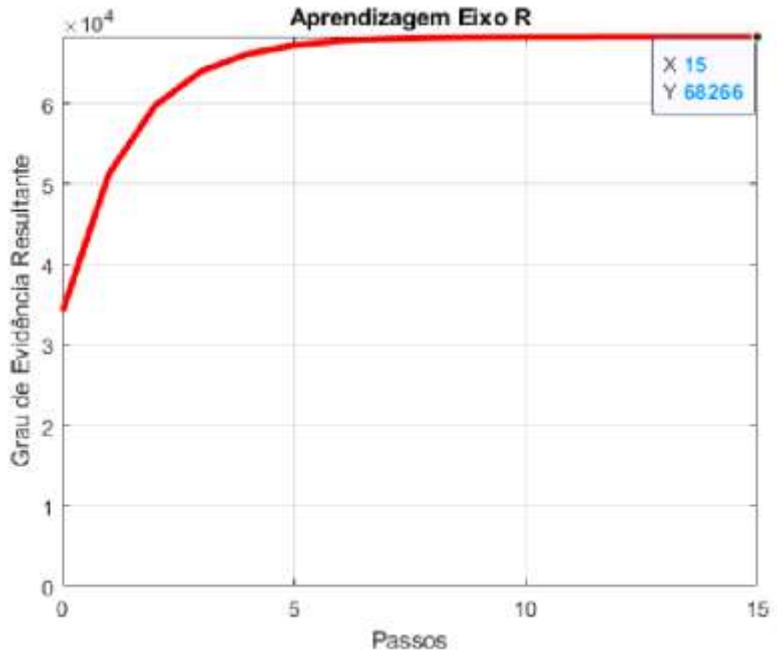


Figura 95 – Resultados gráficos do Aprendizado com 15 passos para o eixo R.

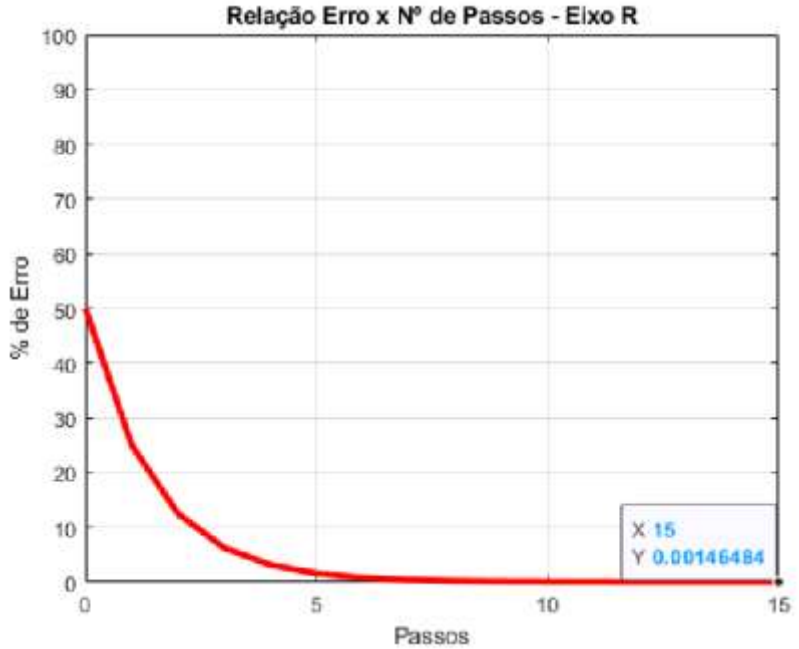


Figura 96 – Porcentagem de erro do Aprendizado com 15 passos para o eixo R.

A seguir, o Aprendizagem do Eixo “B”, com ângulo de 70° correspondente a 50972 pulsos, tem seus dados completos apresentados na tabela 10.

Tabela 10 - Valores do processo completo de aprendizagem Eixo "B".

Interação	Padrão μ_1	Estados Lógicos Paraconsistentes		Saída $\mu_{E\ (K+1)}$	% de Erro Estado Final	Observações
		G_c	G_{ct}			
0	50972	0,000000000	-1,000000000	25487		**Início Aprendizagem**
1	50972	0,500019619	0,499980381	38230		
2	50972	0,750019619	0,249980381	44601		
3	50972	0,875009809	0,124990191	47787		
4	50972	0,937514714	0,062485286	49380		
5	50972	0,968767166	0,031232834	50176		
6	50972	0,984383583	0,015616417	50574		
7	50972	0,992191792	0,007808208	50773		
8	50972	0,996095896	0,003904104	50873		
9	50972	0,998057757	0,001942243	50923		
10	**50972**	**0,999038688**	**0,000961312**	**50948**	**0,048066**	**Resultado 10 iterações (teste 1)**
11	**50972**	**0,999529153**	**0,000470847**	**50960**	**0,023542**	**Resultado 11 iterações (teste 2)**
12	50972	0,999764577	0,000235424	50966		
13	50972	0,999882288	0,000117712	50969		
14	50972	0,999941144	0,000058856	50971		
15	**50972**	**0,999970572**	**0.000029428**	**50972**	**0,000981**	**Resultado 15 iterações (teste 3)**

Os resultados gráficos da aprendizagem do eixo "B", com a respectiva porcentagem de erro para as 10, 11 e 15 iterações são apresentados a seguir.

Teste 1 – 10 iterações no aprendizado

a- Aprendizagem e erro resultante do eixo "B" após os 10 passos podem ser observados nos gráficos das figuras 97 e 98, respectivamente.

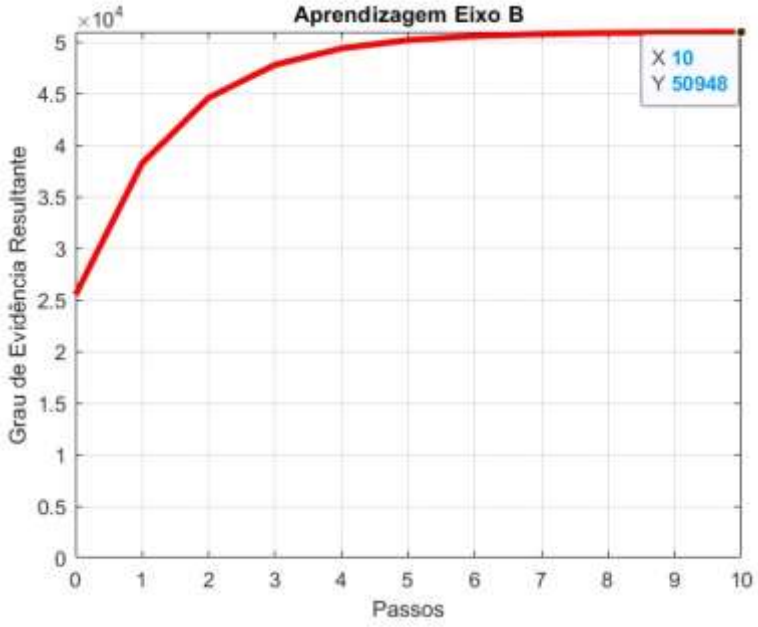


Figura 97 – Resultados gráficos do Aprendizado com 10 passos para o eixo B.

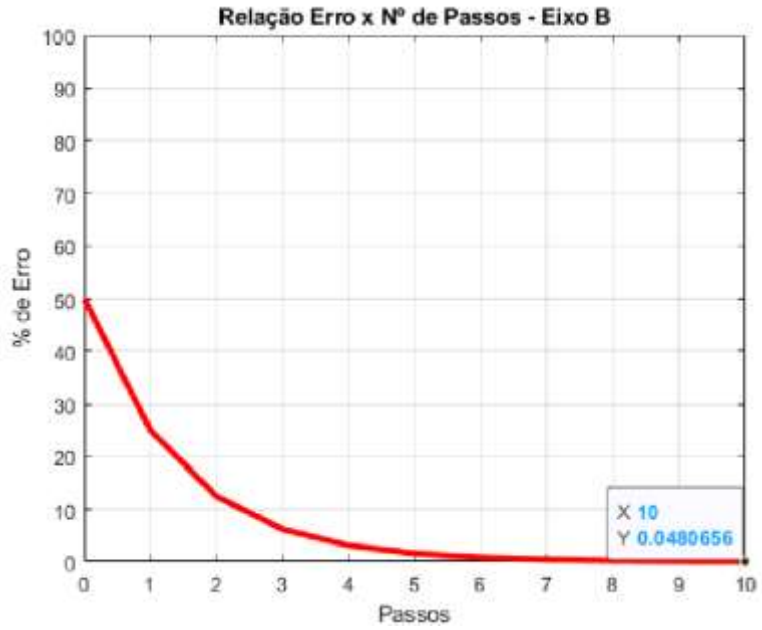


Figura 98 – Porcentagem de erro do Aprendizado com 10 passos para o eixo B.

Teste 2 – 11 iterações no aprendizado

b- Aprendizagem e erro resultante do eixo “B” após os 11 passos podem ser observados nos gráficos das figuras 99 e 100, respectivamente.

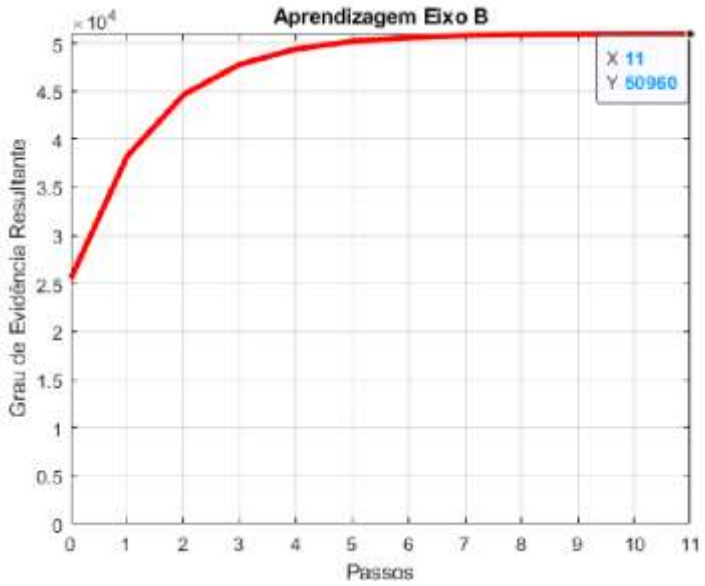


Figura 99 – Resultados gráficos do Aprendizado com 11 passos para o eixo B.

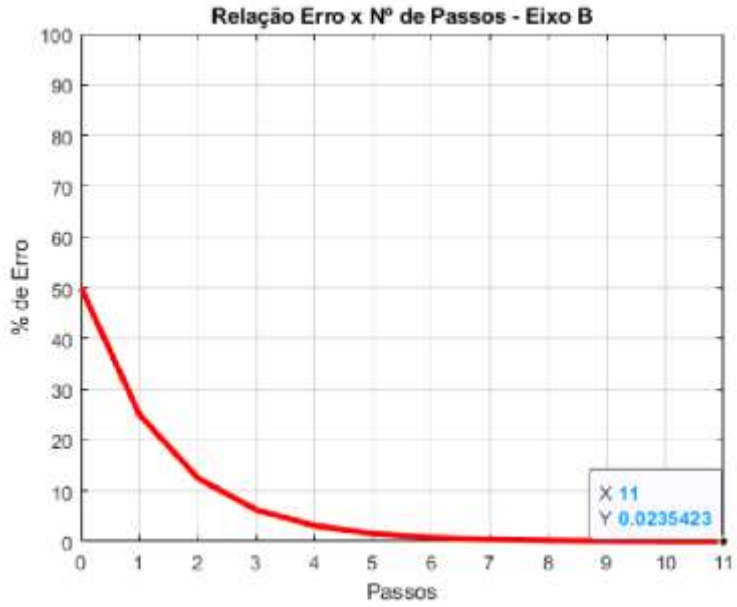


Figura 100 – Porcentagem de erro do Aprendizado com 11 passos para o eixo B.

