TRABALHO DE GRADUAÇÃO

Integração e Planejamento de Trajetória de um Robô Cartesiano para Construção de Modelos por Soldagem 3D

Rodrigo Flores Mendes

Brasília, julho de 2019

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Integração e Planejamento de Trajetória de um Robô Cartesiano para Construção de Modelos por Soldagem 3D

Rodrigo Flores Mendes

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Guilherme Caribé de Carvalho, ENM/UnB
Orientador _____

Prof. Jones Yudi Mori Alves da Silva, ENM/UnB _____

Prof. Walter de Britto Vidal Filho, ENM/UnB _____

Brasília, julho de 2019

FICHA CATALOGRÁFICA

MENDES, RODRIGO FLORES
Integração e Planejamento de Trajetória de um Robô Cartesiano para Construção de Modelos por Soldagem 3D,

[Distrito Federal] 2019.

xii, 69p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2019). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Robótica Industrial
2. Soldagem 3D
3. Planejamento de Trajetórias
4. Integração de Sistemas

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

MENDES, R. F., (2019). Integração e Planejamento de Trajetória de um Robô Cartesiano para Construção de Modelos por Soldagem 3D. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-n°17, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 81p.

CESSÃO DE DIREITOS

AUTOR: Rodrigo Flores Mendes

TÍTULO DO TRABALHO DE GRADUAÇÃO: Integração e Planejamento de Trajetória de um Robô Cartesiano para Construção de Modelos por Soldagem 3D.

GRAU: Engenheiro ANO: 2019

Rodrigo Flores Mendes

CLN 409 Bloco C, apt 207, Asa Norte.

70857-530 Brasília – DF – Brasil.

RESUMO

Os manipuladores robóticos são dispositivos programáveis projetados para executar uma imensa variedade de tarefas de forma repetitiva. Esses robôs podem ser utilizados inclusive para realizar a deposição de metal de solda em camadas sucessivas, a técnica conhecida como soldagem 3D. O objetivo deste trabalho é tornar o robô MAXR23-S42-H42-C42 apto a realizar esta técnica.

Neste trabalho de graduação apresenta-se o processo de integração dos sistemas de controle de movimentação e o planejamento de trajetória do robô cartesiano.

Para atingir a integração dos sistemas de controle foi estabelecida a comunicação entre os três servo-drives e o controlador de movimentos via rede industrial CANMotion.Além disso, foi estabelecida a comunicação entre a interface homem-máquina por meio do protocolo de comunicação Ethernet TCP/IP, utilizando o software Vijeo-Designer para projeto gráfico das telas. Toda a configuração dos diferentes tipos de redes de comunicação foi realizada por meio do software SoMachine V4.1.

Para desenvolver o sistema de planejamento de trajetória do robô cartesiano foi feito um interpretador de código G utilizando a biblioteca SM3_CNC do CoDeSys Softmotion, que realiza a leitura de um arquivo em código G, o pré-processamento dos dados e a interpretação dos pontos que serão enviados para os servo-drives. Toda a programação realizada durante o trabalho baseiou-se nas bibliotecas SM3_Basic e SM3_CNC, disponíveis no software SoMachine V4.1, utilizando a linguagem FBD (Function Block Diagram).

A realização das etapas descritas anteriormente possibilitaram o desenvolvimento de rotinas que integram os sistemas de controle de movimento e de interface com o usuário do robô e realizam a interpretação de uma sequência de pontos escritas em linguagem de código G. O sistema configurado e programado possibilita a programação e execução de trajetórias descritas por meio de um código G.

Palavras Chave: Soldagem 3D, SoMachine, Rede CAN-Motion, Planejamento de Trajetória, Integração de Sistemas.

ABSTRACT

Robotic handlers are programmable devices designed to perform a huge variety of tasks repetitively. These robots can even be used to deposit weld metal in successive layers, the technique known as 3D welding. The objective of this work is to make the robot MAXR23-S42-H42-C42 able to perform this technique.

This undergraduate work presents the process of integrating the motion control systems and the trajectory planning of the cartesian robot.

To achieve the integration of the control systems, communication was established between the three servo drives and the motion controller through the CANMotion industrial network. In addition, the communication between the human machine interface was established via the TCP/IP ethernet communication protocol, using Vijeo-Designer software for graphic design of screens. All configurations of the different types of communication networks was performed using SoMachine V4.1 software.

To develop the robot's trajectory planning system, a G code interpreter was done using CoDeSys Softmotion's SM3_CNC library, which reads a G code file, preprocesses the data and interprets the points. that will be sent to the servo drives. All programming performed during the work was based on the libraries SM3_Basic and SM3_CNC, available in SoMachine V4.1 software, using the Function Block Diagram (FBD) language.

Performing the steps described above allowed the development of routines that integrate the robot's motion control and user interface systems and perform the interpretation of a sequence of points written in G code language. The configured and programmed system enabled the execution of paths described by means of a G code.

Keywords: 3D Welding, SoMachine, CAN-Motion Network, Trajectory Planning, Systems Integration.

SUMÁRIO

Lista de Figuras

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

QoS	Qualidade de Serviço (do inglês, *Quality of Service*)
CAN	*Controller Area Network*
CSMA	*Carrier Sense Multiple Access*
COB	Objetos de Comunicação (COB, do inglês *Communication Objects*)
CNC	Comando Numérico Computadorizado
POU	*Program Organization Unit*
CLP	Controlador Lógico Programável
IHM	Interface Homem-Máquina
ASCII	American Standard Code for Information Interchange
IEC	International Electrotechnical Commission
CSP	Cyclic Synchronous Position Mode
CSV	Cyclic Synchronous Velocity Mode
CST	Cyclic Synchronous Torque Mode
TPDO	Transmit Process Data Object

Capítulo 1

Introdução

1.1 Contextualização

Recentemente, com o avanço da tecnologia, as técnicas de impressão 3D têm se tornado cada vez mais populares e acessíveis. A impressão 3D permite a criação de peças tridimensionais, baseadas em modelos computadorizados, a partir da adição de material. Dessa forma a fabricação de peças complexas que seriam custosas de se obterem por meio de técnicas convencionais é facilitada [1].

A impressão 3D também permite a modelagem e geração de trajetória através do uso do computador, o que facilita o desenvolvimento de produtos. O uso do computador também traz flexibilidade e rapidez no desenvolvimento de novos produtos e, por isso, as técnicas de impressão 3D são utilizadas na chamada prototipagem rápida, onde são feitos protótipos de peças para testes e validação [1].

Uma das técnicas utilizadas na impressão 3D é a fusão e deposição (FDM) de material em camadas sucessivas, em que um filamento de material polimérico termoplástico é aquecido e estradado por meio de um cabeçote cujo movimento é previamente planejado para que os cordões de material plástico sejam depositados formando camadas que se superpõe de modo a construir fisicamente um modelo computacional previamente fatiado em camadas paralelas[2].

A soldagem 3D, também conhecida como Shaped Metal Deposition (SMD) e mais recentemente denominada como Wire Arc Additive Manufacturing (WAAM), é uma técnica de fabricação de peças metálicas ainda em desenvolvimento. Essa técnica assemelha-se bastante à FDM com a diferença que o material depositado provém de um arame metálico, que é fundido e depositado utilizando uma tocha de soldagem a arco. O movimento da tocha deve ser planejado de tal forma a se ter uma trajetória contínua de cordões sobrepostos que, quando depositados, resultem na construção da geometria previamente desenhada no computador. Aspectos tais como controle de velocidade de soldagem, balanço térmico e posição de soldagem durante a deposição de metal influenciam na qualidade do solido construído e, também na viabilidade de sua construção[3].

Trabalhos anteriores realizados na Universidade de Brasília (UnB) trabalhos anteriores [8], [5], resultaram no projeto de montagem mecânica e elétrica de um robô cartesiano (Schneider

Electric MAXR23-S42-H42-C42 com servo-drives Lexium 32M e controlador LMC058). Com vistas a continuação dos trabalhos anteriores e facilitar a operação do robô para realização do processo de soldagem 3D, este trabalho visa à integração dos diversos sistemas componentes do robô e à implementação de rotinas para execução de tarefas pré-programadas e comandas via interface homem-máquina, assim como rotinas para interpretação e execução de programas de movimentação escritos em linguagem G, tornando o robô apto a movimentar uma tocha de soldagem durante o processo de soldagem 3D.

1.2 Objetivo

O objetivo principal deste trabalho é integrar os sistemas de movimentação e interface com o usuário e implementar um interpretador de código G para planejamento e realização de caminhos complexos pré-programados nessa linguagem.

Para alcançar esse objetivo definiram-se os seguintes objetivo especificos: -Configuração dos servo-drives (Lexium LXM32M) para o controle de movimento sincronizado de 3 servo-motores via rede industrial Can-Motion,

-Configuração da comunicação em rede entre controlador, IHM e computador do operador para possibilitar a programação e operação de todos os componentes do sistema,

-Desenvolvimento de programas em linguagem nativa do controlador (FBD): busca de referência do robô, ajuste de offsets para referência da peça, movimento manual para posições absolutas e relativas (*jog*), carregamento e execução de programas.

-Programação do controlador e interface homem-máquina (IHM) para leitura e interpretação de arquivo em código G,

1.3 Organização do trabalho

Tendo em vista que o foco deste trabalho é fazer a integração dos sistemas instalados e o planejamento de trajetória do robô MAXR23-S42-H42-C42, é importante verificar a literatura da área para se apoiar nos conceitos para atingir a proposta do trabalho.

O capítulo 2 apresenta os conceitos do protocolo de comunicação CANopen utilizado na configuração da conexão entre os *drives* da família Lexium 32M e o controlador LMC058. Neste capítulo também é abordado o conceito do programa de comando numérico computadorizado utilizado no desenvolvimento do sistema de planejamento de trajetória dos eixos.

No capítulo 3 apresenta os softwares utilizados para o desenvolvimento do trabalho de graduação. O software SoMachine V4.1 utilizado para a configuração da comunicação entre configuração entre controlador e *drives* e a programação básica do controlador.Além disso, o Vijeo-Designer, usado para estabelecer a comunicação entre a IHM e o controlador e criar telas com funções pré-programadas na IHM.

O capítulo 4 detalha quais etapas foram realizadas para concluir a integração dos sistemas do robô cartesiano, a programação feita para realizar o movimento sincronizado dos eixos.

O capítulo 5 detalha as etapas do processo de planejamento de trajetória do robô cartesiano, abordando como foi feita as etapas de leitura do arquivo com extensão (.txt), o pré-processamento e a interpolação dos pontos que formam o caminho criado por um usuário na linguagem G.

O capítulo 6 apresenta os resultados obtidos no trabalho de graduação, além disso, detalha o programa que foi desenvolvido integrando os sistemas e realizando o planejamento de trajetória do robô MAXR23-S42-H42-C42.

Capítulo 2

Revisão Bibliográfica

2.1 Robótica Industrial

Atualmente, há uma necessidade cada vez maior de se realizarem tarefas com grande eficiência e precisão. Além disso, existem também tarefas que devem ser realizadas em lugares de difícil acesso, lugares onde a presença humana pode ser arriscada ou mesmo impossível. Para realizar tais tarefas, torna-se necessária a presença de dispositivos mecatrônicos (robôs).

Os robôs podem ser classificados segundo diversos critérios: quanto à aplicação, quanto à cadeia cinemática, quanto ao tipo de atuadores, etc. Mesmo o termo robô pode ter várias representações, um robô pode ser um veículo autônomo, um humanoide ou mesmo um simples braço mecânico.

Os robôs industriais, por sua vez, são usados em operações em que se deseja alcançar maior velocidade e demandam precisão e trabalho repetitivo. Eles são comumente utilizados na indústria da manufatura, e operam em ambientes relativamente estáticos e produzem em média escala [4]. Os robôs industriais são, em sua grande maioria, máquinas projetadas para substituir o trabalho humano em situações de desgaste físico ou mental, ou ainda situações perigosas e repetitivas no processo produtivo em indústrias.

Robôs industriais têm se estabelecido na indústria manufatureira por mais de 30 anos, e vêm sendo utilizados para trabalhos como empilhamento, seleção, pintura, ordenação, soldagem, entre outros [4]. Atualmente, há novos desenvolvimentos na área de robotização de indústrias, cujo trabalho é perigoso para o homem. Esses trabalhos podem então serem feitos através do controle de robôs a distância. Além disso, enquanto a fábrica está em funcionamento, operadores podem reprogramar o robô para realizar sua próxima tarefa [5].

Os sistemas robóticos são integrados a outros sistemas de automação de uma indústria de modo que permitem produção *just-in-time* e suportam novos níveis de manufatura personalizada de forma economicamente viável. Uma vez programada, uma linha de produção robotizada pode criar diferentes modelos ou fabricar variantes dos mesmos produtos, como a demanda exigir [5].

Com o que foi discutido anteriormente, fica evidente a importância da robótica industrial nos processos atuais, facilitando o trabalho dos funcionários reduzir atuação humana em atividades

repetitivas e aumentar a eficiência da indústria no desenvolvimento de novos produtos.

2.2 Soldagem

A soldagem é o mais importante processo industrial de fabricação de peças metálicas. Processos de soldagem ou processos afins são também utilizados na recuperação de peças desgastadas, para a aplicação de revestimentos de características especiais sobre superfícies metálicas e para corte. O sucesso da soldagem está associado a diversos fatores e, em particular, com a sua relativa simplicidade operacional [6].

Um grande número de diferentes processos utilizados na fabricação e recuperação de peças, equipamentos e estruturas se encaixa no termo soldagem. Classicamente, a soldagem é considerada como um método de união, porém, muitos processos de soldagem ou variações destes são usados para a deposição de material sobre uma superfície, visando a recuperação de peças desgastadas ou para a formação de um revestimento com características especiais. Diferentes processos intimamente relacionados com os processos de soldagem são utilizados para o corte de peças metálicas [6].

Nos últimos anos, técnicas modernas de instrumentação e controle também foram absorvidas pela soldagem, juntamente com os desenvolvimentos na área de robótica e informática. Modelos teóricos e principalmente empíricos têm sido usados para uma melhor compreensão dos fenômenos associados à soldagem. Tudo isto possibilitou o desenvolvimento de sistemas com maior grau de mecanização e automação e, até mesmo, capacidade de tomada de decisão e alteração dos parâmetros de soldagem, durante o processo, independentemente do operador [6]. Essas mudanças oferecem mais segurança aos profissionais envolvidos neste processo, já que a soldagem está associada a diversos riscos, incluindo riscos ergonômicos. Os serviços de soldagem impõem condições e posturas críticas de trabalho, riscos ocupacionais devido a radiação, riscos ocupacionais devido aos ruídos, riscos ocupacionais devido a temperaturas extremas, riscos ocupacionais devido a poeiras e partículas e riscos ocupacionais devido aos fumos e gases da solda [7].

Os consumíveis para soldagem têm evoluído de acordo com a demanda, sendo adaptados para aplicação aos novos materiais e equipamentos, de forma cada vez mais rápida e eficiente, contribuindo para uma diminuição nos tempos e custos das operações de soldagem. O resultado final é um grande aumento na qualidade e produtividade com menor dependência de habilidade manual do soldador [6]. Do ponto de vista da automação do processo a movimentação da tocha deve ser feita de modo semelhante ao realizado por um soldador, buscando movimento suave com velocidade constante durante a deposição de material. No caso da soldagem 3D, o planejamento da trajetória a ser seguida pela tocha deve ser feito de modo que, ao depositar o metal em camadas, se possa construir a geometria previamente projetada no ambiente computacional.Essa trajetória deve ser planejada a partir do fatiamento do modelo computacional e extração de pontos no espaço que, se seguidos por uma tocha de soldagem depositando metal, reconstrói-se fisicamente a geometria previamente definida no computador. O fatiamento do modelo computacional depende da entrada de parâmetros que são relacionados ao processo de soldagem assim como ao material utilizado para

a construção do modelo metálico. Dentre esses parâmetros de entrada do programa de fatiamento a altura esperada para cada camada depositada é o principal fator geométrico necessário a geração dos pontos da trajetória do robô. Outros aspectos tais como largura dos cordões depositados, que tem relação com os parâmetros de soldagem e com a energia de soldagem devem ser obtidos a partir de bancos de dados previamente estabelecidos relativos ao processo de soldagem e ao material a ser depositado. Do ponto de vista da programação do robô, o que importam são os pontos da trajetória (dotados de orientação) e a velocidade com que a tocha deve ser deslocada ao passar por esses pontos. Em trabalho anterior realizado na UnB, desenvolveu-se um software de fatiamento de modelos sólidos tipo casca e geração de pontos e trajetória contínua para a programação do robô IRB 2000 [3].

2.3 Robô MAXR23-S42-H42-C42

O robô MAXR23-S42-H42-C42 da *Schneider Electric* pode ser visto na figura 2.1, bem como os dispositivos de controle e alimentação dos eixos, que serão apresentados nos capítulos adiante.

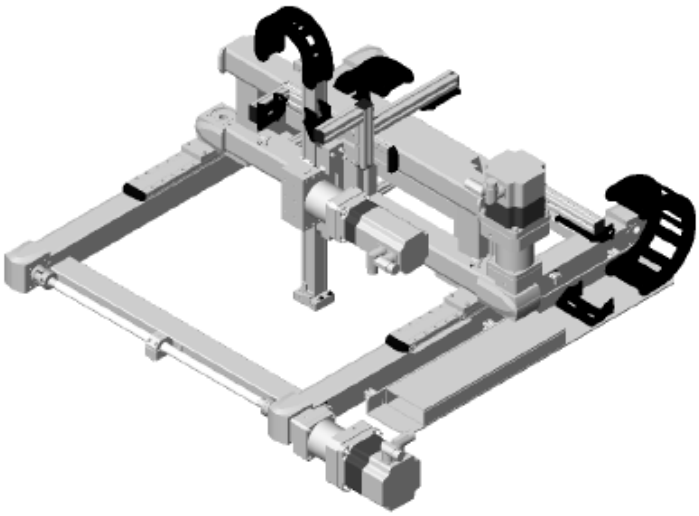


Figura 2.1: Robô MAXR23-S42-H42-C42 da *Schneider Electric* [11].

A tabela 2.1 detalha as principais características do manipulador do robô.

Tabela 2.1: Principais características do robô.[8]

Parâmetro	Unidade	Característica
Transferência de movimento	-	Correia dentada 25HTD-5M
Tipo de guia	-	Guia por rolamento
Carga útil	kg	15
Constante de alimentação	mm/rev	155
Diâmetro efetivo da polia da correia dentada	mm	49.338
Stroke Nominal X/Y	mm	800
Stroke Nominal Z	mm	500
Velocidade máxima X/Y	m/s	1,27
Velocidade máxima Z	m/s	1,0
Aceleração máxima X/Y	m/s^2	7,03
Aceleração máxima Z	m/s^2	8,0
Resolução	mm	0.1

Este robô é dotado de três bases que completam os eixos X, Y e Z. O eixo X é formado pela base MAXS2BB e é representado na figura 2.2. As principais características dessa base são:

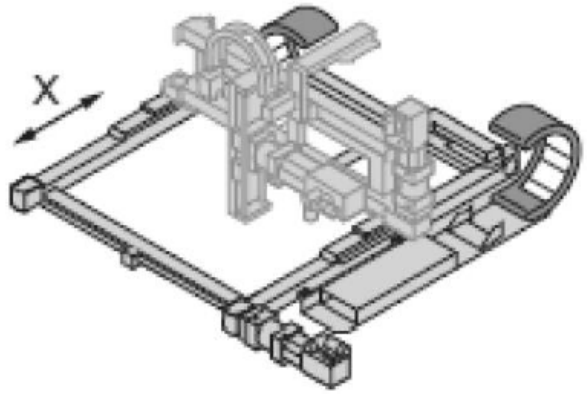


Figura 2.2: Eixo X do robô MAXR23-S42-H42-C42 da *Schneider Electric* [11].

O eixo Y é formado pela base MAXH2BB, mostrado na figura 2.3, que possui as seguintes características:

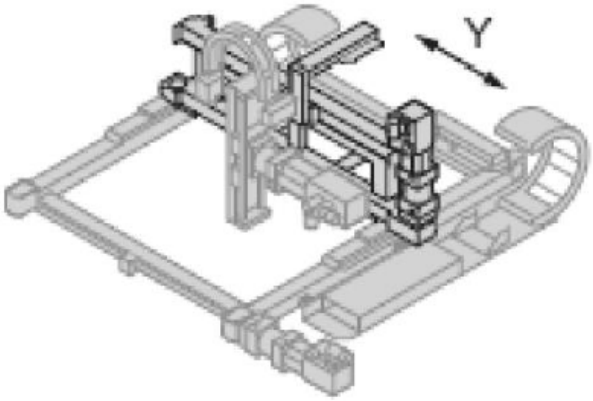


Figura 2.3: Eixo Y do robô MAXR23-S42-H42-C42 da *Schneider Electric* [11].