Teste 3 – 15 iterações no aprendizado

c- Aprendizagem e erro resultante do eixo “B” após os 15 passos podem ser observados nos gráficos das figuras 101 e 102, respectivamente.

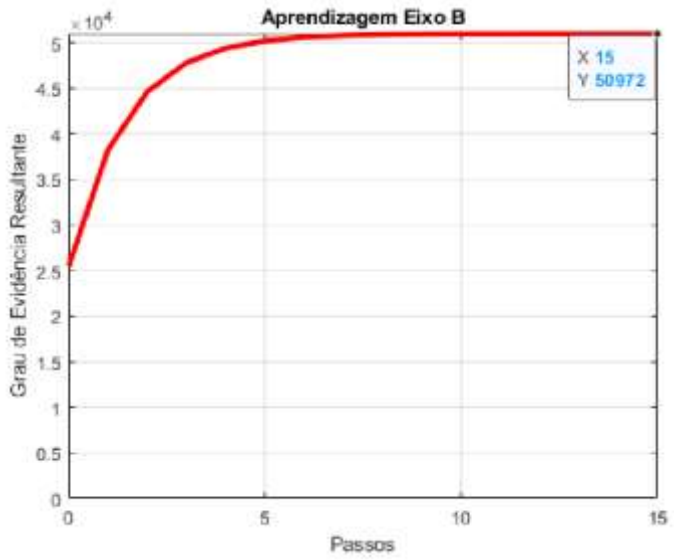


Figura 101 – Resultados gráficos do Aprendizado com 15 passos para o eixo B.

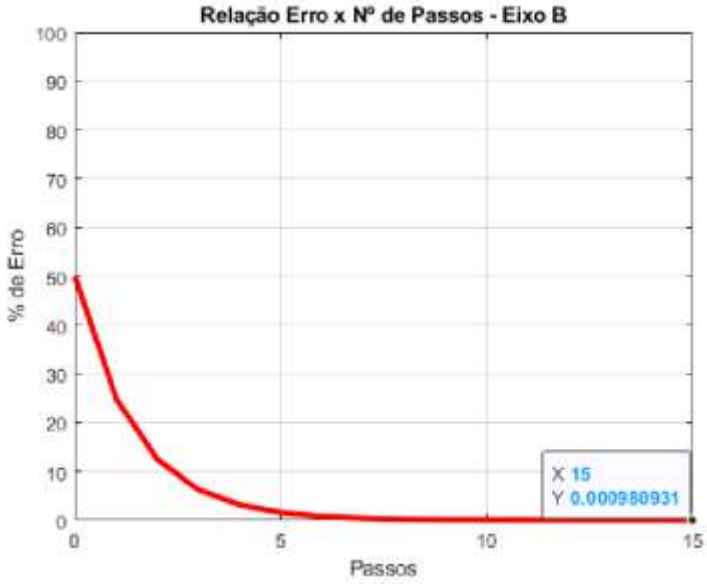


Figura 102 – Porcentagem de erro do Aprendizado com 15 passos para o eixo B.

O eixo “T” do robô não foi utilizado na movimentação e sim no envio de confirmação de fim de ciclo de aprendizagem, fato esse devido aos bytes disponíveis para utilização no YASKAWA MOTOMAN GP8 utilizado nesse trabalho.

Com os dados aprendidos disponíveis em cada fase de testes do aprendizado por demonstração (10, 11 e 15 passos), o Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v entra para a fase de Imitação. Estes valores são então enviados para o robô Industrial afim de validar o aprendizado e realizar a imitação com base nos estados resultantes obtidos na fase do Aprendizado por

Demonstração. Os respectivos resultados da fase de Imitação são apresentados nas figuras 103, 104 e 105 com os 10, 11 e 15 estados, respectivamente. Para todos os testes foi mantida a velocidade padrão no Robô industrial YASKAWA em 5rad/s.

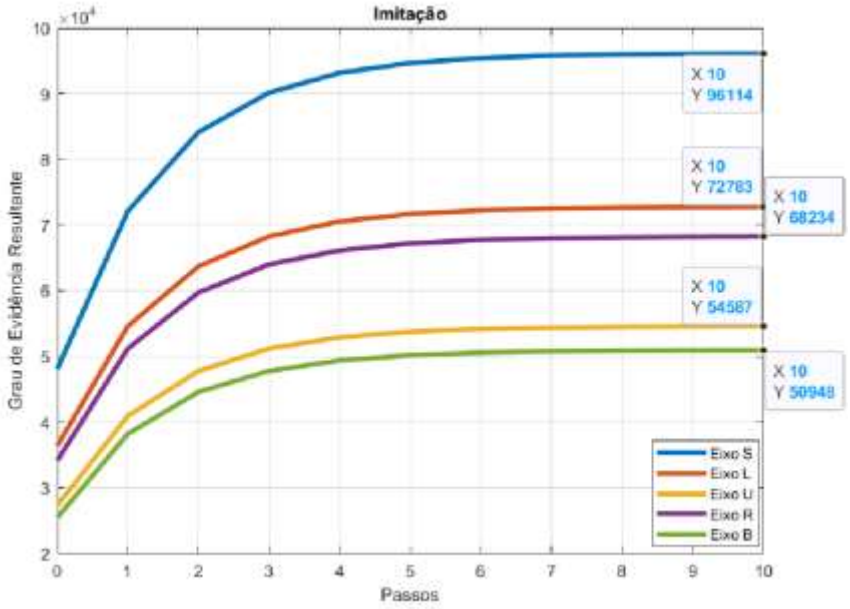


Figura 103 – SPApD-LPA2v - Imitação a partir da aprendizagem por demonstração efetuada com 10 passos (teste 1).

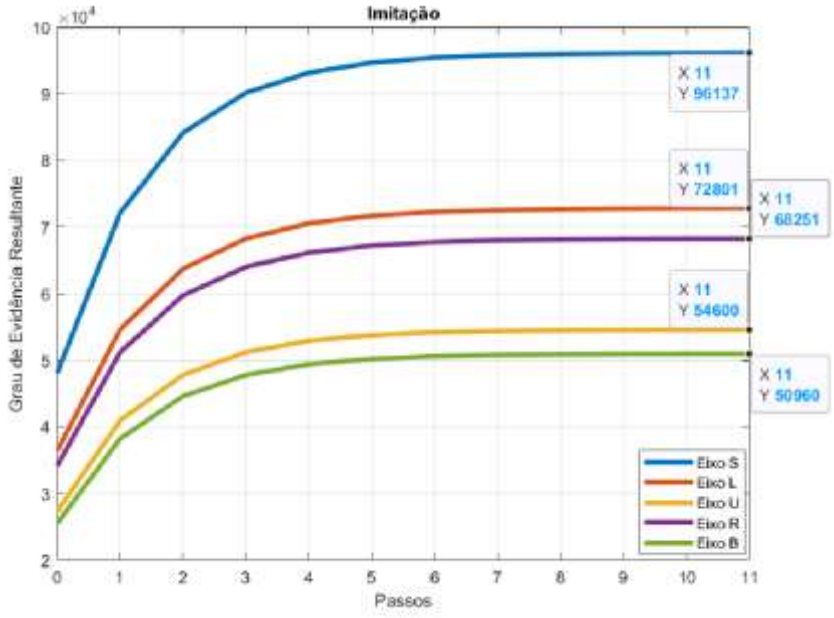


Figura 104 – SPApD-LPA2v - Imitação a partir da aprendizagem por demonstração efetuada com 11 passos (teste 2).

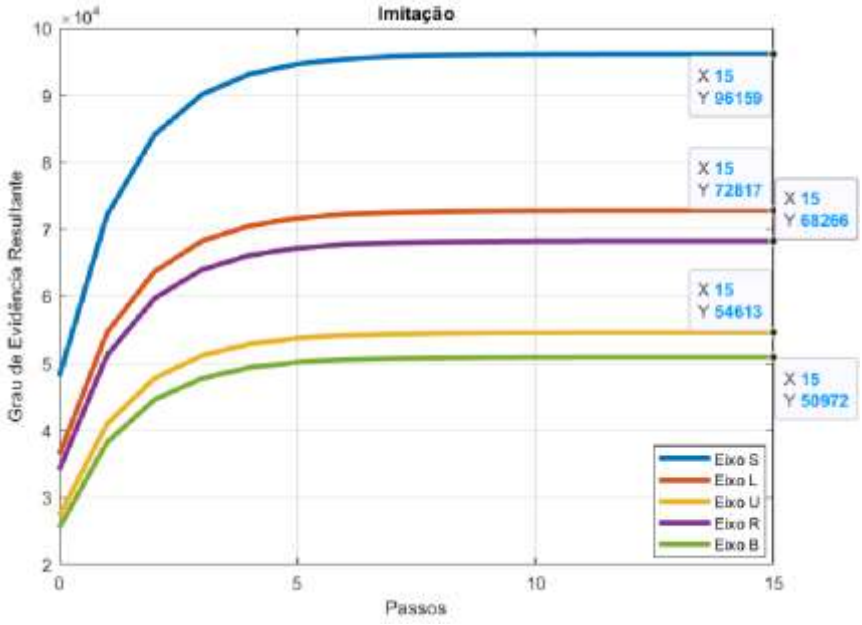


Figura 105 – SPApD-LPA2v - Imitação a partir da aprendizagem por demonstração efetuada com 15 passos (teste 3).

A seguir nas figuras 106, 107 e 108 são apresentados os resultados da etapa de imitação para 10, 11 e 15 passos respectivamente com os erros evidenciados no programa das CNAPaps e no *Pendant*.

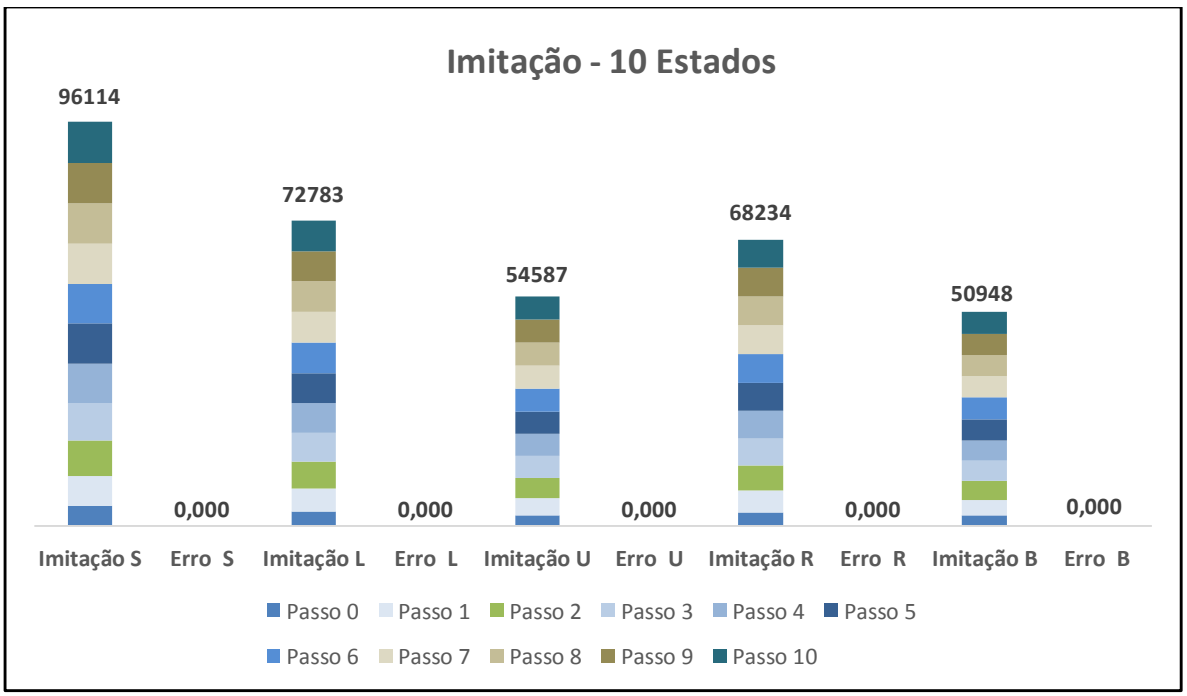


Figura 106 – SPApD-LPA2v - Imitação com erros apresentados - 10 estados (teste 1).

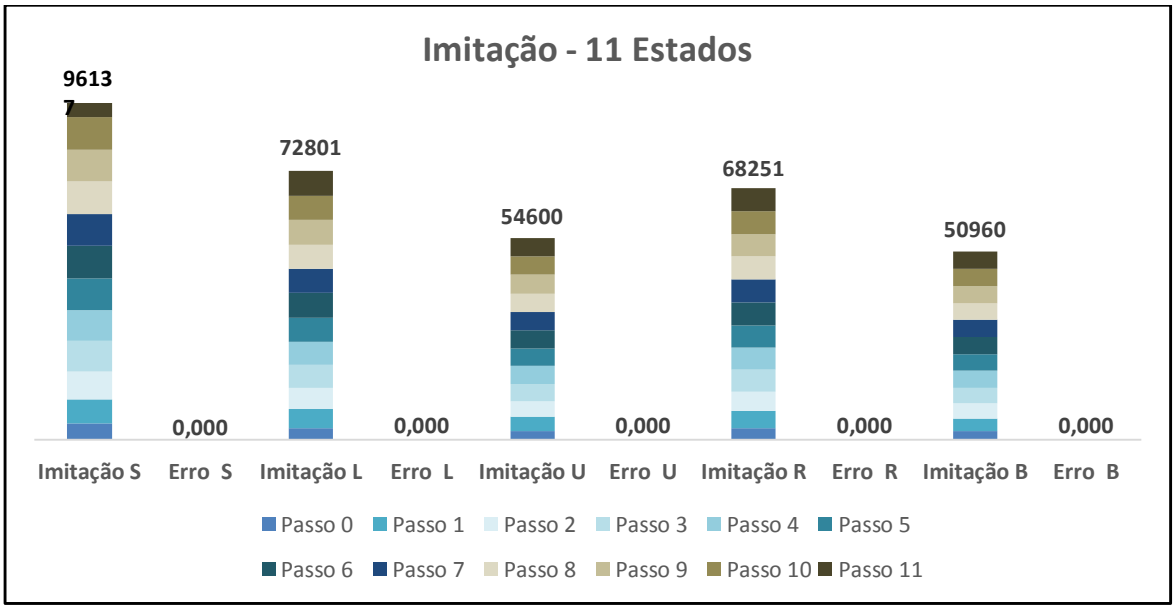


Figura 107 – SPApD-LPA2v - Imitação com erros apresentados - 11 estados (teste 2).

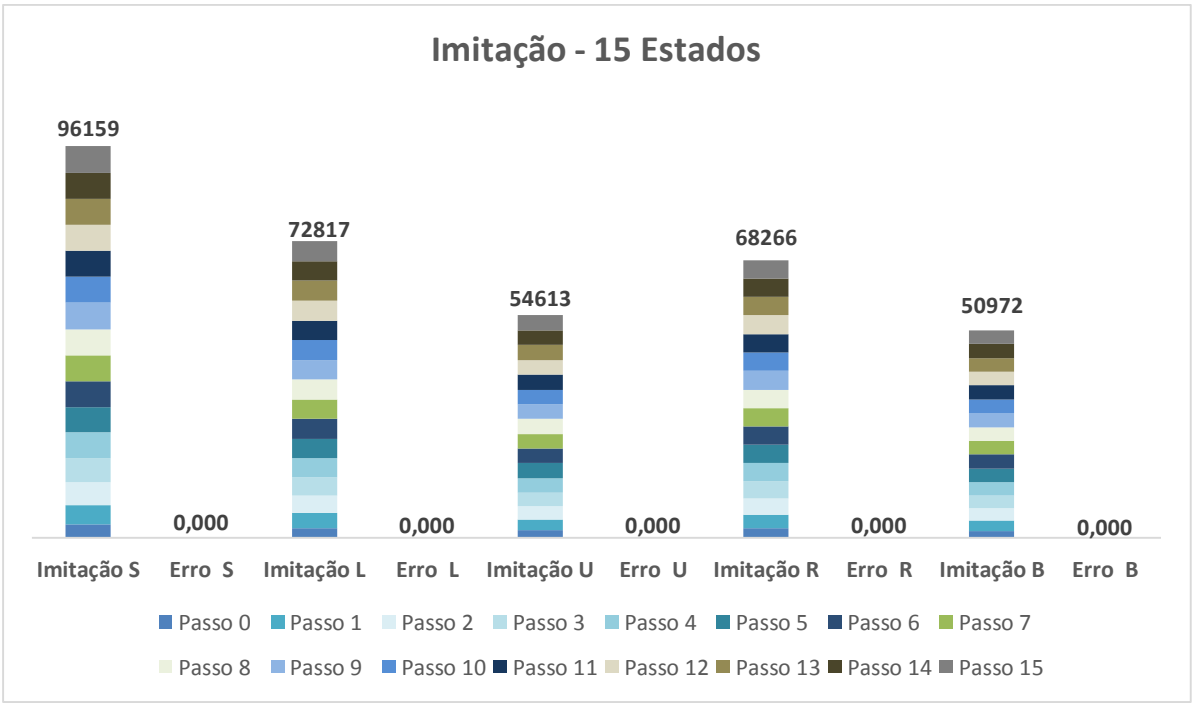


Figura 108 – SPApD-LPA2v - Imitação com erros apresentados - 15 estados (teste 3).

Conforme foi descrito no capítulo Materiais e Métodos, o processo de Teleoperação foi efetuado a partir da manipulação de dados utilizando como elo o Controlador Programável Schneider M262 e os protocolos de comunicação OPC UA e Ethernet enviando sinais de informação entre as CNAPs inseridas no MATLAB e o robô YASKAWA. Portanto, nessa pesquisa a teleoperação utilizada como processo

para ensinar os movimentos ao Robô Industrial por meio de demonstração, não foi realizada por ferramenta manual de operação (*Teach Pendant*).

Analisando os valores das figuras 103, 104 e 105 e das figuras 106, 107 e 108, verifica-se que, devido a Teleoperação ter sido efetuada via implantação de *softwares* dedicados especialmente para essa função, não houve diferenças significativas entre os pulsos obtidos no aprendizado e os mostrados nas respostas de posições na movimentação dos eixos do Robô Industrial na fase da Imitação. Portanto, os resultados indicam que a metodologia utilizada nessa pesquisa abrangendo o Sistema Paraconsistente de Aprendizagem por Demonstração - SPApD-LPA2v, se mostrou com baixa porcentagem de erro, tanto no estado final como nos estados intermediários que acontecem no processo de aprendizagem, o que demonstra alta eficiência nas fases de aprendizagem e imitação.

De maneira geral os resultados mostraram que os desempenhos dos *softwares* construídos facilitaram a inserção da teleoperação no aprendizado a partir da troca de informações entre as CNAPs e o robô, agilizando a programação e minimizando tempo de parada de produção. Sendo essa uma importante demanda dos diversos setores da indústria que utilizam robôs, os valores expostos nas tabelas confirmam que a pesquisa desenvolvida apresenta um alto grau de contribuição para solucionar o problema que envolve reprogramação de tarefas em robôs industriais.

4 CONCLUSÕES

Neste estudo, foi apresentada uma aplicação inovadora da Lógica Paraconsistente no método de Aprendizagem por Demonstração (ApD), explorando o potencial dessa abordagem em contextos complexos como robôs industriais. Ao longo das pesquisas desenvolvidas, foi observado que o uso da Lógica Paraconsistente, especialmente através do algoritmo da Célula Neural Artificial de aprendizagem (CNAPap), revelou-se promissor para aprimorar e ser um indutor do processo de ApD. Sendo uma lógica não clássica a Lógica Paraconsistente, capaz de lidar com informações contraditórias, demonstrou sua aplicabilidade ao tratar dados provenientes do aprendizado por demonstração. Os algoritmos construídos com base na LPA2v apresentaram resultados com bons níveis de eficácia, indicando a abertura de novos caminhos nesta área de conhecimento. A rede construída e utilizada neste estudo, tendo como base o algoritmo de uma CNAPap e fundamentado no Sistema Paraconsistente de Aprendizagem por Demonstração (SPApD-LPA2v), respondeu de maneira eficiente ao aprendizado de padrões normalizados, destacando-se pela capacidade de armazenar gradualmente informações através de técnicas de recorrência. Verificou-se que a implementação prática desse algoritmo no MATLAB, utilizando o Controlador Programável (CP) Schneider M262 como meio de comunicação entre a aplicação e o robô industrial YASKAWA modelo MOTOMAN - GP8, proporcionou modularidade e flexibilidade da aplicação seguindo os conceitos da IEC 61131-3. Os resultados mostraram também que os métodos utilizados para comunicação, OPC UA entre o programa CNAPap desenvolvido na plataforma MATLAB e o CP, bem como o protocolo ETHERNET entre o CP e o robô, foram adequados e poderão ser replicáveis em futuros estudos do SPApD-LPA2v. Os experimentos com o Robô industrial indicam que todos os objetivos foram alcançados e essas contribuições representam avanços significativos no desenvolvimento de sistemas neurais capazes de realizar um aprendizado por demonstração, em contextos como a movimentação de robôs industriais instalados nos mais diversos processos industriais. Ressalta-se, portanto, que com a utilização de um SPApD-LPA2v, será possível fomentar um ecossistema flexível de manufatura para os setores

industriais, uma vez que a robótica demandará o desenvolvimento de um fluxo de trabalho com uma programação altamente ágil, interligada e eficiente, necessária para atender às demandas em constante mutação dos diversos modelos de plantas fabris.

4.1 Trabalhos futuros

Este estudo destaca a viabilidade e potencial da aplicação da Lógica Paraconsistente Anotada no campo do método de Aprendizado por Demonstração. A rede neural, tendo como base o uso do algoritmo da Célula Neural Artificial Paraconsistente de aprendizagem (CNAPap), revelou-se promissora para base de desenvolvimentos futuros devido a sua modularidade e assim capaz de oferecer flexibilidade para construção de Redes Neurais Artificiais Paraconsistentes em diferentes configurações.

Considerando que os resultados obtidos na implementação em robôs industriais indicaram eficiência no aprendizado e na imitação de movimentação dos eixos de um robô, trabalhos com outros tipos de robôs utilizando esta técnica serão necessários para o avanço do ApD em sistemas mais complexos. Portanto, novas pesquisas podem ser desenvolvidas com o sistema SPApD-LPA2v, processos de reaprendizado em virtude de erros detectados na trajetória, além de poder explorar configurações adicionais de redes neurais paraconsistentes, visando aplicações mais amplas em sistemas dinâmicos e complexos na área da robótica industrial, fomentando um efetivo ganho de agilidade na programação, flexibilidade e produtividade nos diversos processos da manufatura.

REFERÊNCIAS

ABE, J. M. **Fundamentos da Lógica Anotada**, (Foundations of Annotated Logic) (in Portuguese), Ph. D. Thesis, Universidade de São Paulo, São Paulo, 1992.

ABE, J. M. **Paraconsistent Artificial Neural Networks: an Introduction, Lecture Notes In Artificial Intelligence 3214**, Springer, Eds. J.G. Carbonell & J. Siekmann, ISSN 0302-9743, ISBN 3-540-23206-0, pp. 942-948, 2004.

ARGALL B. D., CHERNOVA S., VELOSO M., BROWNING B. **A survey of robot learning from demonstration, Robotics and Autonomous Systems**, Volume 57, Issue 5, 2009, Pages 469-483, https://doi.org/10.1016/j.robot.2008.10.024.

BARBUY, H. S.; GOLDEMBERG, C. **Regulador De Tensão De Gerador**. Boletim Técnico Da Escola Politécnica Da Usp. Bt/Pea, São Paulo, V. 621, N.Bt/Pea, P. 3121-3133, ISSN 1413-2214, 2001.