Por último, o eixo Z é formado pela base CAS42BB, mostrado na figura 2.4, cujas características são:

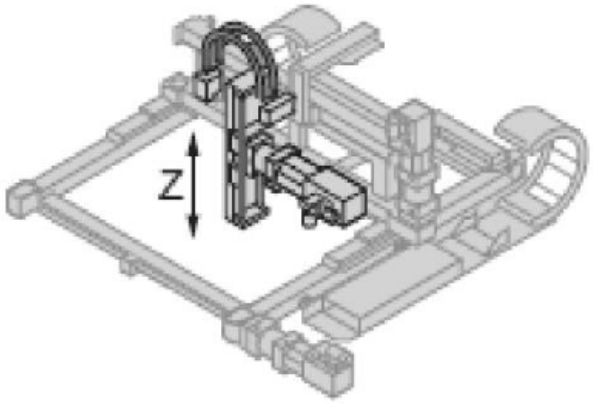


Figura 2.4: Eixo Z do robô MAXR23-S42-H42-C42 da *Schneider Electric* [11].

Maiores detalhes relativos a estrutura do robô podem ser obtidos na referência [8]

O volume de trabalho desse robô é de 800x800x500 mm, ou seja, o robô pode fazer movimentação livre dentro deste volume de atuação. Cada eixo é movimentado por um servo-motor diferente e possui uma função específica para a movimentação do robô. Por exemplo, os servo-motores dos eixos X e Y se diferenciam, principalmente, em termos de torque. Como o eixo X é responsável por carregar toda a carga a ser movida pelo robô, além de carregar os eixos Y e Z, o torque fornecido por ele deve ser maior. Entre os eixos Y e Z, a principal diferença está no sistema de frenagem do eixo Z, para evitar que o eixo desça repentinamente ao ser desligado ou quando estiver movimentando uma carga. As principais características dos servo-motores podem ser vistas na tabela 2.2.

Tabela 2.2: Principais características do servo-motores.

Servo-Motor	BMH0702P06A2A	BMH0701P06A2A	BMH0701P06F2A
Frenagem	Não	Não	Sim
Torque contínuo	2,48 N.m	1,4 N.m	1,4 N.m
Torque máximo	7,44 N.m	4,2 N.m	4,2 N.m
Torque nominal	2,23 N.m	1,34 N.m	1,34 N.m
Corrente nominal	2,7 A	1,75 A	1,75 A
Potência nominal	0,7 kW	0,42 kW	0,42 kW

Neste trabalho são utilizados 3 servo-drives de acionamento. Estes são utilizados para fazer a alimentação e controle dos três eixos do robô. Os servo-drives instalados no painel de comando do robô são da família Lexium 32M (ou LXM) da Schneider Electric. Mais especifícamente os modelos utilizados são o LXM32MU90M2 e LXM32MD18M2, cujas características principais são:

• LXM32MU90M2 (2 unidades para acionamento dos eixos Y e Z):

a) Servo drive de controle de um eixo;

b) Drive modular;

c) Corrente de entrada de 4,5 ampères;

d) Corrente de pico de saída de 9 ampères;

e) Corrente nominal de 3A;

f) Tensão de alimentação de 115/200/240 Vac;

g) Potência nominal: 0,5 kW a 230 Vac;

• LXM32MD18M2 (1 unidade para acionamento do eixo X):

a) Servo drive de controle de um eixo;

b) Drive modular;

c) Corrente de entrada de 8,4 ampères;

d) Corrente de pico de saída de 18 ampères;

e) Corrente nominal de 6A;

f) Tensão de alimentação de 115/200/240 Vac;

g) Potência nominal: 1 kW a 230 Vac

A figura 2.5 mostra a visão geral das conexões do dispositivo, enquanto a tabela 2.3 explica a funcionalidade de cada conexão.

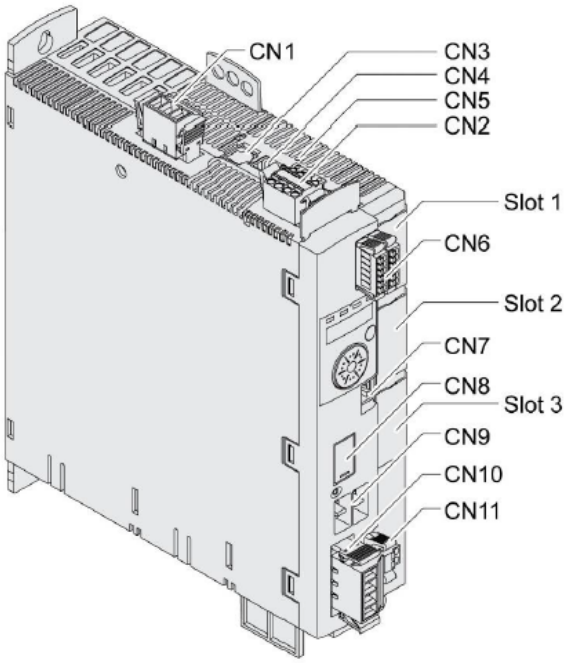


Figura 2.5: Visão geral do dispositivo LXM32M [12].

Tabela 2.3: Visão geral das conexões do servo.

Conexão	Atribuição
CN1	Fonte de alimentação (Power stage supply)
CN2	Fonte de controle e função de segurança STO (Safe Torque Off)
CN3	Encoder 1 (Motor encoder)
CN4	PTO (Pulse Train Out – encoder simulation ESIM)
CN5	PTI (Pulse Train In – sinais A/B, P/D e CW/CCW)
CN6	Entradas/Saídas digitais
CN7	Interface de comissionamento Modbus
CN8	Braking resistor externo
CN9	Conexão DC bus para operações em paralelo
CN10	Motor phases
CN11	Holding brake (Conexão com o freio do manipulador)
Slot 1	Módulo de segurança ou módulo E/S
Slot 2	Encoder 2 (módulo encoder)
Slot 3	Módulo Fieldbus

2.4 Protocolo de Comunicação CANOpen

As redes de controle industriais possuem requisitos diferentes das redes comerciais. Uma das principais diferenças entre estas redes é que as redes de controle industriais são conectadas em equipamentos físicos (um motor, por exemplo) para controle e monitoramentos de ações e condições do mundo real [13]. Para garantir este trabalho em tempo real, há um conjunto de requisitos de Qualidade de Serviço (*Quality of Service*, QoS) que fazem parte das redes industriais.

Tendo conhecimento da importância de atender a necessidade do trabalho em tempo real, muitos dos sistemas *fieldbus* e protocolos de controle por rede estão sendo desenvolvidos com este intuito [4].

Há protocolos que garantem a qualidade de serviço que as redes industriais requerem. Esses protocolos aplicam diversas técnicas para manter os dados consistentes, não sofrer interferências externas e garantir sua transmissão [13]. Dentre os protocolos de redes industriais mais representativos estão: CAN, EtherNet/IP, PROFIBUS, PROFINET, INTERBUS, *Foundation Fieldbus* [14].

O protocolo CAN [15] (*Controller Area Network*) atua nas duas primeiras camadas da rede (física e enlace). Este protocolo também dispõe de um conjunto de recursos que atendem os requisitos de uma rede industrial. Alguns desses recursão são a consistência na transmissão de dados, a rapidez da transmissão dos mesmos e mecanismos de detecção de erros. Baseados no protocolo CAN, existem três protocolos de referência que atuam na camada de aplicação [16]. Estes protocolos são o CANopen [18], o DeviceNet [19] e o Smart Distributed System (SDS) [20] - baseados no padrão *Controller Area Network* (CAN).

Um destes protocolos é o CANOpen que é adotado pela Schneider Electric para comunicação com dispositivos de campo a partir do controlador LMC058, que foi utilizado neste projeto, e será o único discutido com mais detalhes. O protocolo CANMotion, também utilizado neste trabalho para comunicação entre os *servo-drives* e o controlador, é construído sobre o protocolo de comunicação CANOpen, sua aplicação é dedicada ao movimento sincronizado de até 6 acionamento (4 *servo-drives* e 2 *stepper-drives*) e trabalha com *baud-rate* de até 1000 kbp/s [9].

O protocolo que se baseiam tanto o CANOpen quanto o CANMotion, é um protocolo de comunicação serial, definido no padrão ISO 11898, com arquitetura multi-mestre baseada em *broadcast* [13]. Sua arquitetura padrão pode ser observada na figura 2.6.

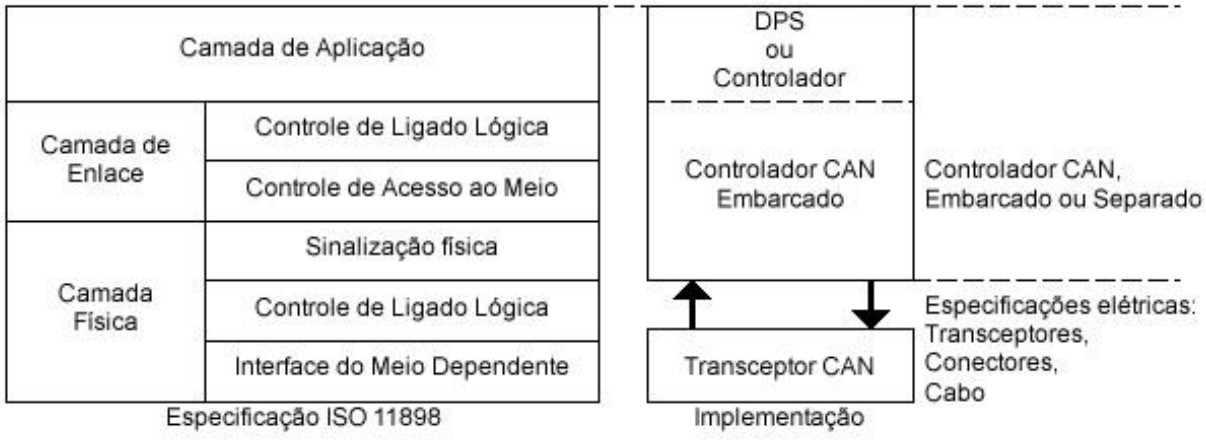


Figura 2.6: Arquitetura padrão do protocolo CAN [13].

Este protocolo provê uma solução geral para controle de redes para muitos requisitos industriais, e pode ter sua aplicação distribuída em um equipamento ou em uma célula de trabalho [17]. O

protocolo tem um bom custo-benefício, permite transmissão de dados de até 1 Mbits/s e, como esperado de uma rede industrial, pode ser implementado em sistemas de tempo real. Além disso, adota mecanismos robustos de detecção de erros. O protocolo também permite a substituição de componentes da rede durante a operação do sistema.

O CAN utiliza *Carrier Sense Multiple Access* (CSMA) com detecção de colisão e arbitragem na prioridade da mensagem. As colisões são resolvidas por arbitragem de lógica binária, com base na prioridade pré-programada de cada mensagem [21] [22]. Os dispositivos deste protocolo podem ter identificador de 11 bits ou 29 bits, também chamados de CAN 2.0A e CAN 2.0B, respectivamente [15] [21], o CANOpen utiliza o CAN 2.0A, portanto o campo identificador tem 11 bits. Os pacotes de dados para o CAN 2.0A e o CAN 2.0B estão ilustrados nas figuras 2.7 e 2.8.

Figura 2.7: Campos do pacote de dados no CAN 2.0A [13].

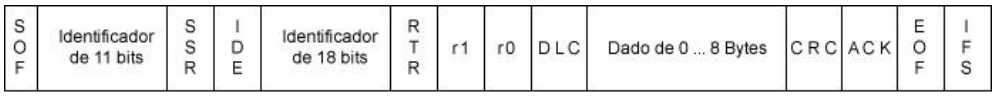


Figura 2.8: Campos do pacote de dados no CAN 2.0B [13].

Como pode ser observado pela figura 2.7, os pacotes de dados no CAN 2.0A possuem os seguintes campos [13]:

• SOF: Início de quadro com apenas 1 bit dominante. Um bit marca o início da mensagem e é usado para sincronizar no barramento após ficar ocioso.

• Identificador: Valor que estabelece a prioridade da mensagem, quanto menor o valor maior a prioridade da mensagem.

• RTR: Único bit dominante - nível lógico alto - para pedido de transmissão remota. Todos os nós recebem o pedido, porém o campo Identificador determina o nó específico. A resposta também chega em todos os nós.

• IDE: Único bit dominante que significa que o campo Identificador não segue o padrão estendido.

• r0: Bit reservado.

• DLC: Dado de 4 bits que contêm o número em bytes do dado que está sendo transmitido.

• Dado: Dado da aplicação a ser transmitido que pode chegar até 64 bits.

• CRC: Teste de redundância cíclica de 16 bits para detecção de erro.

• ACK: Todo nó que receber a mensagem livre de erros substitui o bit recessivo desse campo por um bit dominante. Caso o receptor perceber algum erro, ele deixa o bit como está e descarta a

mensagem. O nó transmissor repete a mensagem após a arbitragem das prioridades. ACK possui 2 bits, um para escrita e outro como delimitador.

• EOF: 7 bits que indicam o final do quadro (mensagem) e verificam o erro de bitstuffing.

• IFS: 7 bits que indicam o tempo necessário para o controlador mover o quadro (mensagem) para a memória(aplicação).

Além dos campos anteriores, o CAN 2.0B também possui os seguintes campos:

• SSR: Bit único que substitui o RTR na posição da mensagem como uma área reservada no formato estendido.

• IDE: Bit recessivo que indica a extensão do Identificador com acréscimo de 18 bits.

• r1: Bit reservado adicional.

As mensagens transmitidas podem ser de quatro tipos [5]:

1. Dado: tipo mais comum, e inclui os campos de arbitragem, dado, RCR e ACK;

2. Remota: tem o propósito de solicitar dado de outro nó da rede. É similar à mensagem de dado, porém o campo RTR contém um bit recessivo e o campo de dados é vazio;

3. Erro: Tipo especial que viola o formato de mensagem CAN e é transmitida caso seja detectado um erro em uma mensagem por algum nó. Uma mensagem é considerada sem erro quando o último bit no campo EOF é recebido como bit recessivo livre de erro, se o bit no EOF for dominante, isso faz o transmissor reenviar a mensagem;

4. Sobrecarga: Transmitido quando um nó está ocupado e é usado, principalmente, para fornecer atraso extra entre mensagens. Tem formato similar à mensagem de erro.

Resistores de pull-up são utilizados para manter o estado recessivo e são conectados aos dois fios do meio de transmissão da rede CAN, e nesse meio de transmissão é avaliada a diferença de tensão entre os fios. Como são utilizados esses resistores de pull-up, a rede recebe um número constante de bits recessivos sem que seja transmitida nenhuma informação. Assim, para marcar o começo de uma transmissão é necessário um estado lógico diferente [21] [22].

O acesso ao barramento é um evento dirigido e ocorre de maneira aleatória; mas, caso haja simultaneidade de acesso, ele será implementado através da arbitragem lógica não destrutiva, fazendo com que o nó vencedor continue transmitindo a mensagem, sem destruí-la ou corrompê-la [21] [22].

Já o CANopen, que é baseado no CAN, é um protocolo de comunicação de alto nível e padroniza a comunicação entre dispositivos e aplicações da rede. O CANopen trata do endereçamento, roteamento, confiabilidade fim-a-fim, sincronização, padronização e representação dos dados. A camada de aplicação é responsável pela descrição de como configurar, transferir e sincronizar os dispositivos da rede [23].

A rede CANopen deve possuir um mestre, que é responsável por gerenciar a rede. Uma rede pode possuir até 127 escravos (nós). Todo nó da rede possui uma lista de objetos denominada

de dicionário de objetos, que contém objetos de comunicação (COB, do inglês *Communication Objects*) responsáveis pela comunicação entre dispositivos da rede. São cinco objetos: objeto de serviço de dados, responsável pelo acesso direto ao dicionário de objetos de um dispositivo da rede; objeto de processamento de dados, usado para acessar os dados de um dispositivo; objeto de emergência, responsável pelo envio de mensagens para indicar a ocorrência de erros no dispositivo; objeto de sincronização,que permite a um dispositivo enviar uma mensagem de sincronização para toda a rede, periodicamente; objeto de gerenciamento de rede, para o mestre da rede gerenciar os seus serviços de controle do dispositivo e serviço de controle de erros nos nós da rede [23].

O protocolo CANopen utiliza a mensagem padrão do CAN com o campo Identificador dividido em duas partes [24]: a primeira, representada por quatro bits, é usada para a identificação da função; a segunda, com sete bits, é usada para identificação do nó.

A mensagem comppleta do protocolo CANopen é mostrada na figura 2.9. A união do campo Identificador com o campo RTR resulta no Identificador de Objetos de Comunicação (COB-ID, do inglês *communication object identifier*), ilustrado na Figura 2.10.

SOF	Identificador da função	Identificador do nó	*RTR*	Reservado RB1,RB0	*DLC*	DADO	*CRC*	*ACK*	*EOF*	NOME DO CAMPO
1	4	7	*1*	2	4	8x8	*16*	*2*	*7*	BITS/CAMPO
---	{0 ... 16}	{0 ... 127}	{0 ... 1}	---	{0 ... 7}	{0 ... 255}x8	---	---	---	FAIXA DE VALOR

Figura 2.9: Mensagem completa do protocolo CANopen [13].

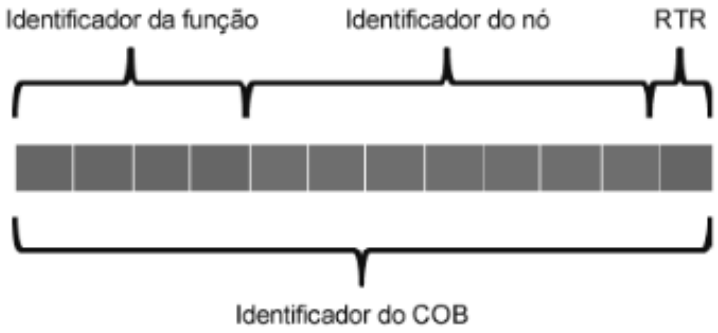


Figura 2.10: Campo Identificador (em bits) modificado pelo protocolo CANopen [13].

O CANMotion é um protocolo baseado no CANOpen dedicado a aplicações de controle de movimento sincronizado em múltiplos eixos, a implementação presente no controlador LMC058 permite o sincronismo de até 6 eixos sendo no máximo 4 *servo-drives* da família Lexium 32M e 2 *stepper-drives* da linha SD328[10].Além disso, o protocolo CANMotion possibilita 3 modos síncronos cíclicos:

-CSP (*Cyclic Synchronous Position*)

-CST (*Cyclic Synchronous Torque*)

-CSV (*Cyclic Synchronous Velocity*)

Durante todo ciclo de tarefas de movimentos (período de ciclo de sincronização (μs)) um novo *setpoint* (posição, torque ou velocidade) é calculado e enviado a cada *servo-drive* por meio de mensagens RPDO (*Receive Process Data Object*). O mecanismo de sincronização faz com que todos os *servo-drives* atualizem o seu *setpoint* ao mesmo tempo. O novo valor atual de cada *servo-drive* é enviado ao controlador por mensagens TPDO (*Transmit Process Data Object*) a máxima variação estatistica do mecanismo de sincronização é de 75 μs.

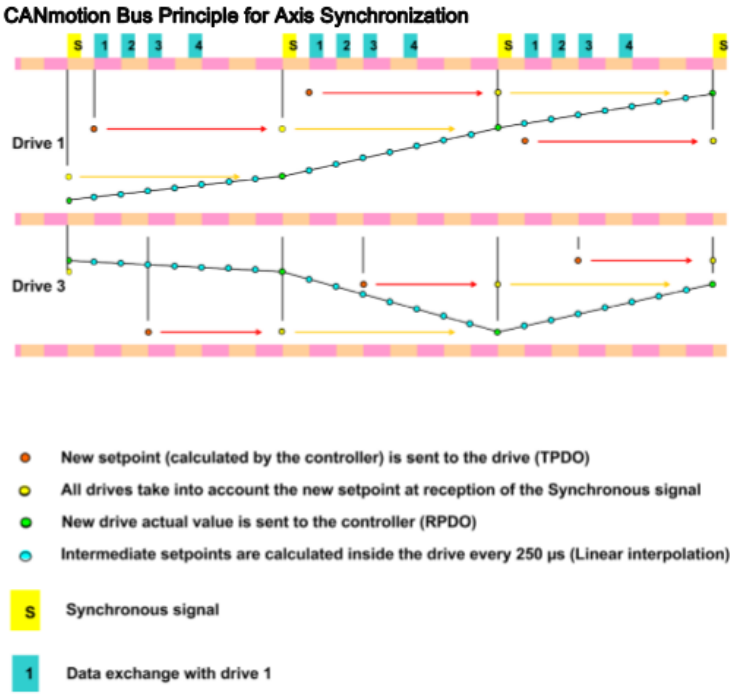


Figura 2.11: Princípio do barramento CANMotion para sincronização de eixos[10].