BARBUY, H. S.; ROCCO, A. ; FERNANDES, L. A. P.; GOLDEMBERG, C.. **Rectifier Choices For Synchronous Generator Excitation**. Anais Do 7º Congresso Brasileiro De Eletrônica De Potência, Fortaleza/ Ceará/ Brasil, P. 160-166, 2003.

BOTTURA FILHO, J. A. **Um estudo de caso de organização de programas de automação industrial baseada na Norma IEC61131-3.** Monografia de Conclusão de Curso. PUC. São Paulo, 2007.

CARBOGIM, D. V. **Programação em Lógica Anotada: Teoria e Aplicações**. Dissertação de Mestrado. Instituto de Matemática e Estatística – USP. São Paulo, 1996.

DA COSTA, N. C. A, ABE, J.M. e SUBRAHMANIAN V. S. **Remarks on Annotated Logic**. Instituto de Estudos Avançados da USP, 2000.

DA COSTA, N. C. A; HENSCHEN, L. J.; LU, J. J. et al. **Automatic Theorem Proving in Paraconsistent Logics: Teory and Implementation**. Instituto de Estudos Avançados da USP - 1990.

DA COSTA, N. C. A & SUBRAHMANIAN, V. S.. **Paraconsistent Logics as a Formalism for Reasoning About Inconsistent Knowledge Bases**. Instituto de Estudos Avançados da USP - 1989.

DA COSTA, N. C. A; SUBRAHMANIAN, V. S. e VAGO, C. **The Paraconsistent Logics Pτ**. Instituto de Estudos Avançados da USP - 1989.

DA SILVA FILHO, J. I. **Métodos de Aplicações da Lógica Paraconsistente Anotada de anotação com dois valores-LPA2v com construção de Algoritmo e Implementação de Circuitos Eletrônicos.** Tese (Doutorado) – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais. São Paulo, 1999.

DA SILVA FILHO, J. I. & ABE, J. M., **Fundamentos das Redes Neurais Paraconsistentes – Destacando Aplicações em Neurocomputação,** (in Portuguese) Editora Arte & Ciência, ISBN 85-7473-045-9, 247 pp., 2001.

DA SILVA FILHO, **Conheça O Robo Emmy – O Primeiro Robô Que Funciona com a Lógica Paraconsistente**. Disponível em: http;//paralogike.com.br/roboemmy.htm. Acesso em 02 de fevereiro de 2024.

DA SILVA FILHO, J. I. **"Treatment of Uncertainties with Algorithms of the Paraconsistent Annotated Logic", Journal of Intelligent Learning Systems and Applications**, vol. 4, pp. 144-153, May, 2012.

DA SILVA FILHO, J. I. ABE, J. M.; TORRES, G. L. **Inteligência Artificial com as Redes de Análises Paraconsistentes,** Editora LTC, 1ª. Ed., Rio de Janeiro, pp.4-5, pp.40-84, pp.101-185. 2008,

DA SILVA FILHO, J. I. **"Métodos de Aplicações da Logica Paraconsistente Anotada de anotação com dois valores-LPA2v"**. Seleção Documental, N.1 Ano1, 3/2006.

DA SILVA FILHO, J. I. et al. **"Support at Decision in Electrical Systems of subtransmission through selection of Topologies by a Paraconsistent Simulator"**. Revista IEEE América Latina, v. 14, p. 1993-1999, 2016.

DA SILVA FILHO, J. I et al. **"Paraconsistent Artificial Neural Network for structuring Statistical Process Control in Electrical Engineering"**, Towards Paraconsistent Engineering Book, Springer International Publishing, pp. 77-102, 2016.

Da Silva Filho, J.I.; Abe, J.M.; Marreiro, A.d.L.; Martinez, A.A.G.; Torres, C.R.; Rocco, A.; Côrtes, H.M.; Mario, M.C.; Pacheco, M.T.T.; Garcia, D.V.; Blos, M.F. **Paraconsistent annotated logic algorithms applied in management and control of communication network routes Sensors**, 2021, 21(12), 4219 https://doi.org/10.3390/s21124219

DE CARVALHO JR, A. et al. **"A Study of Paraconsistent Artificial Neural Cell of Learning Applied as PAL2v Filter**". Revista IEEE América Latina, v. 16, p. 202-208, 2018.

DE CARVALHO, F. R.; BRUNSTEIN, Israel; ABE, J. M. **Um Estudo de Tomada de Decisão Baseado em Lógica Paraconsistente Anotada: Avaliação do Projeto de uma Fábrica.** Revista Pesquisa e Desenvolvimento Engenharia de Produção n.1, p. 47-62, dez. 2003.

FALQUETE, V. L. M. **Utilização de Lógica Paraconsistente para Tratamento de Inconsistências em Sistemas de Raciocínio Baseado em Casos**. Dissertação de Mestrado. Pontifícia Universidade Católica do Paraná, Curitiba, 2004.

FERNANDES, C.L.M., MARIO C. M., DA SILVA FILHO, J. I. **"Study for inclusion of Paraconsistente Annotaded logic in specific Standards for use in Programable Controllers",** Science and Technology, ISSN: 2317-1316, Vol 1 2, p.p 49-53, 2012.

Garcia DV, Da Silva Filho JI, Silveira Jr. L, Pacheco MTT, Abe JM, et al. **Analysis of Raman spectroscopy data with algorithms based on paraconsistent logic for characterization of skin cancer lesions**. Vibrational Spectroscopy 2019;103;102929.

GOMIDE, F. C. e GUDWIN, R. R. **Modelagem, Controle, Sistemas e Lógica Fuzzy**. Departamento de Engenharia de Computação e Automação Industrial (DCA).Faculdade de Engenharia Elétrica (FEE).Universidade Estadual de Campinas (UNICAMP). SBA Controle & Automação/Vol.4 n°3/setembro-outubro, pp. 97-115, 1994.

GUIMARÃES H. C. F., **Norma IEC 61131-3 para Programação de Controladores Programáveis: Estudo e Aplicação.** Monografia de conclusão de curso Universidade Federal do Espírito Santo, 2005.

HASEGAWA, F. M. **Uma Abordagem Baseada em Lógica Paraconsistente para Avaliação de Ofertas em Negociações entre Organizações Artificiais**. Dissertação de Mestrado da Pontifícia Universidade Católica do Paraná, Curitiba, 2004.

HAYKIN, S. **"Neural Networks and Learning Machines**". Editora Prentice Hall; 3rd ed. edição (1 junho 2008).

HUGHES, T. A., **Programable Controllers. - Resources Formeasurements and Control Series - ISA.** North Caroline, InstrumentSociety of America., 1989, 252p.

INTERNATIONAL ELECTROTECHNICAL COMISSION – **IEC. IEC61131-3 Programmable Controllers - Part 3. Programming Languages,** IEC, 1993.

JOHN, K. H., TIEGELKAMP M. **"IEC61131-3: Programing Industrial Automation Systems". Concepts and Prohraming Languages, Requiriments for Programing Systems, Decision** – Making Aids. 2ed. New York, Springer, 390p., 2010.

LEMES NETO, M. C. e VENSON N. **Lógica Paraconsistente**. Departamento de Informática e Estatística – INE, Universidade Federal de Santa Catarina (UFSC), Brasil, 2002.

LIMA JR L. H. C. de. **Lógica Paraconsistente Anotada: Aplicação na Aquisição de Dados de Sensores de Temperatura.** Dissertação de mestrado em Engenharia Elétrica. Universidade Federal de Campina Grande. 2003.

Liu, T.; Lemeire, J. **Efficient and Effective Learning of HMMs Based on Identification of Hidden States**. Mathematical Problems in Engineering, vol. 2017, Article ID 7318940, 26 pages, 2017. https://doi.org/10.1155/2017/7318940

MACIEL, A. C.; FRANCISCO, B. S. U.; FARIAS, I, S. et al, **Aplicação Em Robótica Do Algoritmo Para- Analisador Utilizando A Tecnologia Lego Mindstorms Nxt.** Seleção Documental - GLPA N.21 Ano 6 Ed. Paralogike - Santos - SP Brasil, pp. 5-10, 2011.

MÁRIO, M. C. **PROPOSTA DE APLICAÇÃO DAS REDES NEURAIS ARTIFICIAIS PARACONSISTENTES COMO CLASSIFICADOR DE SINAIS UTILIZANDO APROXIMAÇÃO FUNCIONAL. DISSERTAÇÃO (MESTRADO) -** Universidade Federal de Uberlândia**.** 129 P. Uberlândia- MG, 2003.

MARIO, M. C., FERRARA, L. F. P., e DA SILVA FILHO, J. I, **"Treinamento de uma Célula Neural Artificial Paraconsistente de Aprendizagem"**. Revista Seleção Documental, no. 6, pp. 11-16, Ed. Paralogike, São Paulo, Brazil, 2007.

Mario, M. C., Garcia, D. V., Da Silva Filho, J. I., Silveira Júnior, L., Barbuy, H. S. **Characterization and classification of numerical data patterns using Annotated Paraconsistent Logic and the effect of contradiction. Research**, Society and Development, [S. l.], v. 10, n. 13, p. e283101320830, DOI: 10.33448/rsd-v10i13.20830.

MIYAGI, P. E. **Controle Programável**. Editora Edgard Blücher Ltda. – 1996.

MOLLENKAMP, R. A. **Controle Automático de Processos**. EBRAS Editora Brasileira – SMAR. 1988.

NIEKUM, S; OSENTOSKI, S; KONIDARIS, G.; CHITTA, S.; MARTHI, B.; BARTO, A. G. **Learning grounded finite-state representations from unstructured demonstrations**. The International Journal of Robotics Research. 2015; 34(2): 131-157. doi:10.1177/0278364914554471

OGATA, K. **Engenharia de Controle Moderno**, 2ª ed. – Prentice Hall, 1990.

OLIVEIRA, M.; SEIXAS, C.; BOTTURA FILHO, J., **Aplicando a norma IEC 61131 na Automação de Processos**, 2007.

Pastor, P.; Kalakrishnan, M.; Meier, F.; Stulp, F.; Buchli, J.; Theodorou, E.; Schaal, S. (2013). **From dynamic movement primitives to associative skill memories**. Robotics and Autonomous Systems, 2013, vol. 61, no. 4, pp. 351–361

RIBEIRO, A. M. **Instrumentação**, 8ª ed. Tek Treinamento & Consultoria Ltda, 1999.

ROSÁRIO, João Mauricio. **Automação Industrial**. São Paulo: Editora: Baraúna. 2009.

SCHAAL, S. **Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. in Adaptive Motion of Animals and Machines**. Springer, 2006, pp. 261–280.

SIGHIERI, L. & NISHINARI, A. **Controle Automático de Processos Industriais – Instrumentação**. Editora EDGARD BLÜCHER LTDA. 2ª Edição – 1988.

SUCOLOTTI, A. A.; CENTENARO, A. C. e DRUZIANI, C. F. M. **Recuperação de Informação em Bases Textuais: Uma Abordagem Paraconsistente**. Universidade Paranaense (UNIPAR), 2007.

THOMAS D. McGee. **Principles and Methods of Temperature Measurement** – Wiley-Interscience Publication, 1988.

TORRES et al, **Otimização de Estratégias de Controle em Sistemas Multivariáveis, Congresso Internacional de Automação, Sistemas e Instrumentação – ISA Show Brasil**, São Paulo, outubro, 2001.

TORRES, C. R., **Sistema Inteligente Baseado na Lógica Paraconsistente Anotada Evidencial Et para Controle e Navegação de Robôs Móveis Autônomos em um Ambiente não Estruturado,** Tese de doutorado em Ciências em Energia Elétrica – Universidade Federal de Itajubá. 193 p. Itajubá- MG, agosto de 2010.

APÊNDICE A – Mapa de Memória – CNAPap sentido Robô MOTOMAN GP8

S	CNAPap		Robot_S			
	PLC	OPC UA	dwDwordRobot_S			
		Blocks	DWORD_AS_BIT			
			BIT_AS_WORD2		BIT_AS_WORD1	
		Ethernet	%QW2 (wWordRobot_S2)		%QW1 (wWordRobot_S1)	
	Robot	Input General	IG#(3)	IG#(4)	IG#(5)	IG#(6)
		Program Variable	B003	B002	B001	B000
			D000			
		Positon Variable	P002(1)			

L	CNAPap		Robot_L			
	PLC	OPC UA	dwDwordRobot_L			
		Blocks	DWORD_AS_BIT			
			BIT_AS_WORD2		BIT_AS_WORD1	
		Ethernet	%QW4 (wWordRobot_L2)		%QW3 (wWordRobot_L1)	
	Robot	Input General	IG#(7)	IG#(8)	IG#(9)	IG#(10)
		Program Variable	B007	B006	B005	B004
			D001			
		Positon	P002(2)			

U	CNAPap		Robot_U			
	PLC	OPC UA	dwDwordRobot_U			
		Blocks	DWORD_AS_BIT			
			BIT_AS_WORD2		BIT_AS_WORD1	
		Ethernet	%QW6 (wWordRobot_U2)		%QW5 (wWordRobot_U1)	
	Robot	Input General	IG#(11)	IG#(12)	IG#(13)	IG#(14)
		Program Variable	B011	B010	B009	B008
			D002			
		Positon Variable	P002(3)			

R	CNAPap		Robot_R			
	PLC	OPC UA	dwDwordRobot_R			
		Blocks	DWORD_AS_BIT			
			BIT_AS_WORD2		BIT_AS_WORD1	
		Ethernet	%QW8 (wWordRobot_R2)		%QW7 (wWordRobot_R1)	
	Robot	Input General	IG#(15)	IG#(16)	IG#(17)	IG#(18)
		Program Variable	B015	B014	B013	B012
			D003			
		Positon Variable	P002(4)			

B	CNAPap		Robot_B			
	PLC	OPC UA	dwDwordRobot_B			
		Blocks	DWORD_AS_BIT			
			BIT_AS_WORD2		BIT_AS_WORD1	
		Ethernet	%QW10 (wWordRobot_B2)		%QW9 (wWordRobot_B1)	
	Robot	Input General	IG#(19)	IG#(20)	IG#(21)	IG#(22)
		Program Variable	B019	B018	B017	B016
			D004			
		Positon Variable	P002(5)			

T	CNAPap		Robot_T			
	PLC	OPC UA	dwDwordRobot_T			
		Blocks	DWORD_AS_BIT			
			BIT_AS_WORD2		BIT_AS_WORD1	
		Ethernet	%QW12 (wWordRobot_T2)		%QW11 (wWordRobot_T1)	
	Robot	Input General	IG#(23)	IG#(24)	IG#(25)	IG#(26)
		Program Variable	B023	B022	B021	B020
			D005			
		Positon Variable	P002(6)			

APÊNDICE B – Mapa de Memória – Robô MOTOMAN GP8 sentido CNAPap

S	CNAPap		Robot_S_in			
	PLC	OPC UA	dwDwordRobot_S_send			
		Blocks	BIT_AS_DWORD			
			WORD_AS_BIT2		WORD_AS_BIT1	
		Ethernet	%IW2 (wWordRobot_S2_send)		%IW1 (wWordRobot_S1_send)	
	Robot	Output General	OG#(3)	OG#(4)	OG#(5)	OG#(6)
		Program Variable	B027	B026	B025	B024
			D011			
		Positon Variable	P000(1)			

L	CNAPap		Robot_L_in			
	PLC	OPC UA	dwDwordRobot_L_send			
		Blocks	BIT_AS_DWORD			
			WORD_AS_BIT2		WORD_AS_BIT1	
		Ethernet	%IW2 (wWordRobot_L2_send)		%IW1 (wWordRobot_L1_send)	
	Robot	Output General	OG#(7)	OG#(8)	OG#(9)	OG#(10)
		Program Variable	B031	B030	B029	B028
			D012			
		Positon Variable	P000(2)			

U	CNAPap		Robot_U_in			
	PLC	OPC UA	dwDwordRobot_U_send			
		Blocks	BIT_AS_DWORD			
			WORD_AS_BIT2		WORD_AS_BIT1	
		Ethernet	%IW2 (wWordRobot_U2_send)		%IW1 (wWordRobot_U1_send)	
	Robot	Output General	OG#(11)	OG#(12)	OG#(13)	OG#(14)
		Program Variable	B035	B034	B033	B032
			D013			
		Positon Variable	P000(3)			

R	CNAPar		Robot_R_in			
	PLC	OPC UA	dwDwordRobot_R_send			
		Blocks	BIT_AS_DWORD			
			WORD_AS_BIT2		WORD_AS_BIT1	
		Ethernet	%IW2 (wWordRobot_R2_send)		%IW1 (wWordRobot_R1_send)	
	Robot	Output General	OG#(15)	OG#(16)	OG#(17)	OG#(18)
		Program Variable	B039	B038	B037	B036
			D014			
		Positon Variable	P000(4)			

B	CNAPar		Robot_B_in			
	PLC	OPC UA	dwDwordRobot_B_send			
		Blocks	BIT_AS_DWORD			
			WORD_AS_BIT2		WORD_AS_BIT1	
		Ethernet	%IW2 (wWordRobot_B2_send)		%IW1 (wWordRobot_B1_send)	
	Robot	Output General	OG#(19)	OG#(20)	OG#(21)	OG#(22)
		Program Variable	B043	B042	B041	B040
			D015			
		Positon Variable	P000(5)			

T	CNAPar		Robot_T_in			
	PLC	OPC UA	dwDwordRobot_T_send			
		Blocks	BIT_AS_DWORD			
			WORD_AS_BIT2		WORD_AS_BIT1	
		Ethernet	%IW2 (wWordRobot_T2_send)		%IW1 (wWordRobot_T1_send)	
	Robot	Output General	OG#(23)	OG#(24)	OG#(25)	OG#(26)
		Program	B047	B046	B045	B044
			D016			
		Positon Variable	P000(6)			

APÊNDICE C – Programa Principal Robô MOTOMAN GP8

```
//JOB
//NAME CNAP
//POS
///NPOS 1,0,0,1,0,0
///TOOL 0
///POSTYPE PULSE
///PULSE
C00000=16466,-1046,-5828,2458,-57834,-64761
P00002=0,72818,0,0,0,0
//INST
///DATE 2023/12/09 10:50
///ATTR SC, RW
///GROUP1 RB1
NOP
*LABEL
// MOVJ DEFINE O MOVIMENTO DO EIXO DO ROBÔ EM 5 RAD/S para a posição inicial do robô
MOVJ C00000 VJ=5.00
//AGUARDANDO O ENVIO DO CP DE VALOR DIFERENTE ≠ 0, ENQUANTO 0 => NOP
WHILEEXP IG#(10)=0 (input general–1 byte, 8 bits)
//SEM OPERAÇÃO
        NOP
ENDWHILE
//P002 É A POSIÇÃO DESEJADO PARA O APRENDIZADO, NA VELOCIDADE 5 RAD/S
MOVJ P002 VJ=5.00
// CHAMA O PROGRAMA ENVIA_POSICAO_CLP_5_EIXOS
CALL JOB:ENVIA_POSICAO_CLP_5_EIXOS
JUMP *LABEL
END
```

APÊNDICE D - Função - Envia Posição 5 Eixos - Robô MOTOMAN GP8

```
/JOB
//NAME ENVIA_POSICAO_CLP_5_EIXOS_ROBO
//POS
///NPOS 0,0,0,1,0,0
///TOOL 0
///POSTYPE PULSE
///PULSE
P00000=96160,72818,54613,68267,50972,0
//INST
///DATE 2023/12/09 16:20
///ATTR SC,RW
///GROUP1 RB1
NOP
// $PX000 - CAPTURA A POSIÇÃO ATUAL DO ROBÔ EM PULSOS
// SE EM $PX000 O ÚLTIMO DÍGITO É “0” O ROBÔ ATUA EM “POSIÇÃO” SE 1 DESLOCAMENTO
// INSERE A POSIÇÃO DO ROBÔ EM PX000
GETS PX000 $PX000
'ENVIA EIXO S
GETE D010 P000 (1)
// ELIMINA A POSSIVEL PARTE NEGATIVA SOMANDO O VALOR MÁXIMO DO EIXO (ZERO VIVO)
SET D011  EXPRESS D010 + 153600
//ALOCA EM B027 O BYTE MAIS SIGNIFICATIVO DA CONVERSÃO DA VARIÁVEL D011
SET B027 EXPRESS D011 / 16777216
```

B027								B026								B025								B024							
0	0	0	0	0	0	0	0																								

```
//ALOCA NA B026 O SEGUNDO BYTE MAIS SIGNIFICATIVO DA CONVERSÃO DA VARIÁVEL D011
SET B026 EXPRESS (D011 - B027 * 16777216 ) / 65536
```

B027								B026								B025								B024							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1																

```
//ALOCA NA B025 O TERCEIRO BYTE MAIS SIGNIFICATIVO DA CONVERSÃO DA VARIÁVEL D011
SET B025  EXPRESS ( D011 - B027 * 16777216 - B026 * 65536 ) / 256
```

B027								B026								B025								B024							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1								

```
//ALOCA NA B024 O BYTE MENOS SIGNIFICATIVO DA CONVERSÃO DA VARIÁVEL D011
SET B024  EXPRESS ( D011 - B027 * 16777216 - B026 * 65536 - B025 * 256 ) / 1
```

B027								B026								B025								B024							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0

```
'ENVIA EIXO L
GETE D010 P000 (2)
SET D012  EXPRESS D010 + 72818
SET B031  EXPRESS D012 / 16777216
SET B030  EXPRESS ( D012 - B031 * 16777216 ) / 65536
SET B029  EXPRESS ( D012 - B031 * 16777216 - B030 * 65536 ) / 256
SET B028  EXPRESS ( D012 - B031 * 16777216 - B030 * 65536 - B029 * 256 ) / 1
'ENVIA EIXO U
GETE D010 P000 (3)
SET D013  EXPRESS D010 + 54614
SET B035  EXPRESS D013 / 16777216
SET B034  EXPRESS ( D013 - B035 * 16777216 ) / 65536
SET B033  EXPRESS ( D013 - B035 * 16777216 - B034 * 65536 ) / 256
SET B032  EXPRESS ( D013 - B035 * 16777216 - B034 * 65536 - B033 * 256 ) / 1
'ENVIA EIXO R
GETE D010 P000 (4)
SET D014  EXPRESS D010 + 153600
SET B039  EXPRESS D014 / 16777216
SET B038  EXPRESS ( D014 - B039 * 16777216 ) / 65536
SET B037  EXPRESS ( D014 - B039 * 16777216 - B038 * 65536 ) / 256
SET B036  EXPRESS ( D014 - B039 * 16777216 - B038 * 65536 - B037 * 256 ) / 1
'ENVIA EIXO B
GETE D010 P000 (5)
SET D015  EXPRESS D010 + 87381
SET B043  EXPRESS D015 / 16777216
SET B042  EXPRESS ( D015 - B043 * 16777216 ) / 65536
SET B041  EXPRESS ( D015 - B043 * 16777216 - B042 * 65536 ) / 256
SET B040  EXPRESS ( D015 - B043 * 16777216 - B042 * 65536 - B041 * 256 ) / 1
'C0NFIRMA
//BYTES B047, B046, B045 E B044 PERTENCE AOS BYTES DO EIXO NÃO UTILIZADO O BYTE 044 //ENVIA
A CONFIRMAÇÃO DE FINALIZAÇÃO
SET B047 0
SET B046 0
SET B045 0
SET B044 1
'ENVIA VIA REDE ETHERNET
//ALOCA VARIÁVEIS INTERNAS DO ROBÔ EM VARIÁVEIS PARA COMUNICAÇÃO EXTERNA
SET OG#(3)  EXPRESS B027
SET OG#(4)  EXPRESS B026
SET OG#(5)  EXPRESS B025
SET OG#(6)  EXPRESS B024
SET OG#(7)  EXPRESS B031
SET OG#(8)  EXPRESS B030
SET OG#(9)  EXPRESS B029
SET OG#(10)  EXPRESS B028
SET OG#(11)  EXPRESS B035
SET OG#(12)  EXPRESS B034
SET OG#(13)  EXPRESS B033
SET OG#(14)  EXPRESS B032
SET OG#(15)  EXPRESS B039
```

```
SET OG#(16)  EXPRESS B038
SET OG#(17)  EXPRESS B037
SET OG#(18)  EXPRESS B036
SET OG#(23)  EXPRESS B047
SET OG#(24)  EXPRESS B046
SET OG#(25)  EXPRESS B045
SET OG#(26)  EXPRESS B044
SET OG#(19)  EXPRESS B043
SET OG#(20)  EXPRESS B042
SET OG#(21)  EXPRESS B041
SET OG#(22)  EXPRESS B040
// AGUARDA 1 SEGUNDO
TIMER T=1.000
//ZERA A VARIÁVEL DE CONFIRMAÇÃO
SET B044 0
//REENVIA A VARIÁVEL DE CONFIRMAÇÃO INFORMANDO O TÉRMINO DO ENVIO
SET OG#(26)  EXPRESS B044
//RETORNA PARA O PROGRAMA CNAP
RET
END
```