2.5 Comando Numérico Computadorizado

O comando numérico é um sistema de interpretação de dados que converte um código abstrato em instruções às máquinas-ferramenta de comando numérico. Uma máquina de comando numérico por sua vez, nada mais é do que servoatuador para o comando numérico.

Com o avanço tecnológico, as unidades de comando numérico foram substituídas por computadores; surgiu, então, o comando numérico computadorizado (CNC).

O CNC é uma tecnologia de um horizonte mais amplo que o controle numérico. Ele não se restringe apenas à geração de trajetória de ferramentas de corte girantes. Sua aplicação se estende atualmente a praticamente todos os processos de usinagem (convencionais ou não convencionais).

O sistema de controle de uma máquina CNC é o responsável por ativar as funções necessárias para cada sequência particular de operações. Mas, para que isto ocorra, um programa deve ser introduzido ao sistema de controle. A partir da leitura deste programa, o sistema converte as

informações recebidas em impulsos de controle para a máquina.

O programa de usinagem de uma peça é um arquivo de texto, elaborado seguindo uma determinada sintaxe e formato; que contém instruções de usinagem necessárias para a fabricação da peça [25].

O desenvolvimento deste programa é determinado pelo fabricante do sistema segundo regras padronizadas. Neste projeto foi utilizado o padrão DIN 66025, especificamente, as instruções da letra G (comumente conhecido como código G), reconhecidas por serem funções preparatórias. Estas funções controlam principalmente os deslocamentos de ferramenta. As instruções do código G são apresentadas na tabela 2.4.

Tabela 2.4: Principais instruções do código G de acordo com o padrão DIN 66025.

Código	Significado
G0	Posicionamento rápido
G01	Interpolação linear
G02	Interpolação circular e helicoidal (sentido horário)
G03	Interpolação circular e helicoidal (sentido anti-horário)
G04	Tempo de espera
G10	Parar o pré-processamento de blocos dinâmicos
G17	Seleção no plano xy
G18	Seleção no plano xz
G19	Seleção no plano yz
G20	Pulo condicional
G40	Cancelar compensação de diâmetro da ferramenta de corte
G42	Começar compensação de diâmetro da ferramenta de corte
G90	Sistema de coordenadas absolutas
G91	Sistema de coordenadas relativas
G92	Definir posição sem movimento
G98	Interpretar os valores subsequentes das coordenadas X/Y como absolutos
G99	Interpretar valores subsequentes das coordenadas X/Y como relativos ao ponto inicial (padrão)

A unidade básica da programação em comando numérico é o bloco, que é visto na forma impressa como uma linha do texto. Cada bloco pode conter um ou mais palavras, que consistem em letras que descrevem configurações a serem feitas, ou funções a serem realizadas, seguidos de campos numéricos, que por sua vez, fornecem valores a essa função.

A sintaxe do código G possui o seguinte formato:

1. Número de linhas;
2. Comandos;
3. Parâmetros dos comandos;
4. *Checksum*;
5. Comentários.

O número da linha é a primeira parte de um código G, este número deve ser precedido da

letra N. Assim, na primeira linha de código G, o primeiro campo deve ser preenchido com N1; na segunda linha, com N2 e assim por diante.

O comando é a palavra reservada, como os representados na tabela 2.4, que declara a ação que deve ser executada, no código G não pode haver mais de 1 comando por linha.

Os comandos podem estar acompanhados de argumentos que complementam a ação (parâmetros). Os parâmetros também são representados por palavras reservadas (letras seguidas de números que podem ser positivos ou negativos). Alguns parâmetros mais comuns são:

S (*Speed*): representa a velocidade de rotação nas CNCs, parâmetro genérico no ramo de impressão 3D (utilizado para distância ou temperatura).

X, Y e Z: representam as coordenadas/distâncias podendo ser absolutas ou relativas dependendo do comando em si.

A, B e C: representa as coordenadas ou distâncias angulares, assim como os parâmetros X, Y e Z, podem ser absolutas ou arbitrárias às medidas.

O *checksum* é um campo que tem como objetivo a verificação da consistência do código. Porém, este campo não é usado em impressão 3D.

Os comentários representam as informações que são ignoradas pelo interpretador de código G e são definidos pelo caractere ";" (ponto e vírgula).

Tabela 2.5: Exemplo de Código G.

Linha	Comando	Parâmetros dos Comandos	Descrição
N10	G0	X0 Y0	Realiza o posicionamento rápido para a posição (0,0);
N10	G1	X100 Y100	Realiza a interpolação linear para a coordenada (100,100);
N20	G2	X200 Y100 R50	Realiza a interpolação com raio R para coordenada (200,100);

2.6 Modo de Operação dos Drives

A configuração do modo de operação dos *drives* Lexium 32M (vide figura) 2.12 é uma etapa neccessária para estabelecer o funcionamento do *drive* de acordo com a sua aplicação. A família Lexium 32M possui 5 modos de funcionamento: *Profile Torque*, *Profile Velocity*, *Electronic Gear*, *Jog* e *Motion Sequence*.

Para a aplicação de soldagem 3D a escolha do modo de operação vem do fato que para a execução de uma trajetória de deposição de metal é necessário que os eixos estejam sincronizados e ocorre o movimento suave, preferencialmente com uma velocidade baixa e constante de modo a garantir uma deposição uniforme de metal. Levando em consideração esses pontos a escolha do modo de operação dos *drives* foi a de *Motion Sequence*.

Figura 2.12: Modo de Operação dos Drives

No modo de operação *Motion Sequence*, os movimentos são iniciados por meio de conjuntos de dados parametrizáveis. Um conjunto de dados parametrizável contém configurações do tipo de movimento (tipo de conjunto de dados) e valores alvo (como a velocidade alvo e a posição alvo).

Além disso, é possível especificar dentro do conjunto de dados quando outro conjunto subsequente deve ser iniciado, por exemplo, assim que o movimento foi encerrado. É possível também definir uma condição de transição para iniciar o conjunto de dados. Os conjuntos de dados são parametrizados por meio do comissionamento dos *drives*. O comissionamento consiste na garantia que os componentes estejam projetados, instalados, testados, operados e mantidos de acordo com as necessidades e requisitos operacionais do projeto.

Um conjunto de dados pode ser iniciado de duas maneiras diferentes: com ou sem uma sequência. Ao iniciar com uma sequência o conjunto de dados definidos se inicia, se há um conjunto subsequente, o conjunto de dados subsequente é iniciado assim que a condição de encerramento do primeiro conjunto é satisfeita.

No caso em que não há uma sequência o conjunto de dados definido é iniciado. Se um conjunto subsequente tiver sido definido, porém se o modo utilizado for sem sequência esse conjunto subsequente não será iniciado ao termino do conjunto anterior e o movimento será encerrado.

O conjunto de dados pode ser dos tipos movimento para um valor de posição específico (movimento absoluto, movimento aditivo ou movimento relativo), movimento a uma velocidade específica, *homing* do eixos do robô (movimento de referência ou configuração de posição), repetição de uma determinada sequência (1 ... 65535), movimento sincronizado com sinais de valores de referência externos.

2.7 Interconexão dos Dispositivos do Robô MAXR23-S42-H42-C42

Abordando o tema das conexões entre os *hardwares* do robô cartesiano a figura 2.13 exibe como é feita essa comunicação. Em primeiro lugar tem-se a comunicação entre o controlador LMC058 e os *drives* Lexium 32M que é feita via rede industrial CANMotion. O CANMotion além de ser o modo de comunicação entre o controlador e os *drives* ela também é o meio de comunicação entre os *drives*.

A comunicação entre a IHM HMIS5T e o controlador LMC058 é feita por meio de comunicação Ethernet, importante ressaltar que não existe uma comunicação direta entre a IHM e os *drives* Lexium 32M.

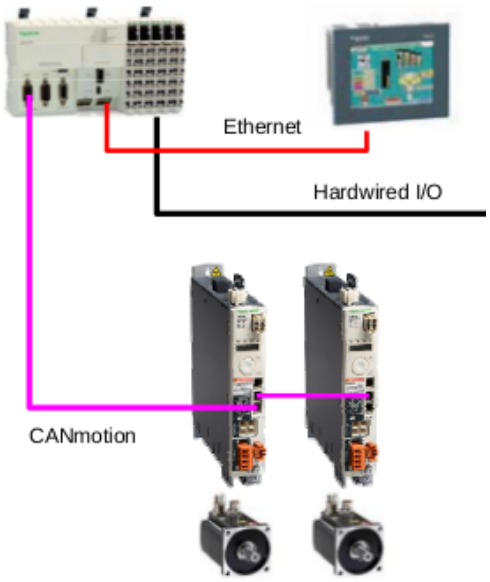


Figura 2.13: Interconexão dos Dispositivos. [27]

No sistema instalado no quadro de comando do robô incluem-se aos elementos mostrados na figura 2.13 um *servo-drive* adicional e um *switch* que funciona como ponte entre a IHM, o controlador e o computador.

Capítulo 3

Softwares utilizados

3.1 SoMachine

De acordo com os manuais fornecidos pela Schneider Electric [26] [27], o software SoMachine V4.1 é definido como uma solução para auxílio no desenvolvimento, configuração e comissionamento de todo o maquinário em um único ambiente (incluindo lógica, controle de motor, interface homem-máquina e funções de automação relacionadas a rede) [26] [27].

O software SoMachine na versão 4.1 fornece uma interface (3.1) de usuário poderosa e conveniente para o mapeamento de hardware e funcionalidade do robô cartesiano.

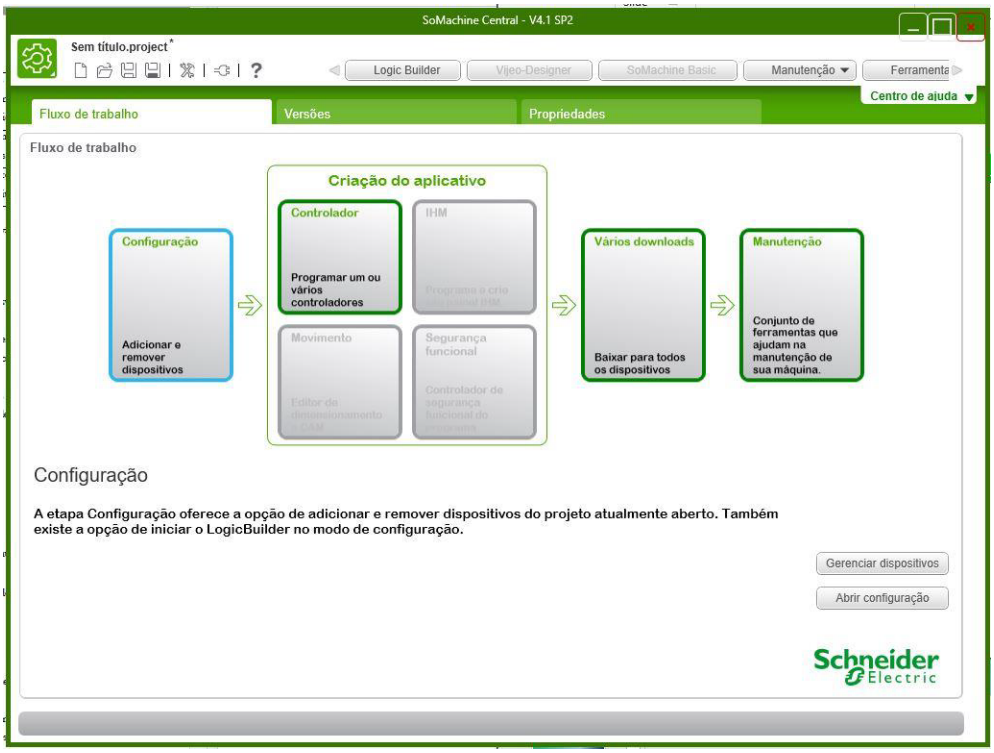


Figura 3.1: Interface do SoMachine V4.1.

O software SoMachine V4.1 fornece um mapa de *hardware* nativo para diversos controladores, incluindo o LMC058, e, também, as ferramentas do CoDeSys SoftMotion (aplicativo para desenvolvimento de aplicações no controlador) que serão utilizadas para a programação do movimento dos eixos. O SoftMotion é a estrutura da biblioteca para controle de movimento independente de funções multi-eixo. O SoftMotion inclui a biblioteca SM3_Basic, bem como funções de interface de acionamentos (*drives*), editores CNC, e gerenciamento de erros.

O software SoMachine V4.1 mantém grande parte da interface básica de um típico programa C, incluindo a declaração de variáveis, bibliotecas incluídas, compilação de programas e criação de um programa executável para ser carregado no dispositivo de destino.

Todas as bibliotecas nativas necessárias para gerenciar dispositivos do robô ou funcionalidades, são mantidos em um repositório. É possível adicionar bibliotecas específicas de aplicativos de modo simplificado. As bibliotecas incluídas estão localizadas no Gerenciador de Bibliotecas.

As aplicações no SoMachine V4.1 são criadas por meio das POUs ou Unidades Organizacionais de Programa. As POUs são as seções de código do programa de uma aplicação desenvolvida no SoMachineV4.1. As seções do programa são criadas usando o padrão IEC 6-1131 e linguagens de programação como Bloco de Funções (*Function Block Diagram*-FBD), Texto Estruturado (*Structured Text*), *Sequential Function Chart*, *Instruction List*, *Ladder* e *Continuous Function Chart*.

As POUs são alocadas dentro de (ou chamadas por) tarefas. Uma POU criada mas não alocada a uma tarefa não será executada no modo de execução do código no controlador.

Uma aplicação criada no SoMachine V4.1 faz uso de programação baseada em tarefas. Uma tarefa é configurada para ser executada ciclicamente de acordo com um dos seguintes procedimentos: (a) tempo de ciclo definido, (b) em resposta a um gatilho periódico; (c) uma entrada de evento ou de fora contínua de acordo com o ciclo de execução do controlador. Normalmente, todas as tarefas são configuradas como cíclicas para fornecer um tempo repetível base. Várias tarefas podem ser configuradas com seu próprio tempo de ciclo e prioridade de execução.

Neste trabalho, foi usado o SoMachine V4.1 para realizar a configuração da rede CANmotion, desenvolvimento do programa para movimento sincronizado dos eixos utilizando a biblioteca SM3_Basic, que faz parte das ferramentas do CoDeSys Softmotion, e desenvolvimento da aplicação do planejamento de trajetória do robô cartesiano, utilizando a biblioteca SM3_CNC.

3.2 Vijeo-Designer

O Vijeo Designer [28] [29], também da *Schneider Electric*, é um dos ambientes presentes no SoMachine. O Vijeo Designer é um software de programação destinado à IHM. Assim, o que for programado neste ambiente, poderá ser mostrado na tela da interface homem-máquina. A interface deste programa é mostrada na figura 3.2.

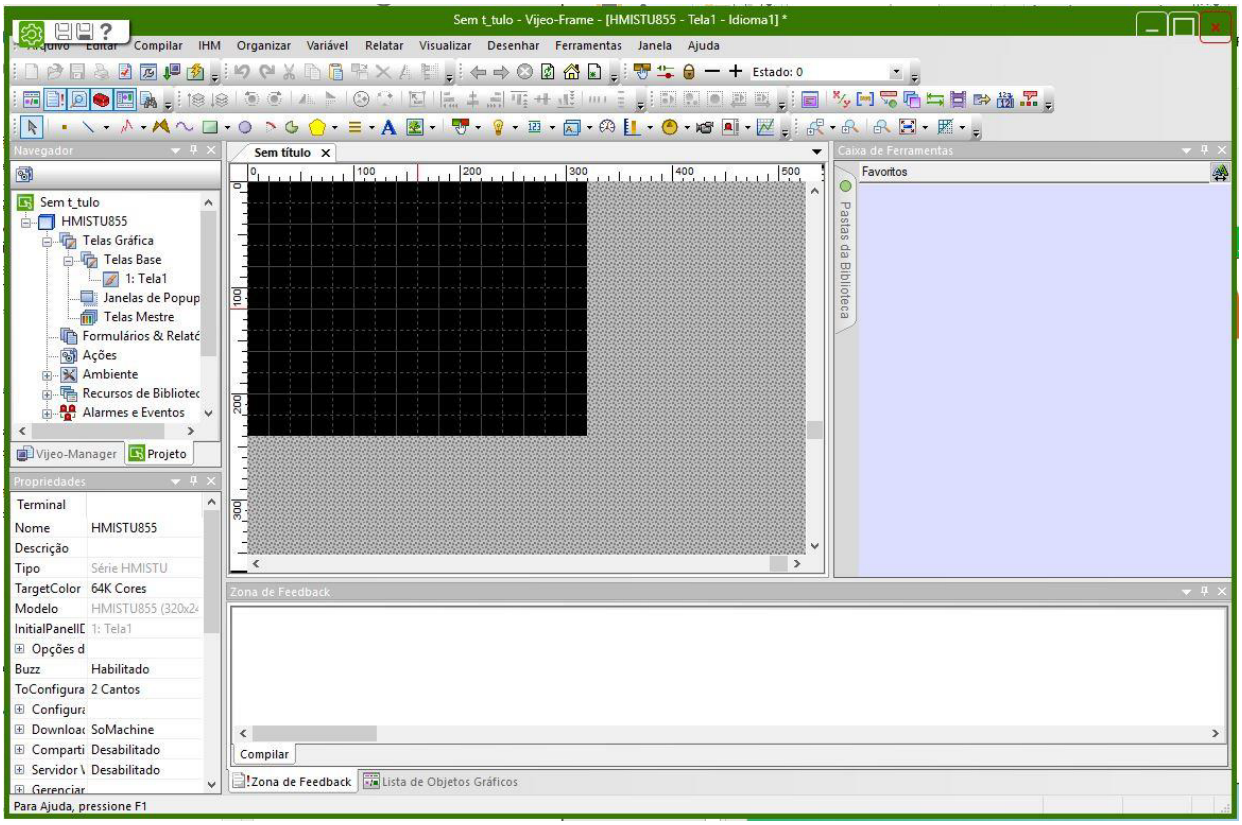


Figura 3.2: Interface do Vijeo Designer.

O ambiente de desenvolvimento de aplicações do Vijeo Designer oferecer um editor gráfico para o desenvolvimento de aplicações simples, bem como aplicações mais sofisticadas. O ambiente permite a criação de aplicações visuais com pontos, linhas, retângulos, elipses, arcos, gráficos de barra, medidores, tanques, enchimentos, gráficos de pizza, curvas, polilinhas, polígonos, polígonos regulares, imagens, alarmes, etc.

Além disso, a ambiente de desenvolvimento do Vijeo Designer possui 8 tipos de animação de objeto gráfico que suportam a criação rápida de animações básicas do tipo: pressionando o painel de toque, mudança de cor, recheio, movimento, rotação, tamanho, visibilidade, exibição de valores.

A biblioteca de objetos gráficos animados torna a criação de telas muito eficiente devido aos inúmeros objetos de animação "prontos". Essa biblioteca inclui mais de 4000 imagens “industriais” 2-D e 3-D. Simplesmente “arrastando e soltando” usando o mouse para posicionar os objetos gráficos na tela que está sendo desenvolvida. Objetos definidos pelo usuário podem ser adicionados a esta biblioteca usando o mesmo método de "drag and drop".

Neste trabalho o Vijeo-Designer foi utilizado para estabelecer a comunicação entre a IHM HMIS5T e o controlador LMC058LF42, além de desenvolver toda a interface de fácil uso para os operadores do robô cartesiano.

Capítulo 4

Integração dos Sistemas

4.1 Configuração da Rede CANMotion

A configuração da rede CANMotion é essencial para estabelecer a comunicação entre os *drives* Lexium 32M e o controlador LMC058. Essa configuração é feita por meio do SoMachine V4.1, o primeiro passo é criar um novo projeto escolhendo os dispositivos adequados do catálogo disponível. A figura 4.1 exibe os dois dispositivos que foram utilizados no projeto, o controlador LMC058LF42 e a interface homem máquina HMIS5T.

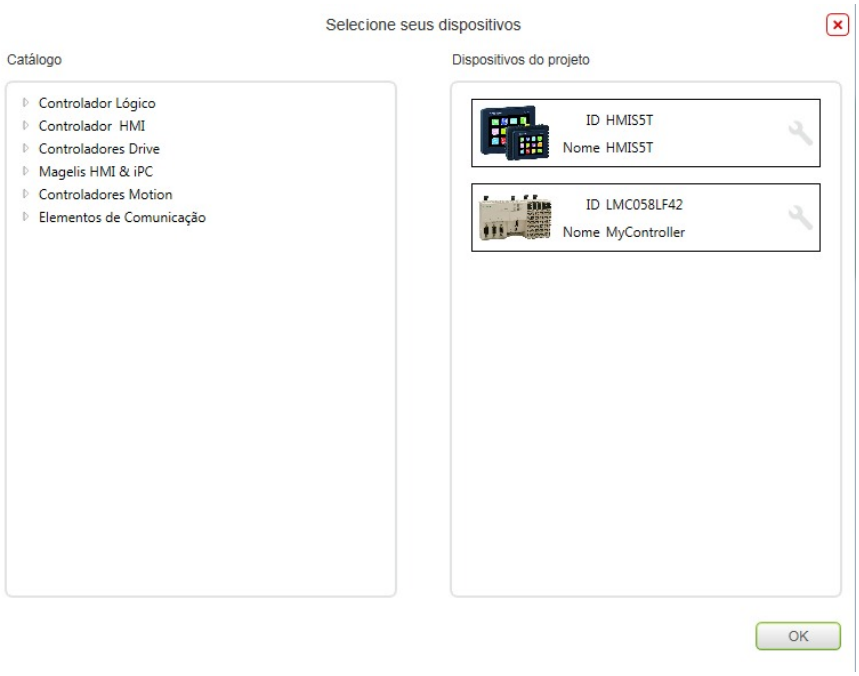


Figura 4.1: Adicionando Dispositivos em um Novo Projeto no SoMachine.

Após a escolha dos dispositivos adequados para a aplicação é necessário acessar dentro do novo projeto a aba de *Dispositivos* que se encontra em *Ver -> Navegadores -> Dispositivos*. Acessando essa aba, o usuário tem acesso a todos os *hardwares* mapeados para o projeto. A figura 4.2 mostra

os dispositivos que foram selecionados do catálogo.

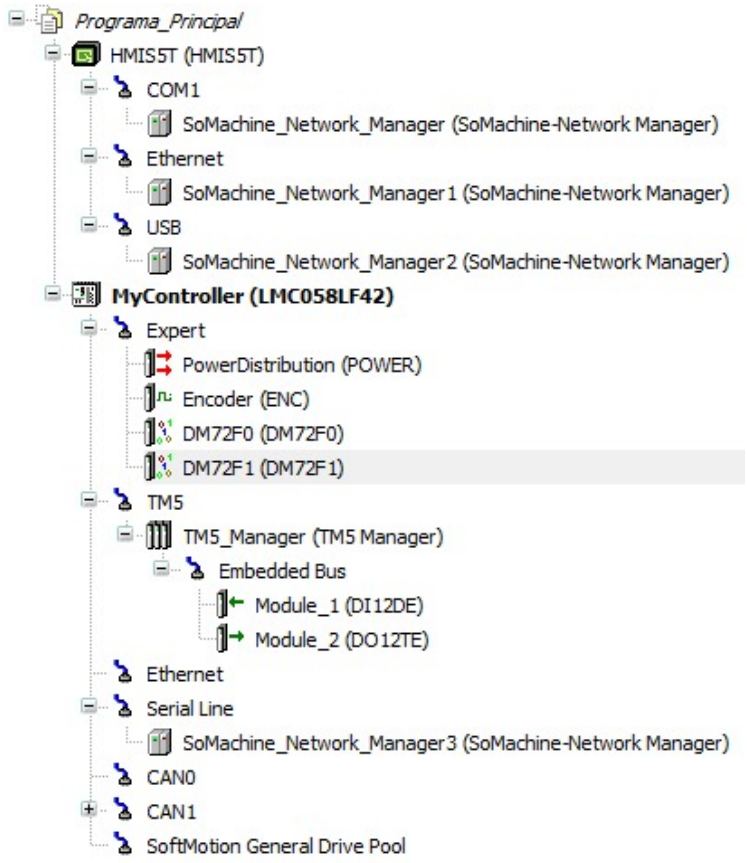


Figura 4.2: Aba dos Dispositivos no ambiente do SoMachine.

Em seguida, adicionam-se os *servo-drives* Lexium 32M no projeto estabelecendo a comunicação entre os *drives* e o controlador. No controlador LMC058, a porta CANMotion é a CAN 1 (4.3), portanto com os *drives* serão adicionados na porta CAN 1 dentro da aba de dispositivos.

Para adicionar os *drives* é preciso acessar a aba de *Dispositivos e Módulos* que se encontra em *Ver -> Catálogo de hardware -> Dispositivos e Módulos*. Dentro da aba de *Dispositivos e Módulos* existe o campo de *Motion Control* (4.4). Expandindo esse campo, é possível acessar os servos Lexium 32M. O SoMachine V4.1 possui um sistema de *drag-and-drop* o que facilita colocar os *drives* dentro da porta CAN.

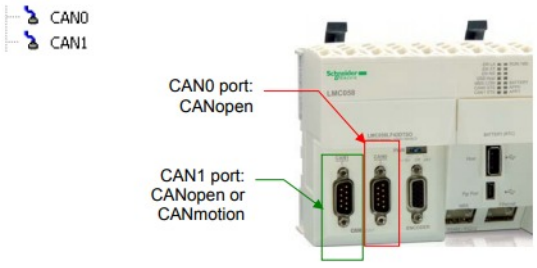


Figura 4.3: Visualização da Porta CAN 1 no Controlador LMC058.

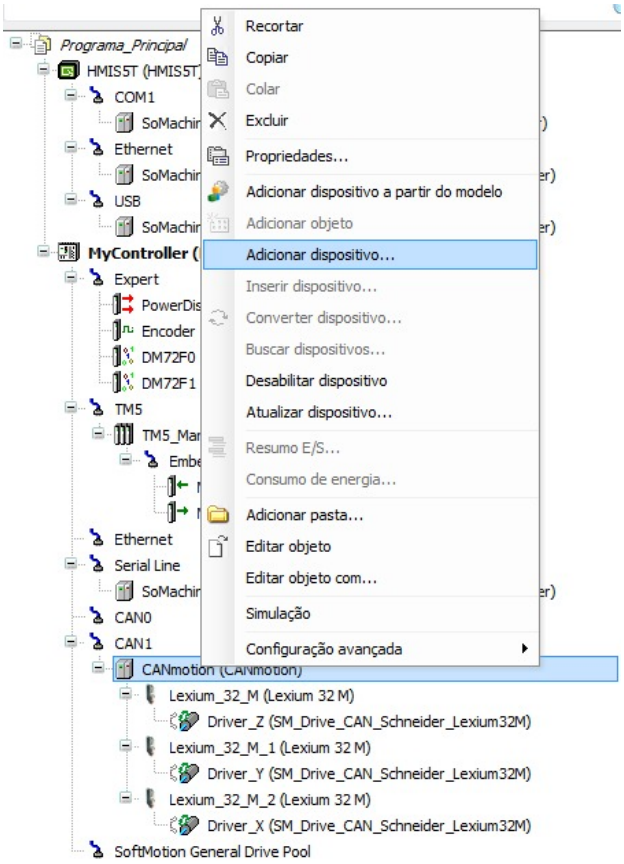


Figura 4.4: Adicionando os Drives Lexium 32M na porta CAN1.

Adicionados os *drives*, é preciso definir alguns parâmetros. Acessando a porta CAN, na aba de dispositivos do SoMachine V4.1, tem-se que definir sua taxa de transmissão de dados em bits/s. No programa desenvolvido, foi utilizada a taxa de transmissão padrão de 1.000.000 bits/s (1 Mbps) (vide figura 4.5).

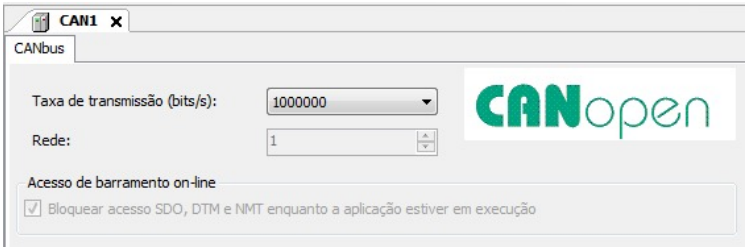


Figura 4.5: Definindo a Taxa de Transmissão da Porta CAN1.

Após estabelecer a taxa de transmissão de dados da porta CAN, é preciso definir o período de ciclo do gerenciador CANMotion.

Tabela 4.1: Tempo de transferência de dados.[10]

Modo CANMotion	250 Kbits/s	500 Kbits/s	1 Mbits/s
Tráfico Padrão	1.01 ms	0.66 ms	0.48 ms
Drive no modo CSP	0.92 ms/drive	0.46 ms/drive	0.23 ms/drive
Drive no modo CST	1.89 ms/drive	0.95 ms/drive	0.47 ms/drive
Drive no modo CSV	1.89 ms/drive	0.95 ms/drive	0.47 ms/drive
Opcional TPDO Header	0.19 ms/drive	0.10 ms/drive	0.05 ms/drive
Opcional TPDO por Byte	0.04 byte/TPDO	0.02 byte/TPDO	0.01 byte/TPDO

A tabela 4.1 mostra os tempos utilizados pelo CANMotion para a realização das tarefas básicas e tarefas de acionamento de eixos sincronizados de acordo com a taxa de transmissão e do modo de controle de movimento configurado nos *drives*. De acordo com o guia de programação do LMC058[10] esse tempo é definido como sendo o maior dentre os tempos de execução da tarefa programada no controlador e a soma dos intervalos de tempo correspondente às tarefas de comunicação necessárias ao controle do movimento sincronizado de vários eixos, sendo dependente da taxa de transmissão configurada anteriormente, do número de *servo-drives* e dos respectivos modos de funcionamento. Com base na tabela, na taxa de transmissão escolhida (1 Mbps) e nos equipamentos instalados e seus respectivos modos de funcionamento (CSP- *Cyclic Synchronous Position*) o tempo mínimo de ciclo seria igual a 1.17 ms multiplicados por 1,2 (20 por cento de folga), o que resulta no valor de 1,404 ms. Entretanto, como não se conseguiu medir o tempo de execução das tarefas programadas no controlador e considerando que, em trabalho anterior [5], reportaram-se problemas de sincronismo com a utilização do tempo acima calculado, optou-se por utilizar o valor de 4 ms (conforme mostra a figura 4.6). Esse período não apresentou nenhuma falha devida a perda de sincronismo, o que indica ser esse tempo suficiente para realização de todas as tarefas programadas e para a troca de mensagens na rede CANMotion. Em se conseguindo medir o tempo de execução da tarefa, é possível reduzir esse tempo de ciclo da rede caso se deseje uma taxa de atualização mais alta.

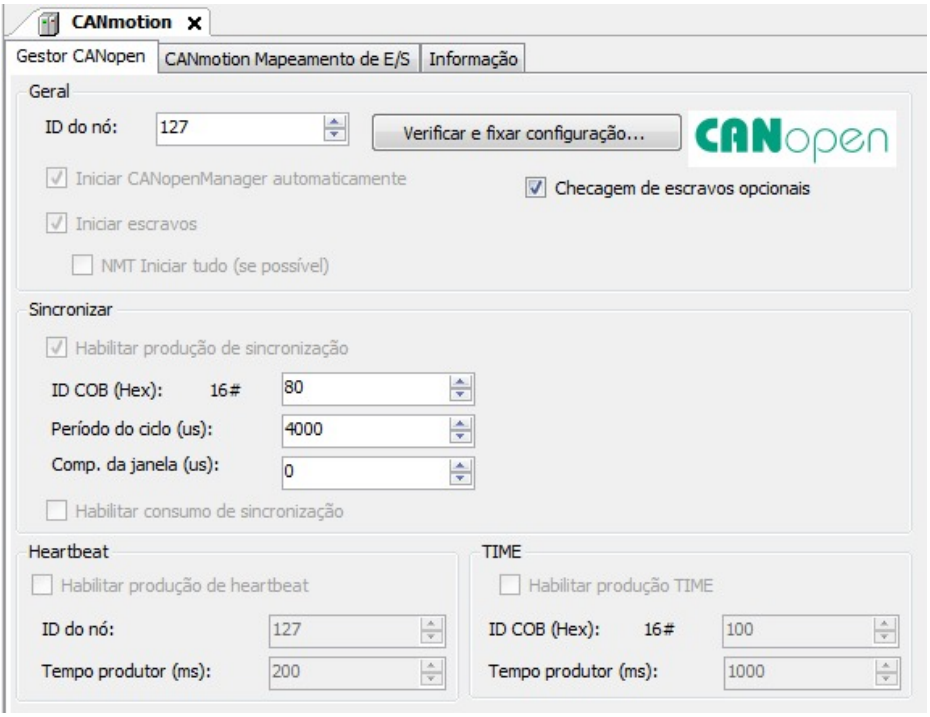


Figura 4.6: Definindo o Período de Ciclo do Gerenciador CANMotion.

Com os passos descritos acima, deve-se estabelecer a configuração dos dispositivos. Nessa etapa, é necessário escolher um número de nó para identificação de cada um dos *servo-drives* na rede CANMotion. Definiu-se que o endereço de nó 1 será referente ao *servo-drive* Z, o endereço de nó 2 será referente ao *servo-drive* Y e o endereço de nó 3 será referente ao *servo-drive* X. Na figura 4.7 é mostrada a configuração da identificação (ID) do nó referente ao *servo-drive* Z.

Figura 4.7: Estabelecendo o Endereço de Nó para cada um dos Drive.

O último passo é a configuração dos eixos. Esta etapa começa pela definição dos três eixos como sendo do tipo "finito", ou seja há limitações fisicas no movimento que cada eixo pode realizar. É possível definir os limites de movimento via software para cada um dos eixos do robô, entretanto, essa limitação será feita em uma etapa posterior.

Nessa etapa define-se para cada eixo o tipo de curva de velocidade (vide figura 4.8). A aplicação de soldagem 3D requer um movimento suave de baixa velocidade. A curva escolhida foi a Sin^2, que garante um movimento suave para os eixos, com valor limitado na derivada da aceleração.A figura 4.9 mostra o perfil de velocidade na curva, o perfil de aceleração e o perfil de posição da curva Sin^2.

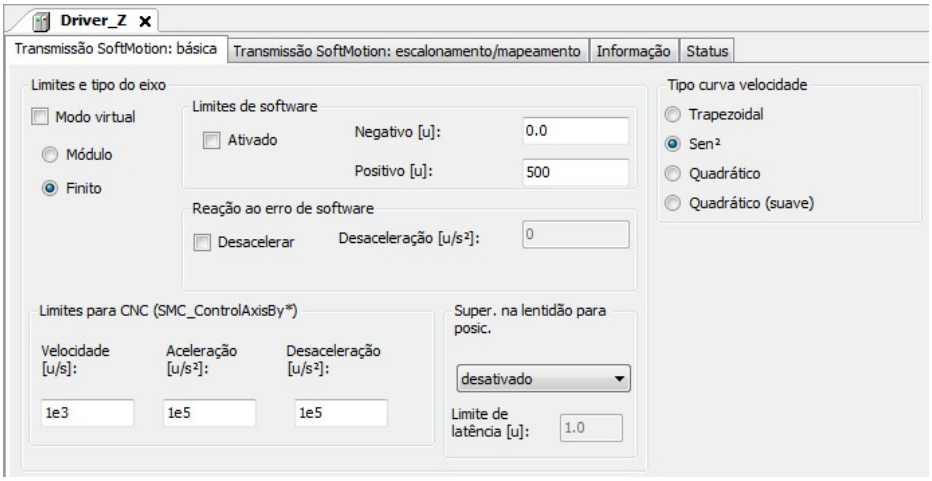


Figura 4.8: Etapa de Configuração dos Eixos.

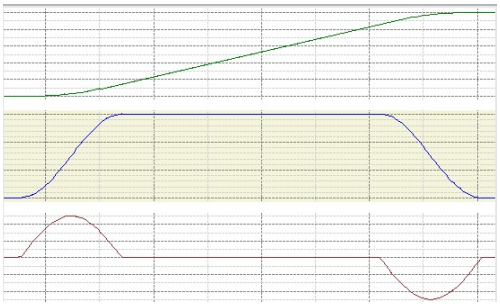


Figura 4.9: Curvas de Deslocamento (Verde), Velocidade(Azul) e Aceleração (Vermelho), em função do tempo para a opção de curva de velocidade do tipo Sin^2.

Ainda na etapa de configuração do eixo, agora é necessário definir o fator de escala dos eixos, conforme mostrado na figura 4.10. Na primeira linha na figura 4.10 é usada para definir o número de incrementos do *servo-drive* que corresponde a uma volta completa do motor. A segunda linha na mesma figura refere-se à relação de redução do redutor planetário acoplado em cada um dos servo motores. A última linha relaciona a distância percorrida no eixo translacional para cada volta completa do eixo de saída do redutor.

O redutor planetário do robô cartesiano possui uma relação de redução de 8:1. O número padrão de pulsos por volta do motor, definido no *servo-drive*, é de 131.072 (em hexadecimal, 20.000). . A relação entre a rotação dos redutores e a distância correspondente no eixo translacional é de 155 mm/volta [8]. A configuração final do escalonamento dos eixos pode ser vista na figura 4.10

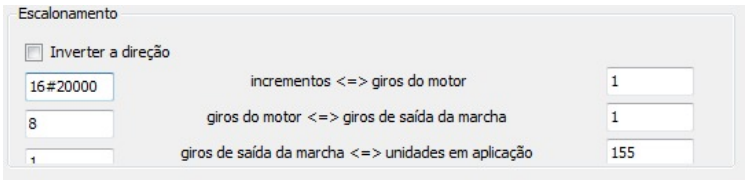


Figura 4.10: Definição da Taxa de Escalonamento para os Eixos.

Realizando todas as etapas citadas a rede CANMotion vai estar estabelecida, permitindo a comunicação entre os *servo-drives* Lexium 32M e o controlador LMC058.

4.2 Estabelecendo a Comunicação com a IHM

A configuração da IHM HMIS5T teve alguns problemas, pois diferente da configuração da rede CANMotion, não foi possível encontrar um manual explicando como configurar a interface. O primeiro passo da configuração da IHM é acessar o Vijeo-Designer. Na tela inicial do Vijeo-Designer (vide a figura 4.11) tem-se o dispositivo que foi selecionado na criação do projeto. Conforme a figura 4.1 o modelo da IHM escolhido foi o HMIS5T.

No lado esquerdo da tela inicial na aba *Navegador* (vide figura 4.11) está o dispositivo que foi escolhido na criação do projeto, acessando a tela "Geral"na aba de propriedades do HMIS5T (lado direito da figura 4.11), pode-se definir uma nova configuração de rede. No campo "Endereço de IP do Alvo", deve-se colocar o endereço IP desejado ao IHM. Esse endereço deve utilizar a mesma sub-rede definida para o controlador.

Na aba "Rede"(mostrado na figura 4.12) dentro das propriedades da IHM, é necessário estabelecer a configuração da rede Ethernet. O endereço IP escolhido para estabelecer a comunicação entre a IHM e o controlador LMC058 foi o 10.10.159.2. Este se encontra dentro da uma sub-rede (10.10.159...) pré-definida no controlador LMC058, cujo o endereço de fábrica é o 10.10.159.153. Para a aplicação em questão, esse endereço foi modificado para 10.10.159.4. Além do IP dos diversos nós, outros parâmetros da rede foram também configurados, tais como máscara de sub-rede (255.0.0.0) e gateway (10.10.159.153). O computador do operador apresenta duas portas ethernet, sendo uma delas configurada na mesma sub-rede do controlador, com endereço 10.10.159.1, e a outra porta, configurada para acesso a internet.

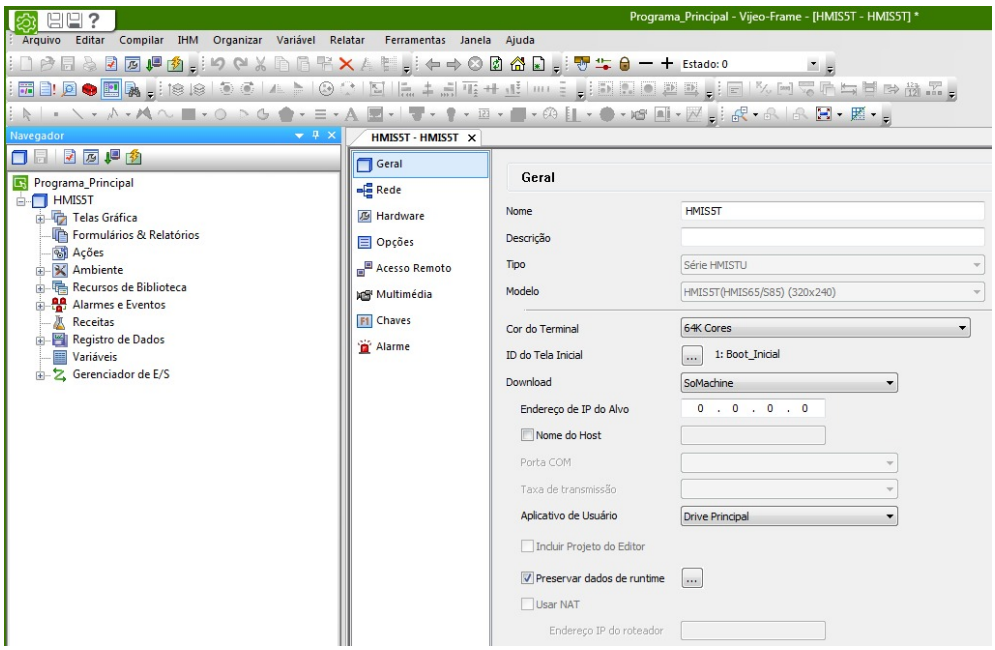


Figura 4.11: Tela Inicial do Software Vijeo-Designer.