APÊNDICE E – Programa completo das CNAPap

```
clc
clear all
close all

%--------------------------------------------------------------------------
%Para cada alteração de set point...
%MATLAB
%Atualizar Set_point
%Atualizar Valor_pulso_minimo
%Atualizar nome do gráfico

%ROBÔ
%Zerar variáveis de posição P002
%Atualizar eixo da variável de posição P002

%ENVIA_POSICAO_CLP
%Atualizar linha 3 P000 (x)
%Atualizar linha 4 (Valor_pulso_minimo)

%--------------------------------------------------------------------------
%                   Estabelece comunicação OPC UA
%--------------------------------------------------------------------------
Comunicacao = opcua('opc.tcp://10.10.0.243:4840') %Inicia comunicação (IP + Porta)
connect(Comunicacao)                        %Conecta com o CLP
Comunicacao                                 %Mostra status a comunicação

Variaveis_CLP_wr = browseNamespace(Comunicacao)    %Procura variáveis OPC UA do CLP (escrita)
Variaveis_CLP_rd = browseNamespace(Comunicacao)    %Procura variáveis OPC UA do CLP (leitura)
%--------------------------------------------------------------------------
% Valores pré carregados de escrita
%--------------------------------------------------------------------------
Robot_S_wr = 0;
Robot_L_wr = 0;
Robot_U_wr = 0;
Robot_R_wr = 0;
Robot_B_wr = 0;
Confirma_wr = 0;
%--------------------------------------------------------------------------
% Valores Inseridos
%--------------------------------------------------------------------------
Set_point_S = 96160;
Valor_pulso_minimo_S = 153600;

Set_point_L = 72818;
Valor_pulso_minimo_L = 72818;

Set_point_U = 54613;
```

```
Valor_pulso_minimo_U = 54614;

Set_point_R = 68267;
Valor_pulso_minimo_R = 153600;

Set_point_B = 50972;
Valor_pulso_minimo_B = 87381;

%--------------------------------------------------------------------------
passo_atual = 0  %Valor inicial da contagem de passos
passo = 0:1:10;
FA = 1;    %Fator de aprendizagem

%--------------------------------------------------------------------------
u1_S = Set_point_S;        %Grau de certeza inicial inserido
ui_S = (Set_point_S/2)+1;   %Grau de certeza inicial

u1_L = Set_point_L;        %Grau de certeza inicial inserido
ui_L = (Set_point_L/2)+1;   %Grau de certeza inicial

u1_U = Set_point_U;        %Grau de certeza inicial inserido
ui_U = (Set_point_U/2)+1;     %Grau de certeza inicial

u1_R = Set_point_R;        %Grau de certeza inicial inserido
ui_R = (Set_point_R/2)+1;   %Grau de certeza inicial

u1_B = Set_point_B;        %Grau de certeza inicial inserido
ui_B = (Set_point_B/2)+1;     %Grau de certeza inicial

%--------------------------------------------------------------------------
uEi_S = (ui_S-(Set_point_S/2)+Set_point_S)/2;   %Valor inicial de evidência
uE_S = uEi_S;                          %Valor de evidência
uE_vec_S(1,1) = uE_S;                       %Valor de evidência (vetor)

uEi_L = (ui_L-(Set_point_L/2)+Set_point_L)/2;   %Valor inicial de evidência
uE_L = uEi_L;                          %Valor de evidência
uE_vec_L(1,1) = uE_L;                       %Valor de evidência (vetor)

uEi_U = (ui_U-(Set_point_U/2)+Set_point_U)/2;   %Valor inicial de evidência
uE_U = uEi_U;                           %Valor de evidência
uE_vec_U(1,1) = uE_U;                        %Valor de evidência (vetor)

uEi_R = (ui_R-(Set_point_R/2)+Set_point_R)/2;   %Valor inicial de evidência
uE_R = uEi_R;                          %Valor de evidência
uE_vec_R(1,1) = uE_R;                       %Valor de evidência (vetor)

uEi_B = (ui_B-(Set_point_B/2)+Set_point_B)/2;   %Valor inicial de evidência
uE_B = uEi_B;                          %Valor de evidência
uE_vec_B(1,1) = uE_B;                       %Valor de evidência (vetor)
```

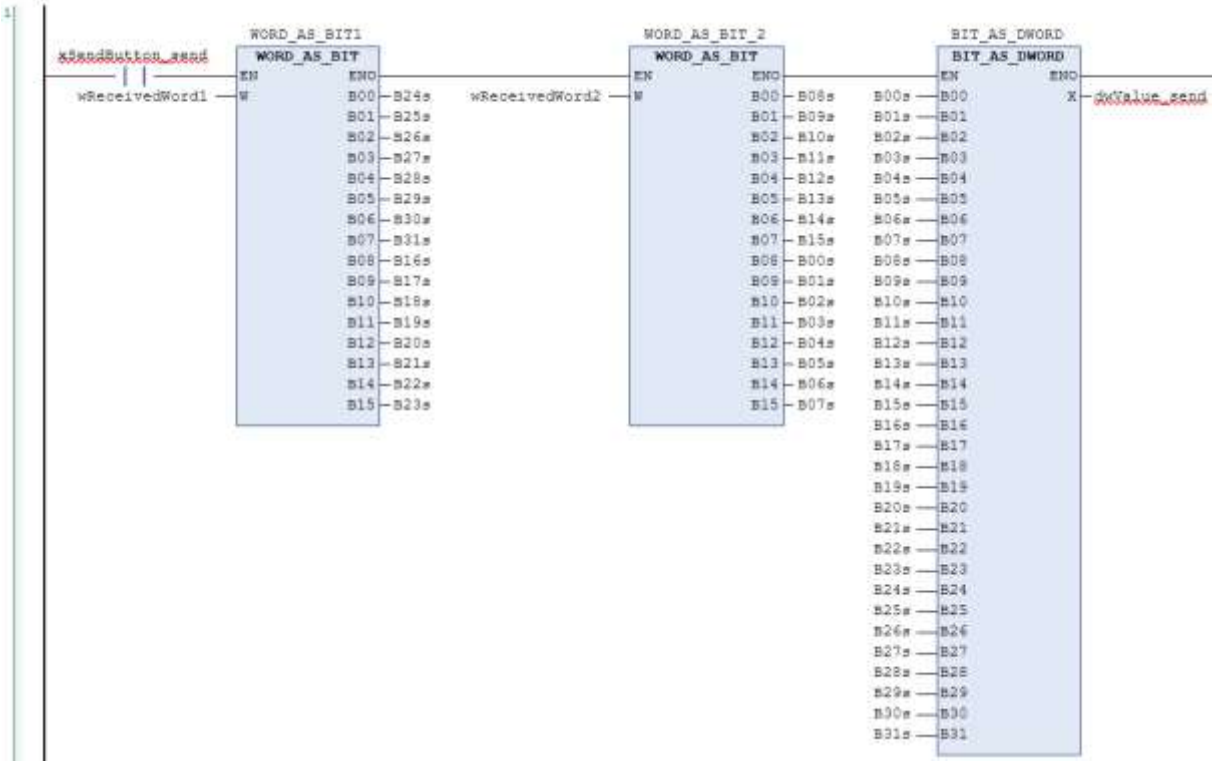
```
%--------------------------------------------------------------------------
% Treino das Células (11 passos)
%--------------------------------------------------------------------------
 while(passo_atual<=9)

   Robot_S_wr = 0;
   Robot_L_wr = 0;
   Robot_U_wr = 0;
   Robot_R_wr = 0;
   Robot_B_wr = 0;
   Confirma_wr = 65535;
   //Valores de wr são enviados via OPCUA para o CP
   Valores_wr = {Robot_B_wr, Robot_L_wr,...%Carrega valores para escrita
            Robot_R_wr, Robot_S_wr,...%
            Confirma_wr, Robot_U_wr}; %
   //Envia o combo wr via OPC UA
   writeValue(Comunicacao,Variaveis_CLP_wr,Valores_wr) %Envia valores para o CLP

%--------------------------------------------------------------------------
% Aguarda receber comando do Robô
%--------------------------------------------------------------------------
  //Lê as variaveis OPC UA do CP e converte para matriz de 5x1 (5 eixos)
  Valores_rd = cell2mat(readValue(Comunicacao,Variaveis_CLP_rd));
  // Converte de matriz em número inteiro de 32 bits (4 bytes)
  Confirma_rd = int32(Valores_rd(5,1));
  //Enquanto o CP trocando dados com o robô Confirma_rd == 0,
  while(Confirma_rd == 0)
      Valores_rd = cell2mat(readValue(Comunicacao,Variaveis_CLP_rd));
      Confirma_rd = int32(Valores_rd(5,1));
      pause(1)
  end
%--------------------------------------------------------------------------
% Ciclo de Treino
%--------------------------------------------------------------------------
    //Retira o valor incrementado como zero vivo no robô e armazena nas variaveis Robot_variaveis_rd
    Robot_S_rd = int32(Valores_rd(4,1)) - Valor_pulso_minimo_S;
    Robot_L_rd = int32(Valores_rd(2,1)) - Valor_pulso_minimo_L;
    Robot_U_rd = int32(Valores_rd(6,1)) - Valor_pulso_minimo_U;
    Robot_R_rd = int32(Valores_rd(3,1)) - Valor_pulso_minimo_R;
    Robot_B_rd = int32(Valores_rd(1,1)) - Valor_pulso_minimo_B;

    //Retorna ao algorítmo da CNAPap
    u1_S = Robot_S_rd;
    u2_S = uE_S;
    u2c_S = Set_point_S - u2_S;
    uE_S = ((u1_S - (u2c_S*FA)) + Set_point_S)/2;

    u1_L = Robot_L_rd;
    u2_L = uE_L;
    u2c_L = Set_point_L - u2_L;
```

```
    uE_L = ((u1_L - (u2c_L*FA)) + Set_point_L)/2;

    u1_U = Robot_U_rd;
    u2_U = uE_U;
    u2c_U = Set_point_U - u2_U;
    uE_U = ((u1_U - (u2c_U*FA)) + Set_point_U)/2;

    u1_R = Robot_R_rd;
    u2_R = uE_R;
    u2c_R = Set_point_R - u2_R;
    uE_R = ((u1_R - (u2c_R*FA)) + Set_point_R)/2;

    u1_B = Robot_B_rd;
    u2_B = uE_B;
    u2c_B = Set_point_B - u2_B;
    uE_B = ((u1_B - (u2c_B*FA)) + Set_point_B)/2;
%-------------------------------------------------------------------------
    passo_atual = passo_atual+1
    passo_vec(1, passo_atual) = passo_atual;

    uE_vec_S(1,passo_atual+1) = uE_S;
    uE_vec_L(1,passo_atual+1) = uE_L;
    uE_vec_U(1,passo_atual+1) = uE_U;
    uE_vec_R(1,passo_atual+1) = uE_R;
    uE_vec_B(1,passo_atual+1) = uE_B;

    %Gc(1,passo+1) = u1_S-u2c;
    %Gct(1,passo+1) = (u1_S+u2c)-1;

%-------------------------------------------------------------------------
%               Envia pulso de finalização para o Robô
%-------------------------------------------------------------------------
% Nível Lógico 1
%-------------------------------------------------------------------------
   Confirma_wr = 65535;
   Valores_wr = {Robot_B_wr, Robot_L_wr,...%Carrega valores para escrita
              Robot_R_wr, Robot_S_wr,...%
              Confirma_wr, Robot_U_wr}; %

   writeValue(Comunicacao,Variaveis_CLP_wr,Valores_wr) %Envia valores para o CLP

   pause(1)
%-------------------------------------------------------------------------
% Nível Lógico 0
%-------------------------------------------------------------------------
   Confirma_wr = 0;
   Valores_wr = {Robot_B_wr, Robot_L_wr,...  %Carrega valores para escrita
              Robot_R_wr, Robot_S_wr,...%
              Confirma_wr, Robot_U_wr};    %
```

```
   writeValue(Comunicacao,Variaveis_CLP_wr,Valores_wr)      %Envia valores para o CLP
%--------------------------------------------------------------------------
   end
   disp("Célula Treinada Padrão de Verdade")

%--------------------------------------------------------------------------
%passo_atual = passo_atual+1
%passo_vec(1, passo_atual) = passo_atual;

%--------------------------------------------------------------------------
% Plota gráficos de Aprendizado
%--------------------------------------------------------------------------
plot(passo, uE_vec_S,'r','linewidth',3)
ylim([0 Set_point_S])
grid on
title("Aprendizagem Eixo S")
xlabel("Passos")
ylabel("Grau de Evidência Resultante")
figure

plot(passo, uE_vec_L,'r','linewidth',3)
ylim([0 Set_point_L])
grid on
title("Aprendizagem Eixo L")
xlabel("Passos")
ylabel("Grau de Evidência Resultante")
figure

plot(passo, uE_vec_U,'r','linewidth',3)
ylim([0 Set_point_U])
grid on
title("Aprendizagem Eixo U")
xlabel("Passos")
ylabel("Grau de Evidência Resultante")
figure

plot(passo, uE_vec_R,'r','linewidth',3)
ylim([0 Set_point_R])
grid on
title("Aprendizagem Eixo R")
xlabel("Passos")
ylabel("Grau de Evidência Resultante")
figure

plot(passo, uE_vec_B,'r','linewidth',3)
ylim([0 Set_point_B])
grid on
title("Aprendizagem Eixo B")
xlabel("Passos")
ylabel("Grau de Evidência Resultante")
```

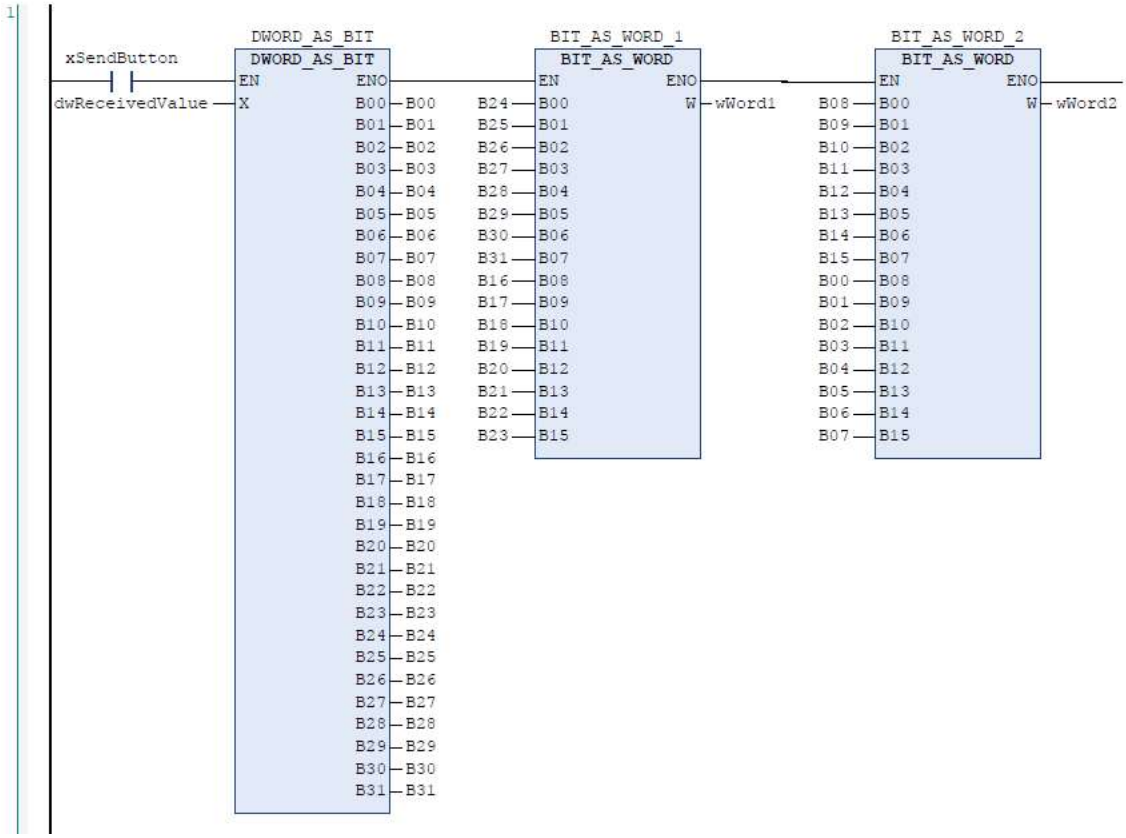
APÊNDICE F – CP - Bloco Funcional 2WORDS TO DWORD

```
3       VAR_INPUT
4           wReceivedWord1  : WORD;
5           wReceivedWord2  : WORD;
6            xSendButton_send : BOOL;
7       END_VAR
8
9       VAR_OUTPUT
10          dwValue_send : DWORD;
11      END_VAR
12
13      VAR
14          B00s : BOOL;
15          B01s : BOOL;
16          B02s : BOOL;
17          B03s : BOOL;
18          B04s : BOOL;
19          B05s : BOOL;
20          B06s : BOOL;
21          B07s : BOOL;
22          B08s : BOOL;
23          B09s : BOOL;
24          B10s : BOOL;
25          B11s : BOOL;
26          B12s : BOOL;
27          B13s : BOOL;
28          B14s : BOOL;
29          B15s : BOOL;
30          B16s : BOOL;
31          B17s : BOOL;
32          B18s : BOOL;
33          B19s : BOOL;
34          B20s : BOOL;
35          B21s : BOOL;
36          B22s : BOOL;
37          B23s : BOOL;
38          B24s : BOOL;
39          B25s : BOOL;
40          B26s : BOOL;
41          B27s : BOOL;
42          B28s : BOOL;
43          B29s : BOOL;
44          B30s : BOOL;
45          B31s : BOOL;
46
47          WORD_AS_BIT1 : WORD_AS_BIT;
48          WORD_AS_BIT_2 : WORD_AS_BIT;
49          BIT_AS_DWORD : BIT_AS_DWORD;
```


1
WORD_AS_BIT1
WORD_AS_BIT
x5sendButton_send
EN
ENO
wReceivedWord1
W
B00
B01
B02
B03
B04
B05
B06
B07
B08
B09
B10
B11
B12
B13
B14
B15
B24s
B25s
B26s
B27s
B28s
B29s
B30s
B31s
B16s
B17s
B18s
B19s
B20s
B21s
B22s
B23s
WORD_AS_BIT_2
WORD_AS_BIT
wReceivedWord2
B08s
B09s
B10s
B11s
B12s
B13s
B14s
B15s
B00s
B01s
B02s
B03s
B04s
B05s
B06s
B07s
BIT_AS_DWORD
BIT_AS_DWORD
X
dwValue_send

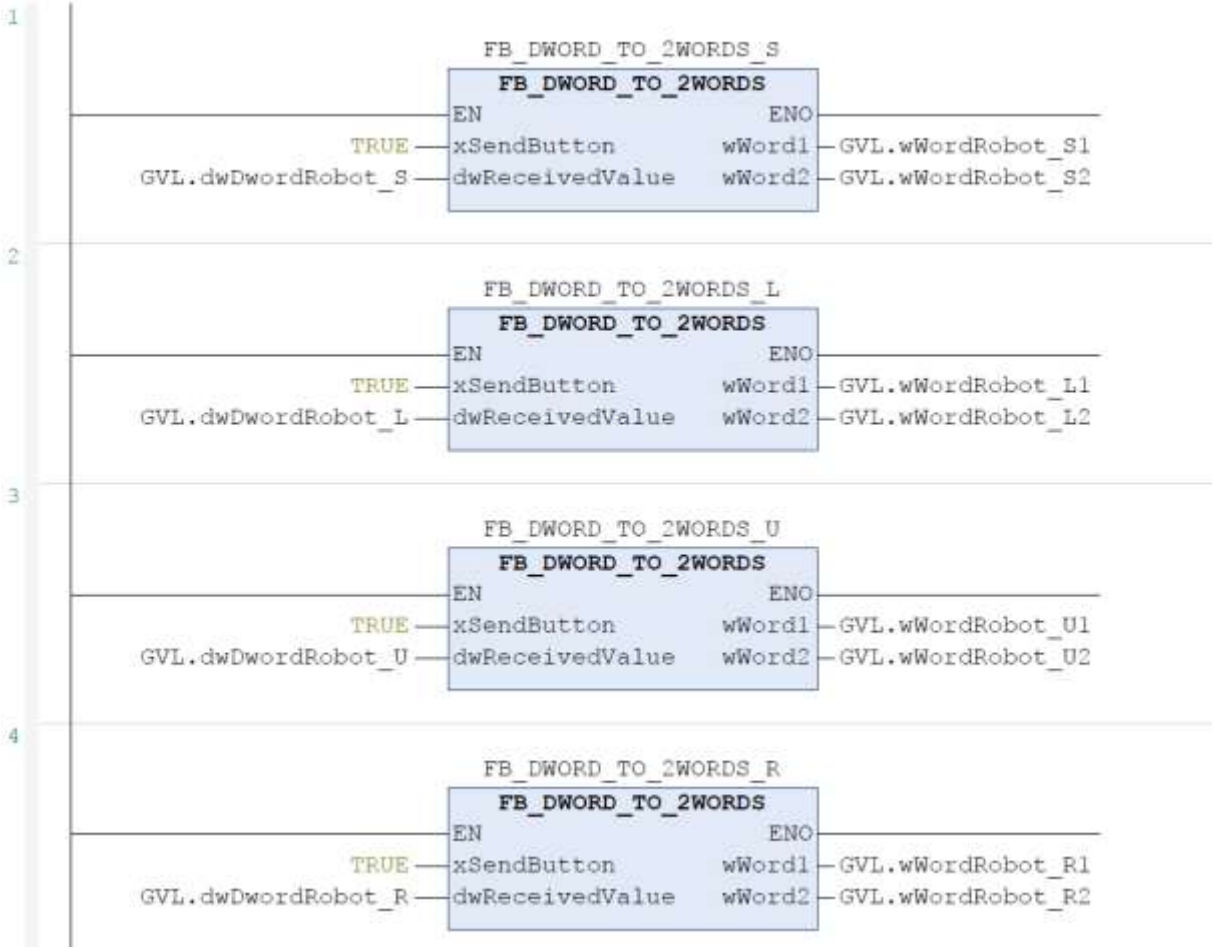
APÊNDICE G – CP - Bloco Funcional DWORDS TO 2WORD