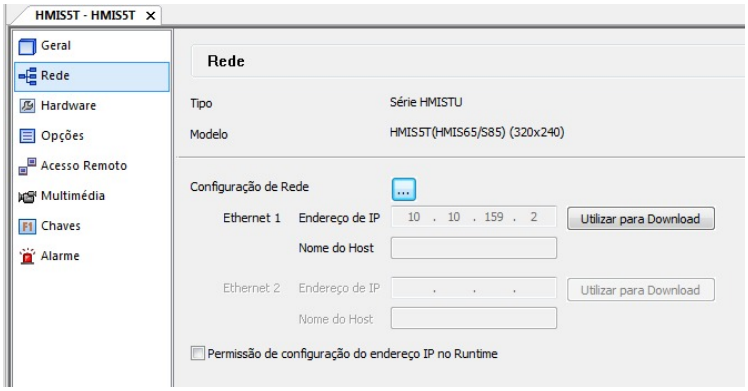


Figura 4.12: Configuração de Rede da IHM.

Após realizar a configuração da rede ethernet é necessário acessar as propriedades do gerenciador de entrada e saída na aba *Navegador* conforme a figura 4.13. Dentro da aba de *Gerenciador de E/S*, é necessário verificar se ambos os *hardwares* estão mapeados, o controlador LMC058 e a IHM HMIS5T. Acessando o dispositivo *SOM_HMIS5T* deve-se verificar se o endereço é o mesmo escolhido na aba de *Rede* do Vijeo-Designer (endereço IP: 10.10.159.2), após isso é preciso acessar a aba *SOM_MyController* para verificar se o endereço IP esta idêntico ao seu endereço (endereço IP: 10.10.159.4).

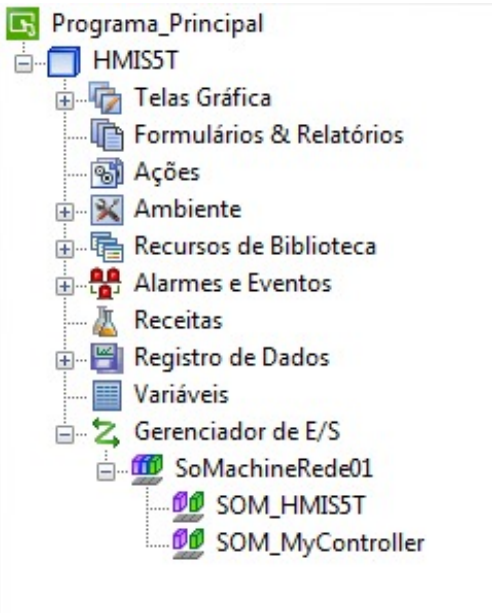


Figura 4.13: Estabelecendo a Comunicação no Gerenciador de Entrada/Saída.

4.3 Movimento Sincronizado dos Eixos

O programa desenvolvido ao longo deste trabalho de graduação foi feito utilizando blocos de funções que estão disponíveis nas bibliotecas SM3_Basic e SM3_CNC do software SoMachineV4.1 da *Schneider Eletric*. As bibliotecas SM3_Basic e SM3_CNC rodam sobre o *engine* do CoDeSys Softmotion, o Softmotion é responsável pelo controle de movimento dos eixos no ambiente SoMachine. A figura 4.14 mostra as aplicações para cada uma das bibliotecas SM3 do *engine* do CoDeSys Softmotion.

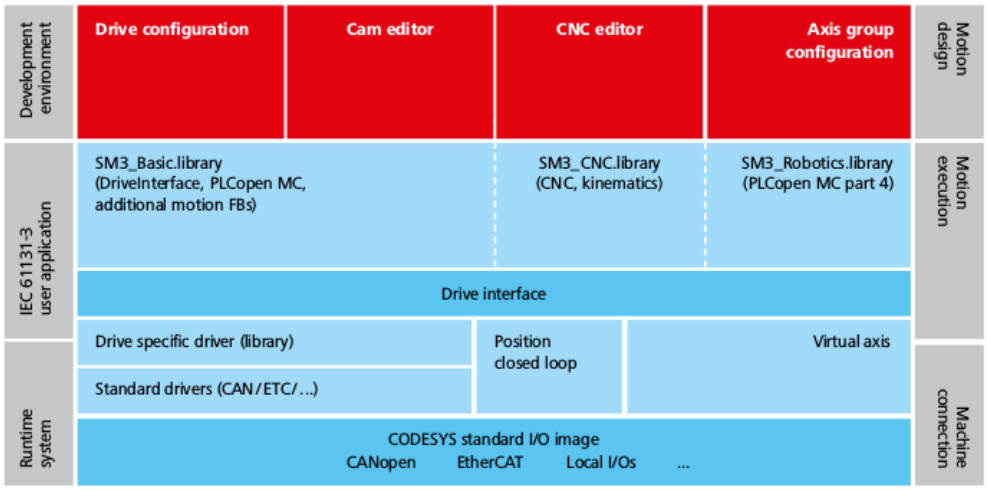


Figura 4.14: Aplicações das Bibliotecas SM3 do CoDeSys Softmotion. [31]

O programa de movimentação dos eixos, o "Motion", realiza o movimento sincronizado dos

eixos do robô cartesiano. O "Motion"foi todo desenvolvido utilizando a biblioteca SM3_Basic do SoMachine.

A figura 4.15 mostra o diagrama dos oito possíveis estados em que o *drive* da Lexium 32M pode estar. A figura também ilustra qual ação leva a troca de estado. O diagrama auxiliou no desenvolvimento do programa, uma vez que facilitou a identificação dos blocos de funções necessários para desenvolver uma aplicação completa.

O estado de "StandStill"é o único estado no diagrama que possibilita iniciar um dos tipos de movimento, *Discrete Motion*, *Continuous Motion* e *Synchronized Motion*. Portanto é necessário garantir que o *drive* sempre esteja nele para iniciar o movimento, além disso, também é necessário evitar o estado de "ErrorStop"em que o *drive* só consegue sair se for voltando para o "StandStill".

A biblioteca SM3_Basic funciona em uma dinâmica de *"confirm...then command"*, que significa que o *drive* primeiro confirma em quais dos estados da figura 4.15 ele se encontra e somente após a confirmação ocorre a mudança de estado.

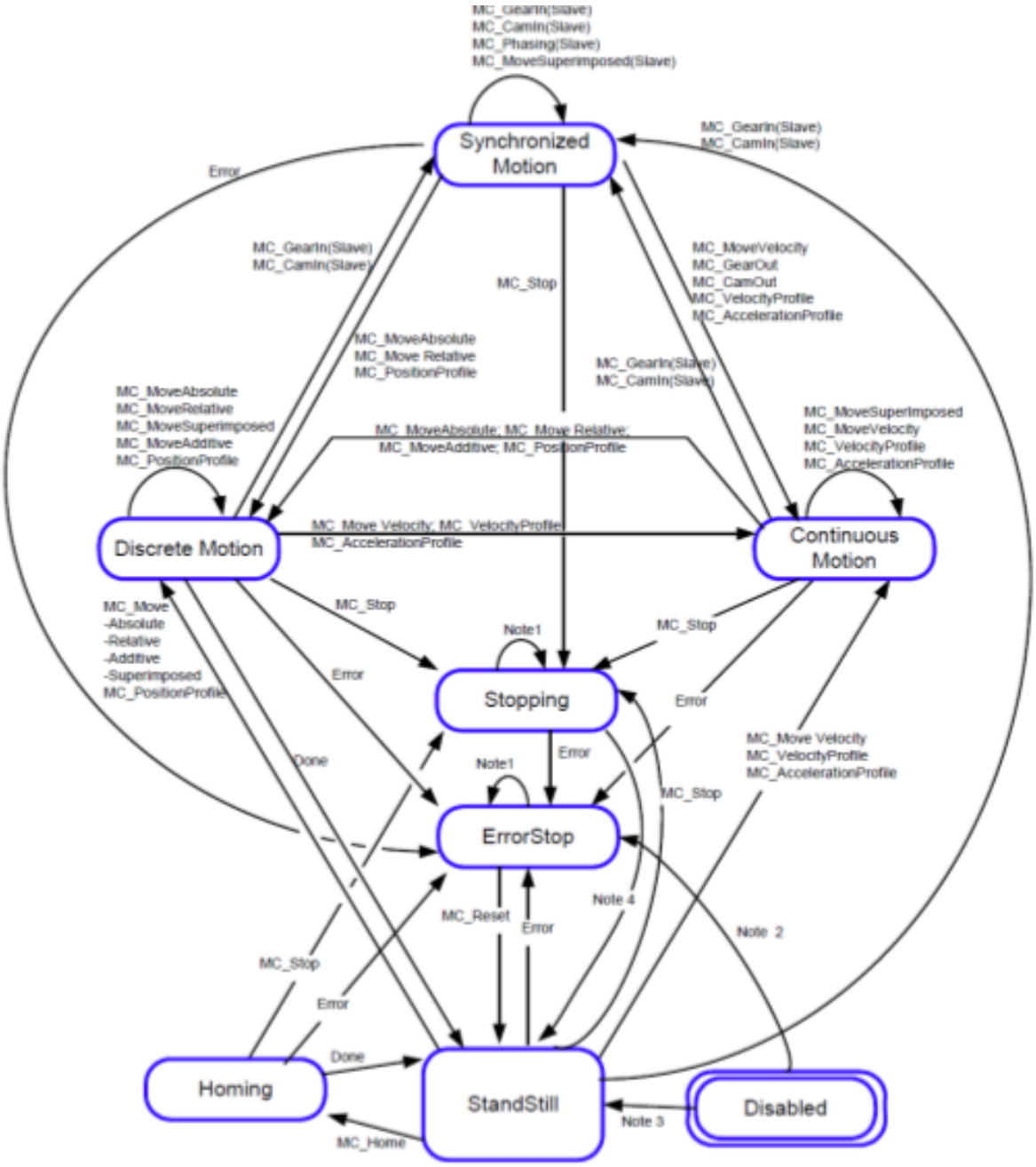


Figura 4.15: Diagrama de Estados dos Drives. [27]

O programa principal desenvolvido neste trabalho é mostrado no apêndice A.

4.4 Biblioteca SM3_Basic

Os blocos de função disponíveis na bilioteca SM3_Basic utilizados no programa são requisitos de qualquer aplicação simples de movimentação utilizando os *drives* Lexium 32M.

O primeiro bloco de função utilizado no programa foi o MC_ReadStatus, esse bloco tem a função de monitorar o estado que o *drive* de um determinado eixo se encontra.

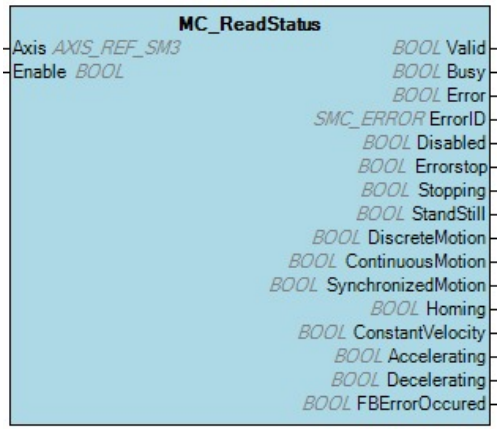


Figura 4.16: Bloco de Função do MC_ReadStatus.[30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  Enable: Quando essa entrada BOOLEANA é ativada o bloco inicia o processo contínuo de monitoramento do *drive*;

- Saídas:

  Valid: Se essa saída BOOLEANA estiver verdadeira, os parâmetros de saída estão disponíveis para leitura;

  Busy: Se verdadeira, o bloco ainda está executando a sua função de monitoramento;

  Error: Se verdadeira, sinaliza que um erro ocorreu com esse bloco de função;

  ErrorID: Identifica qual o tipo de erro ocorreu;

  Disabled: Se verdadeira, o eixo se encontra no estado de "Disabled";

  ErrorStop: Se verdadeira, o eixo se encontra no estado de "ErrorStop";

  Stopping: Se verdadeira, o eixo se encontra no estado de "Stopping";

  StandStill: Se verdadeira, o eixo se encontra no estado de "StandStill";

  DiscreteMotion: Se verdadeira, o eixo se encontra no estado de "DiscreteMotion";

  ContinuousMotion: Se verdadeira, o eixo se encontra no estado de "ContinuousMotion";

  SynchronizedMotion: Se verdadeira, o eixo se encontra no estado de "SynchronizedMotion";

  Homing: Se verdadeira, o eixo se encontra no estado "Homing";

  ConstantVelocity: Se verdadeira, o motor está se movimentando com velocidade constante;

  Accelerating: Se verdadeira, a tensão do motor está aumentando;

  Decelerating: Se verdadeira, a tensão do motor está diminuindo;

  FBErrorOccured: Se verdadeira, se ocorreu um erro em um bloco de funções e o problema ainda não foi resolvido.

Em seguida foi utilizado o bloco de função MC_Power. Esse bloco possibilita a energização dos *drives* de cada um dos eixos do robô cartesiano. O bloco de função MC_Power é essencial para a movimentação dos eixos.

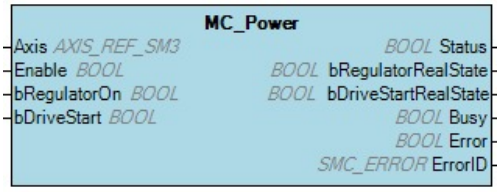


Figura 4.17: Bloco de Função do MC_Power. [30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  Enable: Quando essa entrada BOOLEANA é ativada o bloco de função é ativado;

  bRegulatorOn: Se essa entrada for verdadeira ocorre a energização do *drive* do eixo;

  bDriveStart: Se essa entrada for verdadeira o mecanismo de freio dos motores é desativado;

- Saídas:

  Status: Se verdadeira, o eixo está pronto para movimento;

  bRegulatorRealState: Se verdadeira, o eixo se encontra ativado;

  bDriveStartRealState: Se verdadeiro, o mecanismo de freio do eixo não está ativado;

  Busy: Se verdadeiro, o bloco de função ainda está em execução;

  Error: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  ErrorID: Identifica qual tipo de erro ocorreu;

O próximo bloco de função utilizado no desenvolvimento do programa foi o bloco MC_Home. Esse bloco retorna os eixos para a sua posição inicial na coordenada (0,0,0).

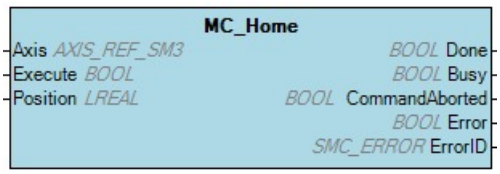


Figura 4.18: Bloco de Função do MC_Home. [30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  Execute: Se verdadeiro, o bloco inicia o processo de *Homing*;

Position: Seta a posição. No caso, é utilizado o valor zero para que, quando o drive atinja a posição inicial, esta seja a posição zero.

- Saídas:

  Done: Se verdadeiro, o eixo se encontra no estado de "StandStill";

  Busy: Se verdadeiro, o bloco ainda está executando a sua função;

  CommandAborted: Se verdadeiro, o comando de *homing* foi abortado por outro comando externo.

  Error: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  ErrorID: Identifica qual tipo de erro ocorreu;

O bloco de função MC_Reset, tem função de mudar o eixo do estado de *ErrorStop* para o estado *StandStill* de acordo com a figura 4.15.

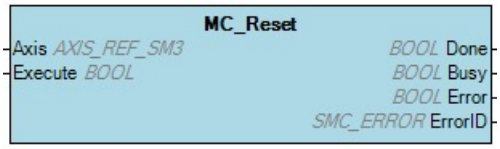


Figura 4.19: Bloco de Função do MC_Reset. [30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  Execute: Se verdadeiro, o bloco inicia o processo de mudar o estado em que o eixo se encontra;

- Saídas:

  Done: Se verdadeiro, o reset para o estado "StandStill"foi realizado;

  Busy: Se verdadeiro, o bloco ainda está executando a sua função;

  Error: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  ErrorID: Identifica qual tipo de erro ocorreu;

O bloco MC_Stop é utilizado como uma parada de emergência via software, esse bloco força o eixo a entrar no estado de "Stopping"causando o eixo a parar o movimento. O eixo permanece no estado de "Stopping"enquanto este bloco estiver ativado.

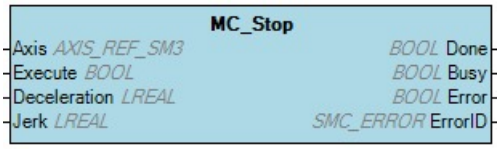


Figura 4.20: Bloco de Função do MC_Stop. [30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  Execute: Se verdadeiro, o bloco muda o estado atual do eixo para o estado "Stopping";

  Deceleration: O valor da desaceleração do eixo, como esse bloco irá atuar como uma parada de emergência via software a desaceleração escolhida foi de 5 m/s^2, esse valor foi escolhido de modo a estar abaixo do limite máximo de 7.03 m/s^2 (vide tabela 2.1);

- Saídas:

  Done: Se verdadeiro, o bloco muda o estado atual para o estado de "Stopping";

  Busy: Se verdadeiro, o bloco ainda está executando a sua função;

  Error: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  ErrorID: Identifica qual tipo de erro ocorreu;

O bloco de função MC_MoveAbsolute comanda o movimento do eixo em relação a uma coordenada absoluta.

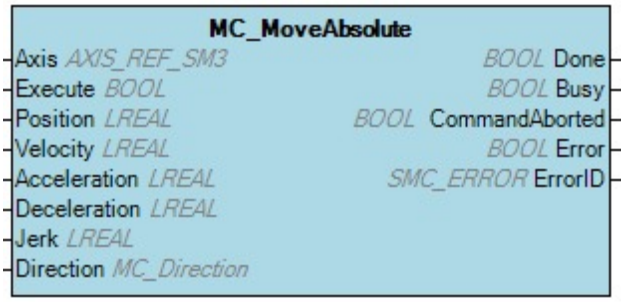


Figura 4.21: Bloco de Função do MC_MoveAbsolute. [30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  Execute: Se verdadeira, inicia a execução do bloco de função;

  Position: Posição em coordenadas absolutas para onde deseja movimentar o eixo.

  Velocity: Valor da velocidade máxima para o deslocamento do eixo, a velocidade inserida nesse campo não necessariamente será atingida. Como esse bloco não será utilizado no processo de soldagem, a velocidade máxima definida foi de 1000 mm/s;

  Acceleration: Valor de aceleração do eixo, pelo mesmo fato citado acima que esse bloco não será utilizado no processo de soldagem 3D, a aceleração inserida nesse bloco foi de 500 mm/s^2;

  Deceleration: Valor da desaceleração do eixo, assim como para a aceleração a desaceleração será de 500 mm/s^2;

Direction: Escolha da direção a ser tomada no acionamento do bloco de função, para a aplicação do movimento dos eixos a escolha foi o modo padrão do bloco que é o "shortest";

- Saídas:

  Done: Se verdadeiro, o eixo se encontra na posição desejada;

  Busy: Se verdadeiro, o bloco ainda está executando a sua função;

  CommandAborted: Se verdadeiro, o comando foi abortado por outro comando externo;

  Error: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  ErrorID: Identifica qual tipo de erro ocorreu;

O bloco MC_SetPosition, é fundamental na aplicação de soldagem 3D pois com esse bloco é possível zerar as coordenadas dos eixos, o zeramento das coordenadas é uma parte essencial quando há aplicações de código G.

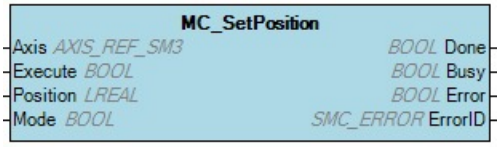


Figura 4.22: Bloco de Função do MC_SetPosition. [30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  Execute: Se verdadeiro, inicia a execução do bloco de função;

  Position: Posição que se deseja definir que o eixo se encontra. Essa entrada na aplicação de Soldagem 3D, será definida como 0, pois quando um novo código G é carregado se deseja fazer o deslocamento do eixo em relação a origem;

  Mode: Se verdadeira = Coordenadas Relativas, Se falso = Coordenadas Absolutas. Para a aplicação de Soldagem 3D as coordenadas serão as absolutas.