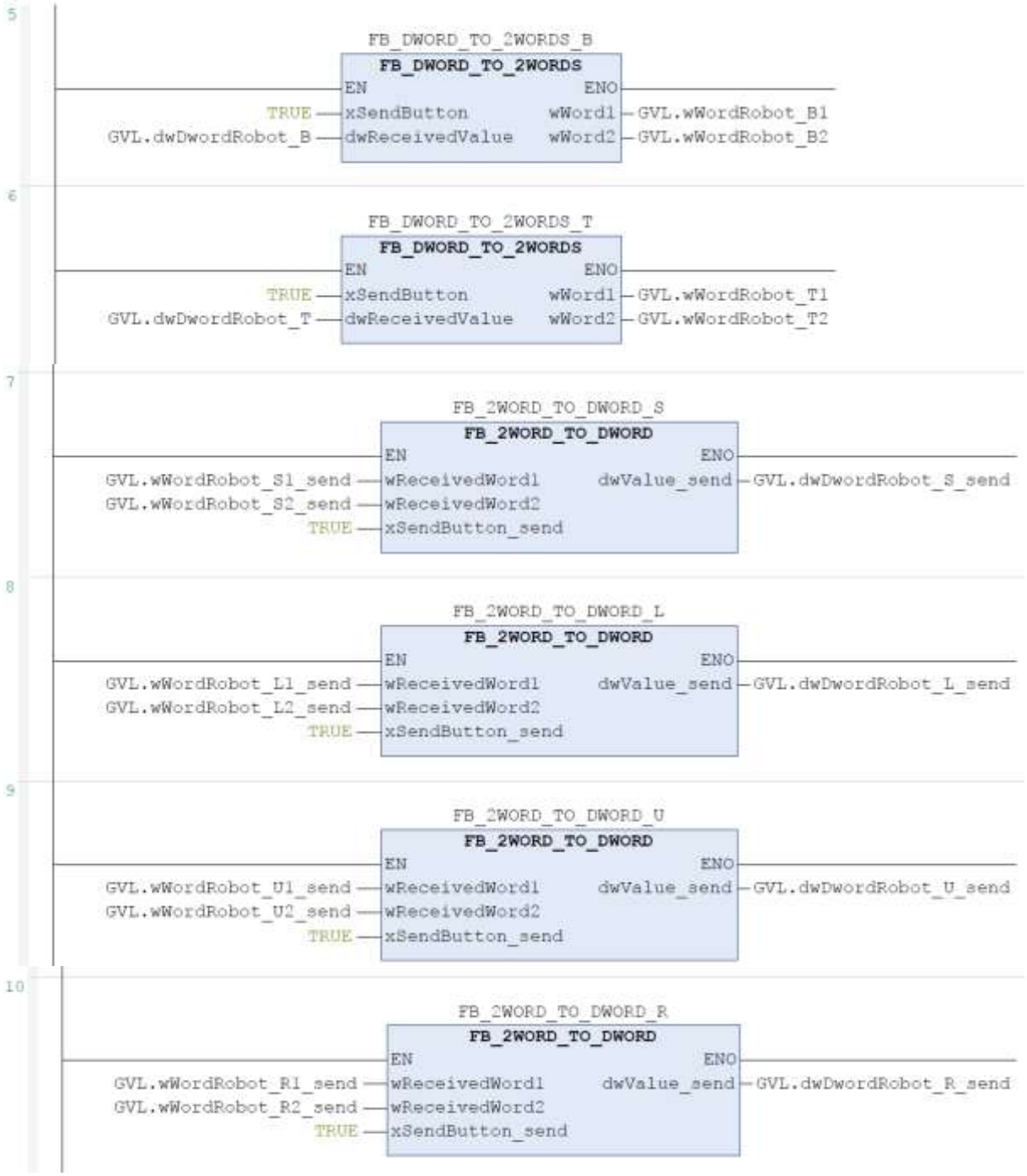
```
1   FUNCTION_BLOCK FB_DWORD_TO_2WORDS
2   VAR_INPUT
3       xSendButton : BOOL;
4       dwReceivedValue : DWORD;
5   END_VAR
6
7   VAR_OUTPUT
8       wWord1 : WORD;
9       wWord2 : WORD;
10  END_VAR
11
12  VAR
13      B00 : BOOL;
14      B01 : BOOL;
15      B02 : BOOL;
16      B03 : BOOL;
17      B04 : BOOL;
18      B05 : BOOL;
19      B06 : BOOL;
20      B07 : BOOL;
21      B08 : BOOL;
22      B09 : BOOL;
23      B10 : BOOL;
24      B11 : BOOL;
25      B12 : BOOL;
26      B13 : BOOL;
27      B14 : BOOL;
28      B15 : BOOL;
29      B16 : BOOL;
30      B17 : BOOL;
31      B18 : BOOL;
32      B19 : BOOL;
33      B20 : BOOL;
34      B21 : BOOL;
35      B22 : BOOL;
36      B23 : BOOL;
37      B24 : BOOL;
38      B25 : BOOL;
39      B26 : BOOL;
40      B27 : BOOL;
41      B28 : BOOL;
42      B29 : BOOL;
43      B30 : BOOL;
44      B31 : BOOL;
45
46      DWORD_AS_BIT : DWORD_AS_BIT;
47      BIT_AS_WORD_2 : BIT_AS_WORD;

48      BIT_AS_WORD_1 : BIT_AS_WORD;
49  END_VAR
50
```

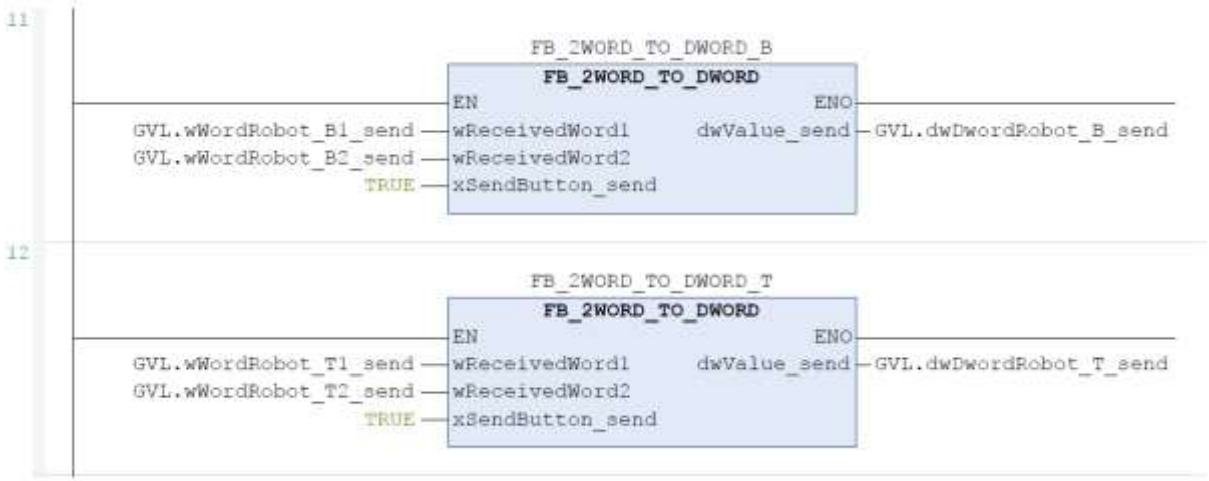

1
xSendButton
dwReceivedValue
DWORD_AS_BIT
DWORD_AS_BIT
EN
ENO
X
B00 B01 B02 B03 B04 B05 B06 B07 B08 B09 B10 B11 B12 B13 B14 B15 B16 B17 B18 B19 B20 B21 B22 B23 B24 B25 B26 B27 B28 B29 B30 B31
BIT_AS_WORD_1
BIT_AS_WORD
EN
ENO
W
wWord1
B24 B25 B26 B27 B28 B29 B30 B31 B16 B17 B18 B19 B20 B21 B22 B23
B00 B01 B02 B03 B04 B05 B06 B07 B08 B09 B10 B11 B12 B13 B14 B15
BIT_AS_WORD_2
BIT_AS_WORD
EN
ENO
W
wWord2
B08 B09 B10 B11 B12 B13 B14 B15 B00 B01 B02 B03 B04 B05 B06 B07
B00 B01 B02 B03 B04 B05 B06 B07 B08 B09 B10 B11 B12 B13 B14 B15

APÊNDICE H – CP – Programa Principal

```
1   PROGRAM SR_Main
2   VAR
3       FB_DWORD_TO_2WORDS_S : FB_DWORD_TO_2WORDS ;
4       FB_DWORD_TO_2WORDS_L : FB_DWORD_TO_2WORDS ;
5       FB_DWORD_TO_2WORDS_U : FB_DWORD_TO_2WORDS ;
6       FB_DWORD_TO_2WORDS_R : FB_DWORD_TO_2WORDS ;
7       FB_DWORD_TO_2WORDS_B : FB_DWORD_TO_2WORDS ;
8       FB_DWORD_TO_2WORDS_T : FB_DWORD_TO_2WORDS ;
9       FB_2WORD_TO_DWORD_S : FB_2WORD_TO_DWORD ;
10      FB_2WORD_TO_DWORD_L : FB_2WORD_TO_DWORD ;
11      FB_2WORD_TO_DWORD_U : FB_2WORD_TO_DWORD ;
12      FB_2WORD_TO_DWORD_R : FB_2WORD_TO_DWORD ;
13      FB_2WORD_TO_DWORD_B : FB_2WORD_TO_DWORD ;
14      FB_2WORD_TO_DWORD_T : FB_2WORD_TO_DWORD ;
15  END_VAR
16
```




5
FB_DWORD_TO_2WORDS_B
FB_DWORD_TO_2WORDS
EN
ENO
TRUE
xSendButton
wWord1
GVL.wWordRobot_B1
GVL.dwDwordRobot_B
dwReceivedValue
wWord2
GVL.wWordRobot_B2
6
FB_DWORD_TO_2WORDS_T
FB_DWORD_TO_2WORDS
EN
ENO
TRUE
xSendButton
wWord1
GVL.wWordRobot_T1
GVL.dwDwordRobot_T
dwReceivedValue
wWord2
GVL.wWordRobot_T2
7
FB_2WORD_TO_DWORD_S
FB_2WORD_TO_DWORD
EN
ENO
GVL.wWordRobot_S1_send
wReceivedWord1
dwValue_send
GVL.dwDwordRobot_S_send
GVL.wWordRobot_S2_send
wReceivedWord2
TRUE
xSendButton_send
8
FB_2WORD_TO_DWORD_L
FB_2WORD_TO_DWORD
EN
ENO
GVL.wWordRobot_L1_send
wReceivedWord1
dwValue_send
GVL.dwDwordRobot_L_send
GVL.wWordRobot_L2_send
wReceivedWord2
TRUE
xSendButton_send
9
FB_2WORD_TO_DWORD_U
FB_2WORD_TO_DWORD
EN
ENO
GVL.wWordRobot_U1_send
wReceivedWord1
dwValue_send
GVL.dwDwordRobot_U_send
GVL.wWordRobot_U2_send
wReceivedWord2
TRUE
xSendButton_send
10
FB_2WORD_TO_DWORD_R
FB_2WORD_TO_DWORD
EN
ENO
GVL.wWordRobot_R1_send
wReceivedWord1
dwValue_send
GVL.dwDwordRobot_R_send
GVL.wWordRobot_R2_send
wReceivedWord2
TRUE
xSendButton_send


11
FB_2WORD_TO_DWORD_B
FB_2WORD_TO_DWORD
EN
ENO
GVL.wWordRobot_B1_send
GVL.wWordRobot_B2_send
TRUE
wReceivedWord1
wReceivedWord2
xSendButton_send
dwValue_send
GVL.dwDwordRobot_B_send
12
FB_2WORD_TO_DWORD_T
FB_2WORD_TO_DWORD
EN
ENO
GVL.wWordRobot_T1_send
GVL.wWordRobot_T2_send
TRUE
wReceivedWord1
wReceivedWord2
xSendButton_send
dwValue_send
GVL.dwDwordRobot_T_send