- Saídas:

  Done: Se verdadeiro, o eixo foi designado uma nova posição;

  Busy: Se verdadeiro, o bloco ainda está executando a sua função;

  Error: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  ErrorID: Identifica qual tipo de erro ocorreu;

O bloco MC_Jog, é utilizado no programa para fazer o deslocamento lento dos eixos para um posicionamento com maior precisão, portanto a velocidade e aceleração deste bloco são baixas.

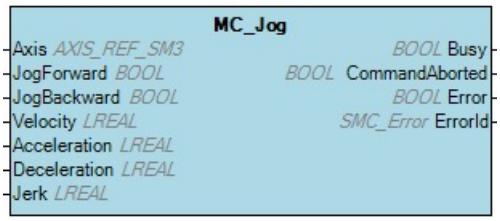


Figura 4.23: Bloco de Função do MC_Jog. [30]

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  JogForward: Se verdadeiro, o eixo se desloca com a velocidade, aceleração e desaceleração definidas no bloco na direção positiva;

  JogBackward: Se verdadeiro, o eixo se desloca com a velocidade, aceleração e desaceleração definidas no bloco na direção negativa;

  Velocity: Definição da velocidade do movimento de Jog. Para o posicionamento da tocha de soldagem em uma posição especifica a velocidade definida nessa entrada foi de 20 mm/s;

  Acceleration: Definição da aceleração do bloco. Para o processo de posicionamento da tocha de soldagem em uma posição especifica a aceleração foi definida como 10 mm/s^2;

  Deceleration: Definição da desaceleração do bloco. Semelhante a aceleração a desaceleração foi definida como 10 mm/s^2;

- Saídas:

  Busy: Se verdadeiro, o bloco ainda está executando a sua função;

  CommandAborted: Se verdadeiro, o comando foi abortado por outro comando externo;

  Error: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  ErrorID: Identifica qual tipo de erro ocorreu;

Capítulo 5

Planejamento de Trajetória

5.1 Programa de Interpolação de Pontos

O programa de interpolação de pontos foi desenvolvido utilizando a biblioteca SM3_CNC do CoDeSys Softmotion, seguindo as etapas ilustradas no fluxograma 5.1.

Analisando o fluxograma (vide figura 5.1), é possível visualizar como é feito o processo que se inicia com a interpretação de informações de um sistema de coordenadas imaginárias, a transformação desses dados por meio da cinemática inversa para o *drive* de interface e finalmente a transformação para coordenadas diretas para o movimento do eixo.

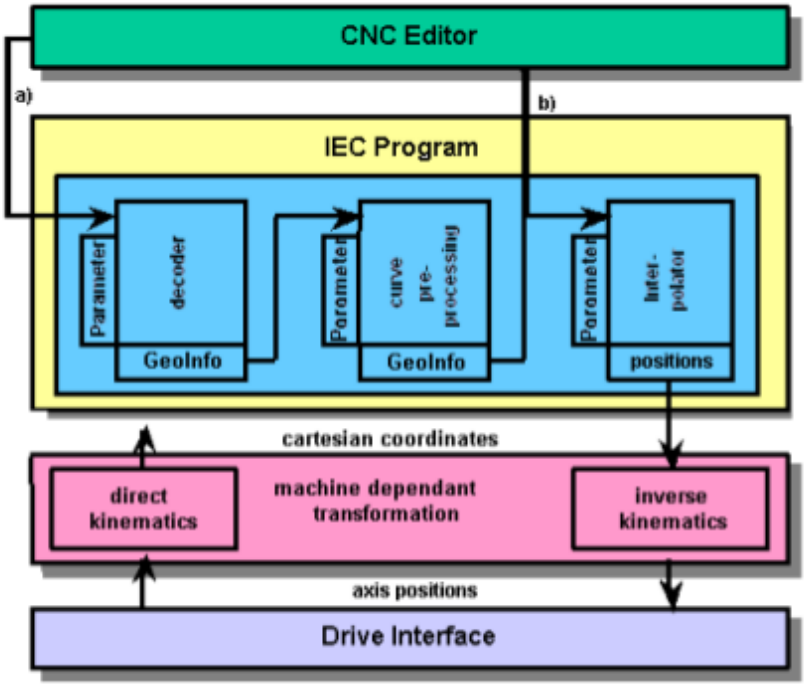


Figura 5.1: Fluxograma do Processo de Leitura, Pré-processamento e Interpolação de Pontos em um Programa na Linguagem G. [30]

O fluxograma exibe de modo simplificado a etapa de decodificação que é composta pela leitura e decodificação de um arquivo com a extensão (.txt) contendo um código G. Primeiro, a informação é lida pelo bloco de leitura *SMC_ ReadNCFile* que tem a função de fazer a leitura do arquivo ASCII em linguagem G. Posteriormente esses dados são transmitidos para o decodificador. O bloco de decodificação *SMC_ NCDecoder* atua como um tradutor de ASCII para dados do tipo *Softmotion-GEOINFO-Structure*.

O dado do tipo *GEOINFO* é um formato padrão das aplicações desenvolvidas utilizando a biblioteca SM3_CNC. Com a execução desses dois blocos é concluída a primeira etapa do fluxograma em seguida é feita a parte de pré-processamento dos dados.

O pré-processamento é uma etapa de segurança, pois por meio de blocos de função é possível adicionar condições ao sistema de modo a permitir que o robô atue dentro dos limites de segurança para não sobrecarregar os atuadores.

No programa foram implementadas 4 medidas de segurança que devem ser seguidas durante o desenvolvimento do código G, a limitação da velocidade circular, arredondamento de caminhos com ângulos muito agudos, rotina para eliminação de sobreposição de pontos e limitação das velocidades de cada eixo.

A limitação das velocidades dos eixos e da velocidade circular são medidas para preventivas para que o código G não sujeite os atuadores do robô a uma velocidade elevada de modo a danificar o equipamento. Para a aplicação de soldagem em camadas sucessivas, o valor definido para a aceleração máxima foi de 7.03 m/s^2, em que g é a variável que representa a constante gravitacional.

O arrendondamento de caminhos com ângulos muito agudos é feito para evitar que durante o processo de deposição de metal haja um caminho em que o atuador tenha que alterar o seu sentido de movimento de maneira brusca. Para essa medida de segurança o valor de ângulo mínimo que pode ser atingido em um caminho é definido de acordo com as configurações padrão de uma aplicação desenvolvida utilizando a biblioteca SM3_CNC.

O processamento dos dados para a eliminação de sobreposição é essencial pois para um processo de fabricação por camadas sucessivas, não pode haver sobreposição de metal indesejado, pois essa falha pode causar assimetria na peça.

Após o pré-processamento dos dados vem a etapa de interpolação do conjunto de pontos que compõem o caminho de fabricação da peça, a interpolação dos pontos e a transformação por cinemática inversa é o que torna possível o movimento dos eixos. Para essa etapa, é importante ressaltar que no processo de soldagem a velocidade permanece constante, ocorre somente a interpolação linear ou circular entre os pontos que formam o caminho. Portanto para a aplicação de soldagem a aceleração e desaceleração máxima foi limitada a 7.03 m/s^2 e defnida como 5 m/s^2 e a velocidade que cada eixo irá se movimentar é de no máximo 200 mm/s.

A interpolação dos pontos permite criar um caminho que por meio de cinemática inversa será transmitido para os *drives* do robô cartesiano que realiza o movimento. Essas são as etapas desenvolvidas para obter o planejador de trajetórias do robô cartesiano.

5.2 Biblioteca SM3_CNC

O planejamento de trajetória do robô cartesiano foi a última etapa desenvolvida neste trabalho de graduação, essa etapa consiste na programação do controlador LMC058 para interpretar pontos em um sistema de referência imaginário para coordenadas cartesianas do robô MAXR23-S42-H42-C42.

A CoDeSys Softmotion que é o *engine* utilzado pelo SoMachine V4.1 para executar o movimento dos eixos do robô cartesiano, possui uma biblioteca específica para o desenvolvimento de soluções de código G DIN66025. Optando pela facilidade e praticidade dos blocos de função pré-programados, foi escolhido o código G para realizar a interpretação dos pontos de um sistema de referência imaginário para o sistema de coordenadas do robô cartesiano.

A biblioteca SM3_CNC é parte do CoDeSys Softmotion, essa biblioteca possui todos os blocos de funções necessários para desenvolver as etapas de leitura, pré-processamento e interpolação de um arquivo desenvolvido em código G para o sistema de coordenadas do robô cartesiano.

O bloco de função SMC_ReadNCFile (vide figura 5.2), é utilizado para realizar a leitura de um programa em código G salvo com a extensão *.txt*. Para o processo de leitura do arquivo é necessário que o programa seja carregado para o controlador LMC058 por meio do software SoMachine V4.1.

De modo simplificado este bloco tem a função de ler os dados do arquivo e tornar esse código disponível para o decodificador.

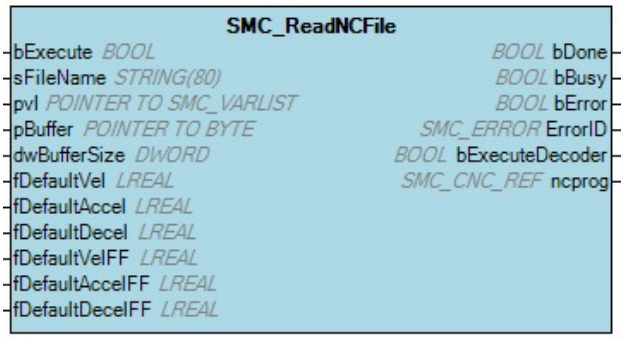


Figura 5.2: Bloco de Função do SMC_ReadNCFile. [30]

- Entradas:

  BExecute: Se verdadeiro, inicia a execução do bloco de função;

  SFileName: Entrada para uma string que contém o nome do arquivo com a extensão .txt;

  Pvl: Ponteiro para a SMC_VARLIST, caso não haja variáveis no programa CNC essa entrada não é definida. Para a aplicação de soldagem não foi definido esse campo;

  PBuffer: Ponteiro para uma área reservada que armazena o código lido do controlador. Para o programa principal foi alocado um vetor de 1000 posições do tipo SMC_GCODE_WORD. O tamanho grande do vetor é para garantir que, independente do código G criado por um usuário o armazenamento será feito;

DWBufferSize: Tamanho do *buffer* que foi alocado para armazenar o código G;

FDefaultVel: Velocidade padrão utilizada caso a velocidade não esteja definida no código G;

FDefaultAcc: Aceleração padrão utilizada caso a aceleração não esteja definida no código G;

FDefaultDecel: Desaceleração padrão utilizada caso a desaceleração não esteja definida no código G;

- Saídas:

BDone: Se verdadeiro, o bloco acabou a realizar o processo de leitura do arquivo .txt;

BBusy: Se verdadeiro, o bloco ainda está realizando o processo de leitua do arquivo .txt;

BError: Se verdadeiro, sinaliza que um erro ocorreu;

ErrorID: Indentifica qual tipo de erro ocorreu;

BExecuteDecoder: Sinal para indicar que o bloco SMC_NCDecoder deve ser acionado. O bloco NCDecoder é o próximo a ser explicado;

NCprog: Entrada para o bloco de decodificação com um vetor de variáveis SMC_CNC_REF que são utilizadas como referência para a biblioteca SM3_CNC;

O bloco SMC_NCDecoder (vide figura 5.3) tem função de converter o código G (DIN 66025) em uma lista de objetos Softmotion_GEOINFO, essa lista é o formato padrão utilizado pelo CoDeSys Softmotion que está presente na biblioteca SM3_CNC do software SoMachine.

Este bloco deve ser usado junto com o SMC_ReadNCFile pois a sua saída está no formato de variáveis de referência utilizado pelo bloco de decodificação.

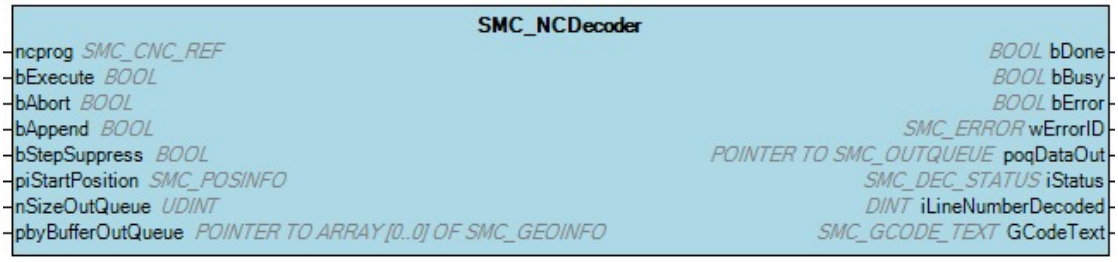


Figura 5.3: Bloco de Função do SMC_NCDecoder. [30]

- Entradas:

NCprog: Input para o bloco de decodificação com um vetor de variáveis SMC_CNC_REF que são utilizadas como referência para a biblioteca SM3_CNC;

BExecute: Se verdadeiro, o bloco de função está ativado;

BAbort: Se verdadeiro, o processo desse bloco de funções foi abortado;

BAppend: Se verdadeiro, ao final da decodificação do programa CNC o vetor que armazena as variáveis de posição e velocidade será limpado. Se falso, o vetor que armazena as variáveis

de posição e velocidade não serão limpados e qualquer outro programa decodificado será acrescentado ao final do vetor;

BStepSuppress: Se verdadeiro, as linhas do programa CNC que começam com "/"serão ignoradas. Caso contrário elas serão processadas;

PiStartPosition: Posição de onde a interpolação de pontos deve ser iniciada, ou seja, posição dos eixos que será a origem do programa CNC;

NSizeOutQueue: Essa variável contém o tamanho da lista de estrutura GEOINFO;

PbyBufferOutQueue: Essa entrada é um ponteiro para a lista de estrutura GEOINFO;

- Saídas:

BDone: Se verdadeiro, o bloco finalizou o processo de decodificação dos dados;

BBusy: Se verdadeiro, o bloco está executando a sua função;

BError: Se verdadeiro, sinaliza que ocorreu um erro no bloco de função;

WErrorID: Indica qual erro ocorreu no bloco de função;

PoqDataOut: Ponteiro para um vetor de informações SMC_GEOINFO;

ILineNumberDecoded: Variável que conta o número de linha do programa CNC decodificado;

GCodeText: Ponteiro para um vetor de SMC_GCODE_TEXT para que o usuário possa confirir o código G lido;

O bloco SMC_RoundPath (vide figura 5.5), é um bloco de pré-processamento de pontos, ele tem como função fazer o arredondamento de curvas que possuem ângulos agudos. A distância entre a aresta e o ponto no qual os dois objetos adjacentes devem ser cortados é calculado pela metade do comprimento do primeiro objeto, metade do comprimento do segundo objeto e a soma do raio da ferramenta com o ângulo de tolerância.

A partir dessas três variáveis, o mínimo é formado e atribuído como ponto de corte (conforme a figura 5.4). Assim, os objetos são cortados em no máximo metade do comprimento do objeto mais curto. Além disso, se a soma de variável de entrada raio da ferramenta e palavra de código G é 0, então nenhum arredondamento é feito.

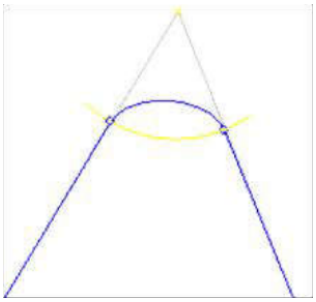


Figura 5.4: Exemplo de uma Trajetória Arredondada utilizando o SMC_RoundPath. [30]

O raio da ferramenta e a ângulo de tolerância são parâmetros pré-programados no bloco de função SMC_RoundPath, porém não tem aplicação no processo de soldagem por deposição de metal, os valores utilizados nesses campos são valores padrões de uma aplicação.

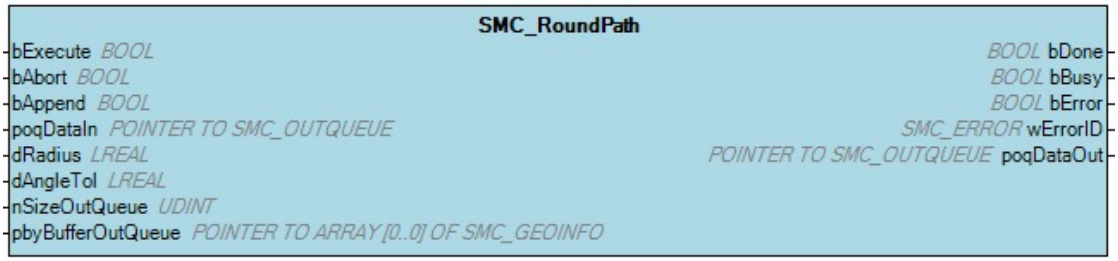


Figura 5.5: Bloco de Função do SMC_RoundPath. [30]

- Entradas:

  BExecute: Se verdadeiro, inicia a execução do bloco de função;

  BAbort: Se verdadeiro, o procedimento realizado pelo bloco de função será abortado;

  BAppend: Se falso, o vetor que armazena informações sobre os pontos pré-processados será limpado ao final da execução do bloco. Caso verdadeiro as informações continuarão armazenadas;

  PoqDataIn: Ponteiro para um vetor de informações GEOINFO que contém pontos que ainda não foram pré-processados pelo bloco de função;

  DRadius: Está entrada contém o valor que será adicionado ao raio de ferramenta para definir a distância miníma em que os objetos particulares serão descartados e substituídos por uma interpolação spline. Essa variável não será utilizada em nenhum momento no programa principal;

  DAngleTol: Está variável descreve o angulo de tolerância em que um ângulo agudo não será arredondado. Essa variável não será utilizada no programa principal;

  NSizeOutQueue: Essa entrada contém o tamanho do vetor de GEOINFO que foi armazenado na memória do controlador;

  PbyBufferOutQueue: Essa entrada contém o ponteiro para o vetor de GEOINFO onde estão armazenadas as informações sobre o programa CNC;

- Saídas:

  BDone: Se verdadeiro, o bloco de função acabou de realizar a sua tarefa;

  BBusy: Se verdadeiro, o bloco ainda está realizando o pré-processamento dos pontos;

  BError: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  WErrorID: Identifica qual tipo de erro ocorreu;

  PoqDataOut: Ponteiro para um vetor de informações GEOINFO que já foi pré-processado pelo bloco de função;

O bloco de função SMC_AvoidLoop (vide figura 5.7), é utilizado na etapa e pré-processamento dos dados, após passar pelo SMC_RoundPath os dados são inseridos no SMC_AvoidLoop.

O bloco copiará um caminho definido, mas cortará os *loops*, ou seja, os ciclos contidos nele (conforme a figura 5.6). Se uma interseção é detectada dentro do caminho original, o caminho será cortado no ponto de intersecção, o *loop* será removido e o caminho será continuado com o resto da curva. Assim, um caminho contínuo livre de *loops* será o resultado.

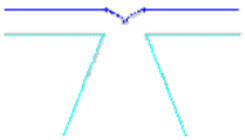


Figura 5.6: Exemplo de um Ciclo Corrigido utilizando o SMC_AvoidLoop. [30]

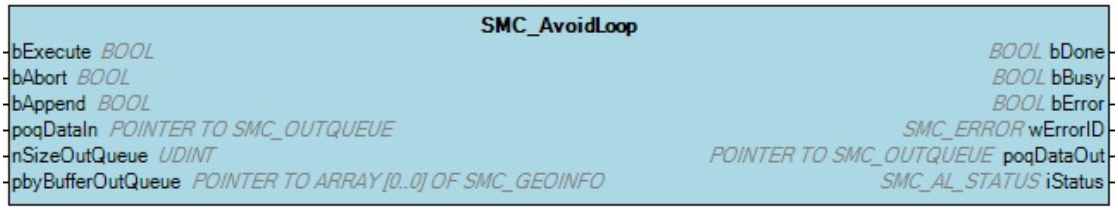


Figura 5.7: Bloco de Função do SMC_AvoidLoop. [30]

- Entradas:

  BExecute: Se verdadeiro, inicia a execução do bloco de função;

  BAbort: Se verdadeiro, o procedimento realizado pelo bloco de função será abortado;

  BAppend: Se falso, o vetor que armazena informações sobre os pontos pré-processados será limpado ao final da execução do bloco. Caso verdadeiro as informações continuarão armazenadas;

  PoqDataIn: Ponteiro para um vetor de informações GEOINFO que contém pontos que ainda não foram pré-processados pelo bloco de função;

  NSizeOutQueue: Essa entrada contém o tamanho do vetor de GEOINFO que foi armazenado na memória do controlador;

  PbyBufferOutQueue: Essa entrada contém o ponteiro para o vetor de GEOINFO onde estão armazenadas as informações sobre o programa CNC;

- Saídas:

  BDone: Se verdadeiro, o bloco de função acabou de realizar a sua tarefa;

  BBusy: Se verdadeiro, o bloco ainda está realizando o pré-processamento dos pontos;

  BError: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  WErrorID: Identifica qual tipo de erro ocorreu;

PoqDataOut: Ponteiro para um vetor de informações GEOINFO que já foi pré-processado pelo bloco de função;

O bloco de função SMC_ToolCorr também faz parte do SM3_CNC (vide figura 5.9), este bloco cria um caminho deslocado baseado no perfil original da trajetória (conforme a figura 5.8). No caminho deslocado cada ponto de cada objeto do caminho tem uma distância definida para o ponto original e sua pontos vizinhos imediatos (correção do raio de ferramenta). Neste caso assim como para os blocos anteriores o raio de ferramenta foi definido com um valor padrão.

Portanto, o caminho deslocado garante que cada de seus pontos tem uma distância fixa para o caminho original. Uma aplicação típica é a fresagem de um contorno programado com uma fresa de diâmetro definido. Para compensar o raio a fresa deve seguir um caminho adequadamente deslocado.

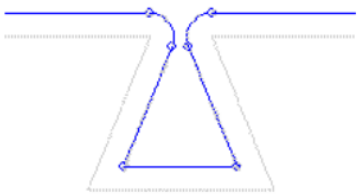


Figura 5.8: Exemplo de um Caminho Corrigido utilizando o SMC_ToolCorr. [30]

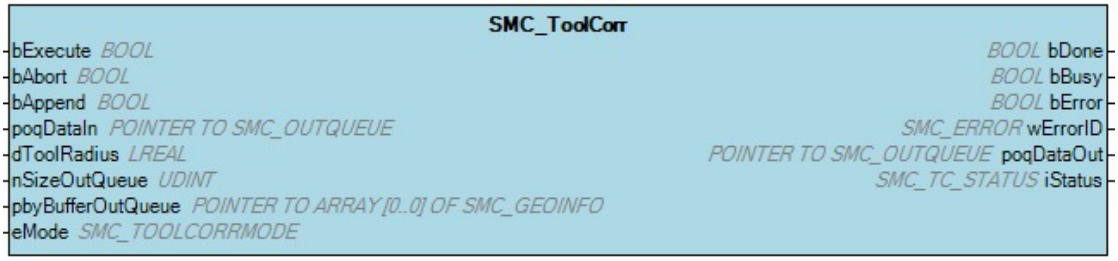


Figura 5.9: Bloco de Função do SMC_ToolCorr. [30]

- Entradas:

  BExecute: Se verdadeiro, inicia a execução do bloco de função;

  BAbort: Se verdadeiro, o procedimento realizado pelo bloco de função será abortado;

  BAppend: Se falso, o vetor que armazena informações sobre os pontos pré-processados será limpado ao final da execução do bloco. Caso verdadeiro as informações continuarão armazenadas;

  PoqDataIn: Ponteiro para um vetor de informações GEOINFO que contém pontos que ainda não foram pré-processados pelo bloco de função;

  NSizeOutQueue: Essa entrada contém o tamanho do vetor de GEOINFO que foi armazenado na memória do controlador;

PbyBufferOutQueue: Essa entrada contém o ponteiro para o vetor de GEOINFO onde estão armazenadas as informações sobre o programa CNC;

- Saídas:

  BDone: Se verdadeiro, o bloco de função acabou de realizar a sua tarefa;

  BBusy: Se verdadeiro, o bloco ainda está realizando o pré-processamento dos pontos;

  BError: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  WErrorID: Identifica qual tipo de erro ocorreu;

  PoqDataOut: Ponteiro para um vetor de informações GEOINFO que já foi pré-processado pelo bloco de função;

O bloco de função SMC_LimitCircularVelocity (vide figura 5.10), é utilizado quando há curvas no código G. Esse bloco verifica os elementos particulares do vetor de informações (OutQueue) e limita as velocidades de trajetória circulares. Para limitar a aceleração a um valor prescrito, a velocidade do arco na transição não deve exceder a raiz quadrada da aceleração pelo raio. O módulo controla a transição entre dois elementos (de um movimento linear para um movimento em arco circular, de um movimento em arco circular para um movimento linear ou de um movimento circular para outro movimento circular) e adapta a velocidade da extremidade do primeiro elemento para que o salto de aceleração não exceda o valor dMaxAccJump. Além disso, o módulo limita a aceleração em X e Y ao valor dMaxAcc.

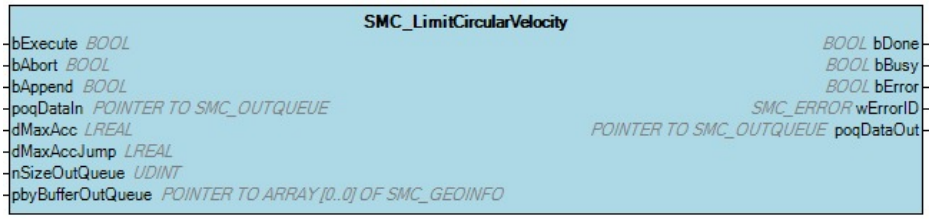


Figura 5.10: Bloco de Função do SMC_LimitCircularVelocity. [30]

- Entradas:

  BExecute: Se verdadeiro, inicia a execução do bloco de função;

  BAbort: Se verdadeiro, o procedimento realizado pelo bloco de função será abortado;

  BAppend: Se falso, o vetor que armazena informações sobre os pontos pré-processados será limpado ao final da execução do bloco. Caso verdadeiro as informações continuarão armazenadas;

  PoqDataIn: Ponteiro para um vetor de informações GEOINFO que contém pontos que ainda não foram pré-processados pelo bloco de função;

  DMaxAcc: Essa entrada fornece o valor da máxima aceleração permitida para os arcos circulares.

DMaxAccJump: Essa entrada fornece o valor do máximo salto de aceleração permitido para a transição entre dois arcos.

NSizeOutQueue: Essa entrada contém o tamanho do vetor de GEOINFO que foi armazenado na memória do controlador;

PbyBufferOutQueue: Essa entrada contém o ponteiro para o vetor de GEOINFO onde estão armazenadas as informações sobre o programa CNC;

- Saídas:

BDone: Se verdadeiro, o bloco de função acabou de realizar a sua tarefa;

BBusy: Se verdadeiro, o bloco ainda está realizando o pré-processamento dos pontos;

BError: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

WErrorID: Identifica qual tipo de erro ocorreu;

PoqDataOut: Ponteiro para um vetor de informações GEOINFO que já foi pré-processado pelo bloco de função;

O último bloco de função utilizado no pré-processamento dos dados é o SMC_CheckVelocities (vide a figura 5.11). Este bloco é importante pois serve para verificar se as velocidade dos pontos que devem ser interpolados estão dentro da máxima velocidade definida por meio do bloco SMC_ReadNCFile (vide figura 5.2).

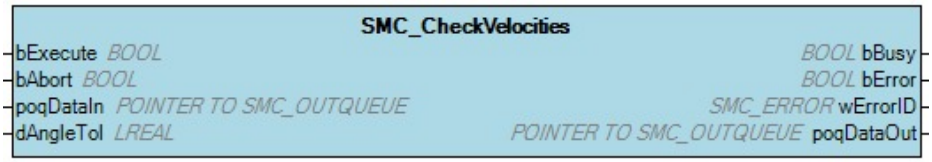


Figura 5.11: Bloco de Função do SMC_CheckVelocities. [30]

- Entradas:

BExecute: Se verdadeiro, inicia a execução do bloco de função;

BAbort: Se verdadeiro, o procedimento realizado pelo bloco de função será abortado;

PoqDataIn: Ponteiro para um vetor de informações GEOINFO que contém pontos que ainda não foram pré-processados pelo bloco de função;

DAngleTol: Está variável descreve o angulo de tolerância em que um ângulo agudo não será arredondado. Essa variável não será utilizada no programa principal;

- Saídas:

BBusy: Se verdadeiro, o bloco ainda está realizando o pré-processamento dos pontos;

BError: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

WErrorID: Identifica qual tipo de erro ocorreu;

PoqDataOut: Ponteiro para um vetor de informações GEOINFO que já foi pré-processado pelo bloco de função;

Após o pré-processamento dos dados ocorre a interpolação dos pontos no sistema de coordenadas do robô cartesiano, a etapa de interpolação é feita utilizando o bloco de função SMC_Interpolator da biblioteca SM3_CNC (vide figura 5.12). O bloco SMC_Interpolator, é usado para converter um caminho contínuo descrito pelos objetos do vetor de GEOINFO (SMC_GEOINFO) em pontos de posição de caminho discretos levando em conta um perfil de velocidade e um padrão definidos. Depois, esses pontos de posição serão tipicamente transformados pelo programa (por exemplo, para posições dos eixos dos *servo-drives*) e enviado para os *servo-drives*.

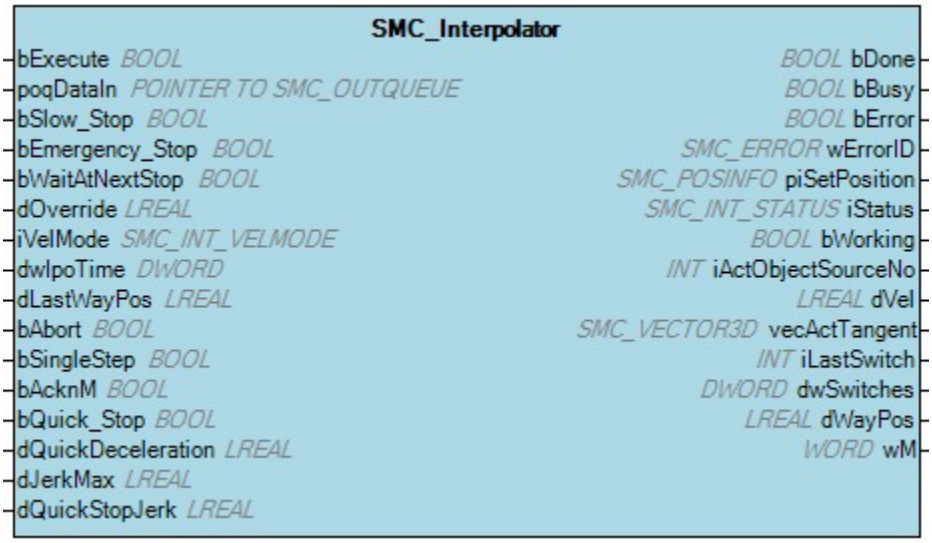


Figura 5.12: Bloco de Função do SMC_Interpolator. [30]

Esse bloco de função possui várias entradas e saídas, entretanto para a aplicação desenvolvida foram utilizados somente as seguintes entradas e saídas.

- Entradas:

  BExecute: Se verdadeiro, inicia a execução do bloco de função;

  PoqDataIn: Ponteiro para um vetor de informações GEOINFO que contém pontos que ainda não foram interpolados pelo bloco de função;

  BEmergency_Stop: Se verdadeiro, o bloco de função encerra imediatamente o seu funcionamento, isso significa que a posição atual será retida e a velocidade será definida como 0;

  IVelMode: Essa entrada define o perfil da curva de velocidade que para o processo de soldagem 3D foi definida como Sin^2;

  DwIpoTime: Essa variável deve ser definida para cada chamada, ela representa o período de ciclo. Em nossa aplicação o período foi definido como 4000 us;

- Saídas:

  BDone: Se verdadeira, o bloco acabou de interpolar todos os pontos;

BBusy: Se verdadeiro, o bloco ainda está realizando a interpolação dos pontos;

BError: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

WErrorID: Indentifica qual tipo de erro ocorreu;

PiSetPosition: Essa saída reflete as posições calculadas que contém as coordenadas cartesianas para o próximo ponto. Essa será a saída para o bloco de função SMC_ControlAxisByPos;

O último bloco de função que deve ser mencionado é o bloco SMC_ControlAxisByPos (vide figura 5.13). Esse bloco pode ser empregado para escrever o alvo posições a uma estrutura de movimentação e testar a estrutura para saltos. É usado principalmente no contexto com CNC e SMC_Interpolator. Para controle direto dos valores de ajuste do eixo. Após a interpolação dos pontos, agora o eixo deve se movimentar para a coordenada cartesiana que corresponde ao código G da linha que foi lida, pré-processada e interpolada por meio do SMC_ControlAxisByPos.

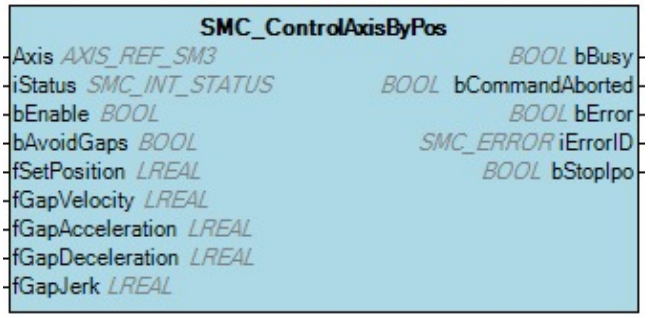


Figura 5.13: Bloco de Função do SMC_Interpolator. [30]

Assim como para o caso do SMC_Interpolator o bloco SMC_ControlAxisByPos possui várias entradas e saídas, para a aplicação de soldagem 3D foram utilizadas as seguintes entradas e saídas.

- Entradas:

  Axis: Recebe o nome do eixo que será movimentado. O nome do eixo é dado no momento em que se escolhe o dispositivo a ser controlado;

  BEnable: Se verdadeiro, inicia a execução do bloco de função;

  fSetPosition: Essa entrada é as posições em coordenadas cartesianas para onde o eixo de se deslocar;

- Saídas:

  BBusy: Se verdadeiro, o bloco ainda está realizando o movimento do eixo para o ponto de destino;

  BError: Se verdadeiro, sinaliza que um erro ocorreu no bloco de função;

  WErrorID: Indentifica qual tipo de erro ocorreu;

Capítulo 6

Resultados e Discussões

6.1 Programa Principal

O programa principal, desenvolvido durante este trabalho, integra os dispositivos de controle de movimento do robô cartesiano mediante a seleção do modo de funcionamento dos *servo-drives* para o movimento sincronizado e interpolado multi-eixos. O programa foi desenvolvido utilizando a linguagem de programação FBD (Function Block Diagram) disponível no controlador. O programa principal incorpora rotinas tais como: busca de referência do robô, ajuste de offsets para referência da peça, movimento manual para posições absolutas e relativas (*jog*), carregamento e execução de programas. Além das rotinas, o programa principal também é capaz de interpretar códigos G, permitindo a leitura e execução de qualquer programa desenvolvido utilizando o formato padronizado segundo a norma DIN 66025.

Para o usuário acessar todas as funcionalidades do programa principal sem a necessidade de um computador e do software SoMachine V4.1, foi desenvolvida uma aplicação no Vijeo-Designer utilizando imagens que são exibidas na tela da IHM. Essa aplicação visual que se encontra no programa principal consiste em telas exibidas na IHM, que utilizam itens visuais como botões de ações, teclados numéricos e indicadores luminoso para iniciar as rotinas descritas acima, inserir novos dados no controlador e alertar de modo visual os estados dos eixos do robô cartesiano. Essa ferramenta visual auxilia no manuseio fácil do robô cartesiano.

Para melhor entender a topologia de como o programa principal está estruturado, foram criados alguns fluxogramas que exibem de modo visual como é o fluxo de dados no programa . Como o fluxo de informações é muito grande para ser mostrado no espaço dessa folha eles foram reduzidos e seus quadros foram numerados de modo a possibilitar o entendimento de suas respectivas funções, cujas descrições se encontram nas tabelas 6.1 6.2 6.3 6.4 6.5 intercaladas entre os fluxogramas mostrados nas figuras 6.1 6.2 6.3 6.4 6.5.

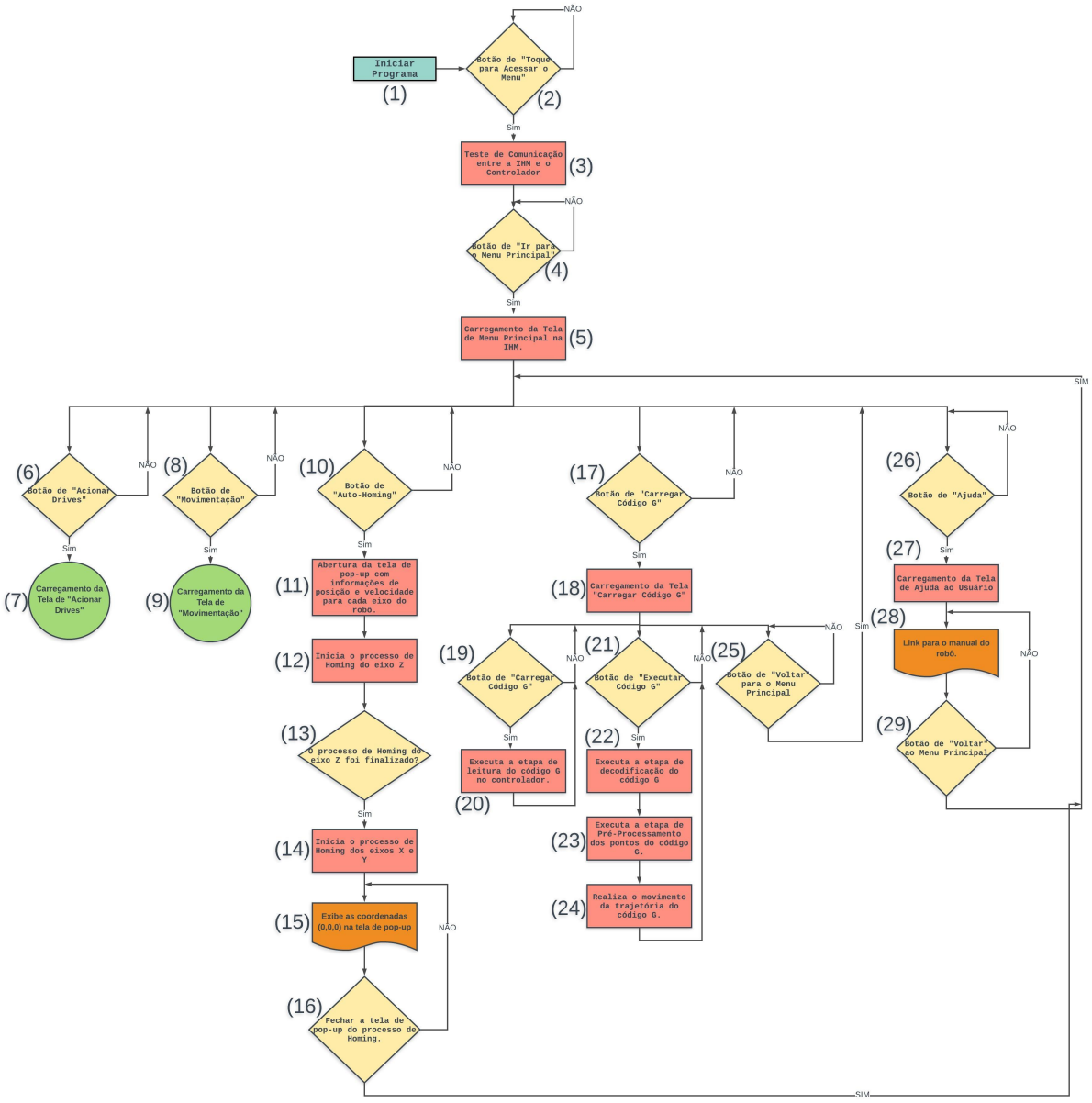


Figura 6.1: Fluxograma Programa Principal (1/5).

Tabela 6.1: Tabela Fluxograma Programa Principal (1/5).

Indicador	Forma/Modelo	Descrição na Forma
(1)	Inicial	Iniciar Programa
(2)	Decisão	Botão de "Toque para Acessar o Menu"
(3)	Processo	Teste de Comunicação entre a IHM e o Controlador
(4)	Decisão	Botão de "Ir para o Menu Principal"
(5)	Processo	Carregamento da Tela de Menu Principal na IHM.
(6)	Decisão	Botão de "Acionar Drives"
(7)	Referência na Página	Carregamento da Tela de "Acionar Drives"
(8)	Decisão	Botão de "Movimentação"
(9)	Referência na Página	Carregamento da Tela de "Movimentação"
(10)	Decisão	Botão de "Auto-Homing"
(11)	Processo	Abertura da tela de pop-up com informações de posição e velocidade para cada eixo do robô.
(12)	Processo	Inicia o processo de "Homing"do eixo Z
(13)	Decisão	O processo de Homing do eixo Z foi finalizado?
(14)	Decisão	Inicia o processo de Homing dos eixos X e Y
(15)	Resultado	Exibe as coordenadas (0,0,0) na tela de pop-up
(16)	Decisão	Fechar a tela de pop-up do processo de "Homing".
(17)	Decisão	Botão de "Carregar Código G"
(18)	Processo	Carregamento da Tela "Carregar Código G"
(19)	Decisão	Botão de "Carregar Código G"
(20)	Processo	Executa a etapa de leitura do código G no controlador.
(21)	Decisão	Botão de "Executar Código G"
(22)	Processo	Executa a etapa de decodificação do código G
(23)	Processo	Executa a etapa de Pré-Processamento dos pontos do código G.
(24)	Processo	Realiza o movimento da trajetória do código G.
(25)	Decisão	Botão de "Voltar"para o Menu Principal
(26)	Decisão	Botão de "Ajuda"
(27)	Processo	Carregamento da Tela de Ajuda ao Usuário
(28)	Resultado	Link para o manual do robô.
(29)	Decisão	Botão de "Voltar"ao Menu Principal

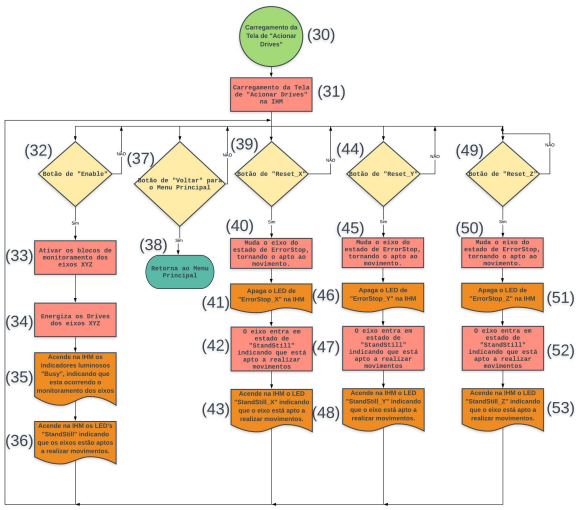


Figura 6.2: Fluxograma Programa Principal (2/5).

Tabela 6.2: Tabela Fluxograma Programa Principal (2/5).

Indicador	Forma/Modelo	Descrição na Forma
(30)	Referência na página	Iniciar Programa
(31)	Processo	Carregamento da Tela de "Acionar Drives"na IHM
(32)	Decisão	Botão de "Enable"
(33)	Processo	Ativar os blocos de monitoramento dos eixos XYZ
(34)	Processo	Energiza os Drives dos eixos XYZ
(35)	Resultado	Acende na IHM os indicadores luminosos "Busy", indicando que esta ocorrendo o monitoramento dos eixos
(36)	Resultado	Acende na IHM os indicadores luminosos "StandStill"indicando que os eixos estão aptos a realizar movimentos.
(37)	Decisão	Botão de "Voltar"para o Menu Principal
(38)	Final	Retorna ao Menu Principal
(39)	Decisão	Botão de "Reset_X"
(40)	Processo	Muda o eixo do estado de ErrorStop, tornando o apto ao movimento.
(41)	Resultado	Apaga o indicador luminoso de "ErrorStop_X"na IHM
(42)	Processo	O eixo entra em estado de "StandStill"indicando que está apto a realizar movimentos
(43)	Resultado	Acende na IHM o indicador luminoso "StandStill_X"indicando que o eixo está apto a realizar movimentos.
(44)	Decisão	Botão de "Reset_Y"
(45)	Processo	Muda o eixo do estado de ErrorStop, tornando o apto ao movimento.
(46)	Resultado	Apaga o indicador luminoso de "ErrorStop_Y"na IHM
(47)	Processo	O eixo entra em estado de "StandStill"indicando que está apto a realizar movimentos
(48)	Resultado	Acende na IHM o indicador luminoso "StandStill_Y"indicando que o eixo está apto a realizar movimentos.
(49)	Decisão	Botão de "Reset_Z"
(50)	Processo	Muda o eixo do estado de ErrorStop, tornando o apto ao movimento.
(51)	Resultado	Apaga o indicador luminoso de "ErrorStop_Z"na IHM
(52)	Processo	O eixo entra em estado de "StandStill"indicando que está apto a realizar movimentos
(53)	Resultado	Acende na IHM o indicador luminoso "StandStill_Z"indicando que o eixo está apto a realizar movimentos.

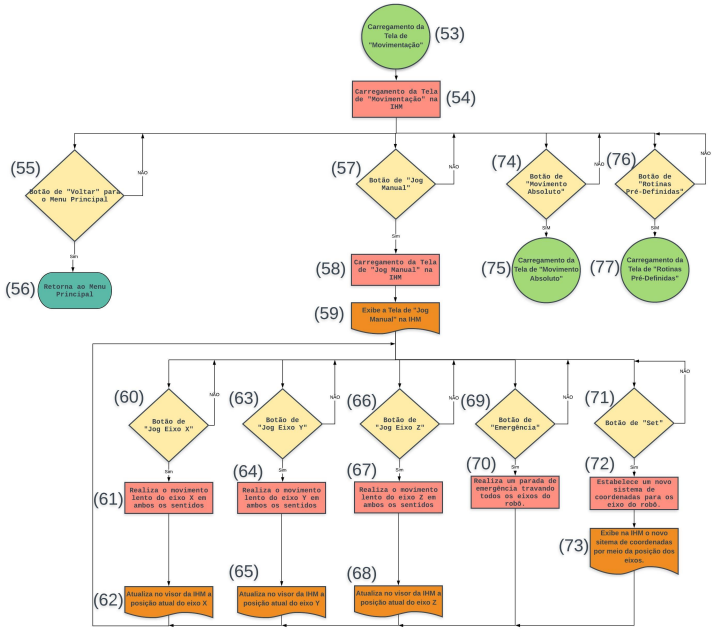


Figura 6.3: Fluxograma Programa Principal (3/5).

Tabela 6.3: Tabela Fluxograma Programa Principal (3/5).

Indicador	Forma/Modelo	Descrição na Forma
(53)	Referência na página	Carregamento da Tela de "Movimentação"
(54)	Processo	Carregamento da Tela de "Movimentação"na IHM
(55)	Decisão	Botão de "Voltar"para o Menu Principal
(56)	Final	Retorna ao Menu Principal
(57)	Decisão	Botão de "Jog Manual"
(58)	Processo	Carregamento da Tela de "Jog Manual"na IHM
(59)	Resultado	Exibe a Tela de "Jog Manual"na IHM
(60)	Decisão	Botão de "Jog Eixo X"
(61)	Processo	Realiza o movimento lento do eixo X em ambos os sentidos
(62)	Resultado	Atualiza no visor da IHM a posição atual do eixo X
(63)	Decisão	Botão de "Jog Eixo Y"
(64)	Processo	Realiza o movimento lento do eixo Y em ambos os sentidos
(65)	Resultado	Atualiza no visor da IHM a posição atual do eixo Y
(66)	Decisão	Botão de "Jog Eixo Z"
(67)	Processo	Realiza o movimento lento do eixo Z em ambos os sentidos
(68)	Resultado	Atualiza no visor da IHM a posição atual do eixo Z
(69)	Decisão	Botão de "Emergência"
(70)	Processo	Realiza um parada de emergência travando todos os eixos do robô.
(71)	Decisão	Botão de "Set"
(72)	Processo	Estabelece um novo sistema de coordenadas para os eixo do robô.
(73)	Resultado	Exibe na IHM o novo sitema de coordenadas por meio da posição dos eixos.
(74)	Decisão	Botão de "Movimento Absoluto"
(75)	Referência na Página	Carregamento da Tela de "Movimento Absoluto"
(76)	Decisão	Botão de "Rotinas Pré-Definidas"
(77)	Referência na Página	Carregamento da Tela de "Rotinas Pré-Definidas"

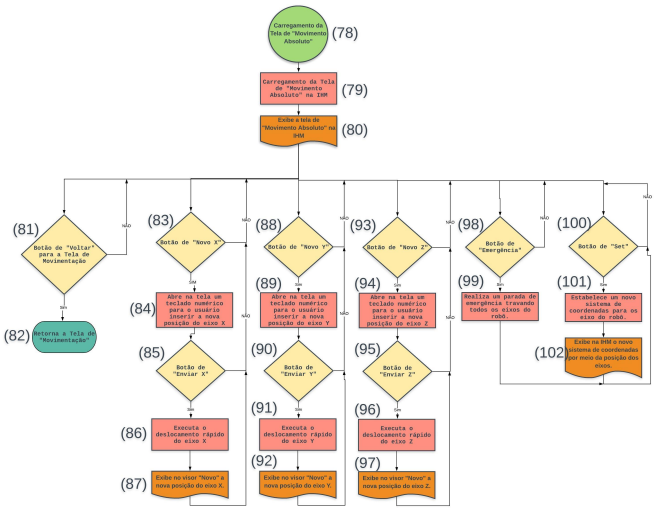


Figura 6.4: Fluxograma Programa Principal (4/5).

Tabela 6.4: Tabela Fluxograma Programa Principal (4/5).

Indicador	Forma/Modelo	Descrição na Forma
(78)	Referência na página	Carregamento da Tela de "Movimento Absoluto"
(79)	Processo	Carregamento da Tela de "Movimento Absoluto"na IHM
(80)	Resultado	Exibe a tela de "Movimento Absoluto"na IHM
(81)	Decisão	Botão de "Voltar"para a Tela de Movimentação
(82)	Final	Retorna a Tela de "Movimentação"
(83)	Decisão	Botão de "Novo X"
(84)	Processo	Abre na tela um teclado numérico para o usuário inserir a nova posição do eixo X
(85)	Decisão	Botão de "Enviar X"
(86)	Processo	Executa o deslocamento rápido do eixo X
(87)	Resultado	Exibe no visor "Novo"a nova posição do eixo X.
(88)	Decisão	Botão de "Novo Y"
(89)	Processo	Abre na tela um teclado numérico para o usuário inserir a nova posição do eixo Y
(90)	Decisão	Botão de "Enviar Y"
(91)	Processo	Executa o deslocamento rápido do eixo Y
(92)	Resultado	Exibe no visor "Novo"a nova posição do eixo Y
(93)	Decisão	Botão de "Novo Z"
(94)	Processo	Abre na tela um teclado numérico para o usuário inserir a nova posição do eixo Z
(95)	Decisão	Botão de "Enviar Z"
(96)	Processo	Executa o deslocamento rápido do eixo Z
(97)	Resultado	Exibe no visor "Novo"a nova posição do eixo Z
(98)	Decisão	Botão de "Emergência"
(99)	Processo	Realiza um parada de emergência travando todos os eixos do robô
(100)	Decisão	Botão de "Set"
(101)	Processo	Estabelece um novo sistema de coordenadas para os eixo do robô.
(102)	Resultado	Exibe na IHM o novo sistema de coordenadas por meio da posição dos eixos.

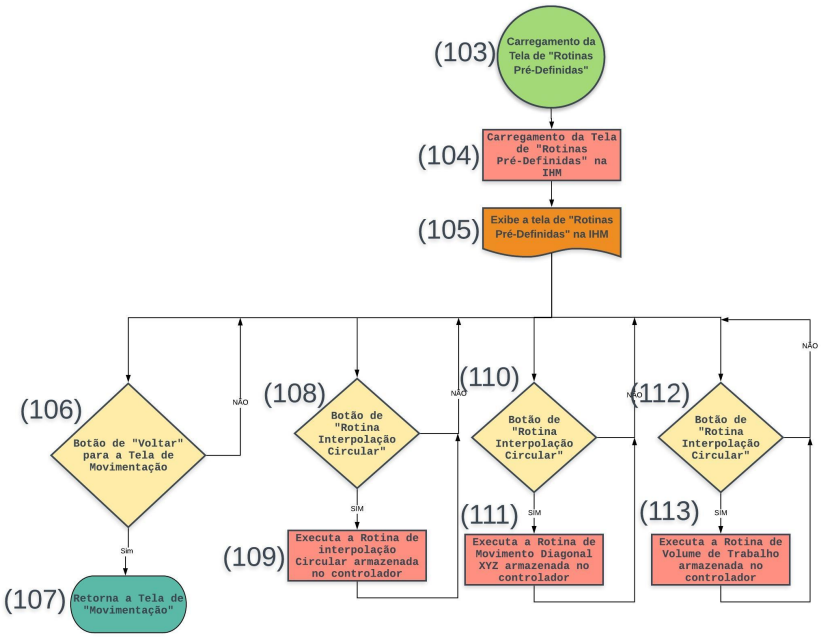


Figura 6.5: Fluxograma Programa Principal (5/5).

Tabela 6.5: Tabela Fluxograma Programa Principal (5/5).

Indicador	Forma/Modelo	Descrição na Forma
(103)	Referência na página	Carregamento da Tela de "Rotinas Pré-Definidas"
(104)	Processo	Carregamento da Tela de "Rotinas Pré-Definidas"na IHM
(105)	Resultado	Exibe a tela de "Rotinas Pré-Definidas"na IHM
(106)	Decisão	Botão de "Voltar"para a Tela de Movimentação
(107)	Final	Retorna a Tela de "Movimentação"
(108)	Decisão	Botão de "Rotina Interpolação Circular"
(109)	Processo	Executa a Rotina de interpolação Circular armazenada no controlador
(110)	Decisão	Botão de "Rotina Interpolação Circular"
(111)	Processo	Executa a Rotina de Movimento Diagonal XYZ armazenada no controlador
(112)	Decisão	Botão de "Rotina Interpolação Circular"
(113)	Processo	Executa a Rotina de Volume de Trabalho armazenada no controlador

O fluxograma exibe tudo o que foi desenvolvido durante o projeto e mostra como corre o fluxo de informação dentro do programa principal. A etapa que mais teve a dedicação do tempo durante o desenvolvimento do projeto foi a etapa de estabelecimento da comunicação entre os *hardwares*.

Por não ser um processo intuitivo foi necessário ler os manuais dos *servo-drives* Lexium 32M [9] [11] [12], os manuais do controlador LMC058 [10] e os manuais da IHM [28] [29]. Além disso, foi necessário fazer a leitura de materiais complementares [30] [31] para entender o funcionamento das bibliotecas SM3_Basic e SM3_CNC. A configuração da rede CANMotion foi essencial para solucionar um problema de sincronismo entre os *servo-drives* que havia sido constatado pelo último aluno a trabalhar no robô cartesiano [5]. Esse problema não está mais ocorrendo e agora é possível executar a movimentação suave dos eixos.

6.2 Operando o Robô MAXR23-S42-H42-C42

A integração dos sistemas do robô MAX R23-S42-H42-C42 permitiu o desenvolvimento de uma interface de fácil uso para os operadores do robô cartesiano. O programa desenvolvido para a IHM, "Programa_Principal", busca fornecer ao usuário uma acesso fácil e rápido a todas as funcionalidades implementadas durante o trabalho realizado neste projeto.A interface permite acionar os *drives* dos eixos X, Y e Z, realizando assim o movimento em coordenadas absolutas referenciadas aos sensores de fim de curso, executar rotinas pré-programadas no controlador, carregar e executar programas desenvolvidos utilizando a linguagem código G de modo simplificado.

A tela inicial figura 6.6, exibe a informação do local e o modelo do robô cartesiano da *Schneider Electric*, na parte inferior da tela é possível ler as instruções "Toque aqui para acessar o MENU". Com um leve clique na parte inferior da tela, o usuário é direcionado para uma tela de espera (vide figura 6.7) enquanto a IHM executa um rápido teste de comunicação. Após cerca de 10 segundos, surge um botão com a mensagem "Ir para Menu Principal"que direciona o usuário para o MENU.

Figura 6.6: Tela inicial da IHM, contem informações sobre o Local e Modelo do Robô Cartesiano.

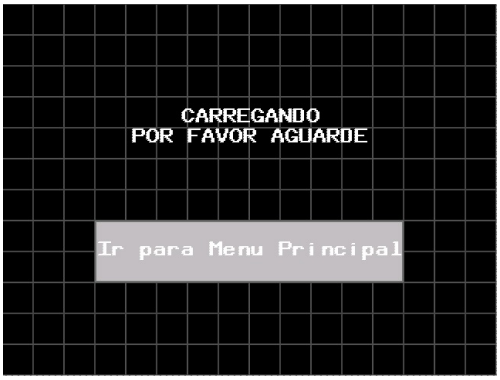


Figura 6.7: Tela de Carregamento, Execução do Teste de Comunicação.

No MENU figura 6.8 o usuário encontra disponível 5 botões, "Acionar Drives", "Movimentação", "Auto-Homing", "Carregar Código G"e "Ajuda". Cada botão possui uma funcionalidade que será abordada e explicada.

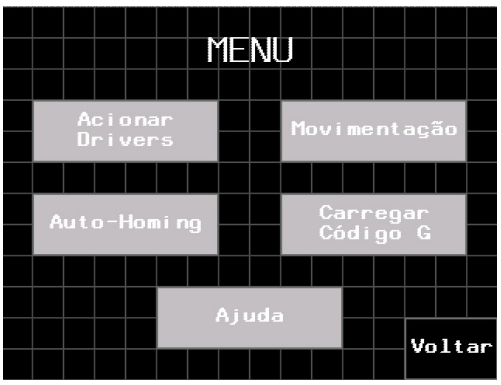


Figura 6.8: Tela do MENU Principal do Programa.

Ao pressionar o botão "Acionar Drives", o usuário é redirecionado para a tela da figura 6.9, essa tela é responsável pelo acionamento simplificado dos *drives* de cada eixo do robô. Além do acionamento, a tela auxilia o usuário a monitorar erros, por meio dos indicadores luminosos, e a ativar a função de *reset* dos *servo-drives* dos eixos X,Y e Z.

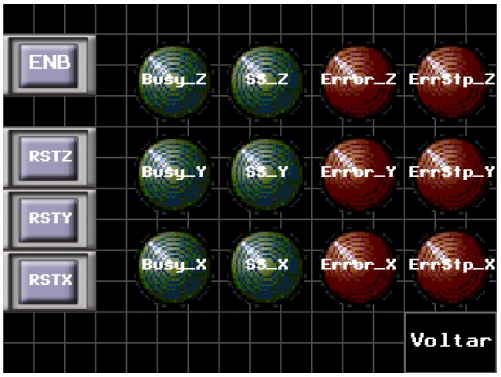


Figura 6.9: Tela de Acionamento, Monitoramento e *Reset* dos *Drives*.

O botão ENB, que se encontra no canto superior esquerdo da tela 6.9, ativa simultaneamente os blocos MC_ReadStatus e MC_Power de cada eixo. Ao pressionar o ENB, o conjunto de indicadores luminosos Busy_Z, Busy_Y, Busy_X, StandStill_Z, StandStill_Y e StandStill_X acendem, indicando que os *servo-drives* foram ativados e o robô está apto para o movimento sincronizado dos eixos. No caso do conjunto de indicadores luminosos Error_Z, Error_Y e Error_X acenderem, isso significa que ocorreu um erro no acionamento dos *servo-drives*. Portanto, é necessário acessar o SoMachine V4.1 para determinar qual o error específico ocorreu.

Para o caso em que os indicadores luminosos ErrorStop_Z, ErrorStop_Y e/ou ErrorStop_Z estão acionados, isso significa que os *drives* não se encontram no estado *StandStill*, o problema pode ser solucionado apertando o(s) botão(ões) Reset_Z, Reset_Y e/ou Reset_X para retornar o respectivo eixo (X, Y ou Z) que se encontra no estado de *ErrorStop* para o estado de *StandStill*.

Ao pressionar o botão "Movimentação"no MENU principal 6.8, o usuário será redirecionado para a tela da figura 6.10. Nesta, existem 3 botões que dão acesso a outras telas.

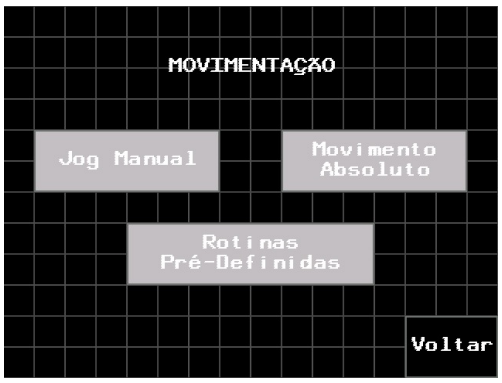


Figura 6.10: Tela de Movimentação do Robô.

O primeiro botão, "Jog Manual", redireciona o usuário para a tela "Jog Manual"(figura 6.11). Nesta, no canto superior esquerdo, existe o botão "SET", que ativa o bloco de função MC_SetPosition para cada um dos eixos XYZ. Esse botão é utilizado para definir uma nova origem de coordenadas onde os eixos estiverem estacionados.

No canto superior direito da tela "Jog Manual", encontra-se o botão "STOP", configurado com o bloco de função MC_Stop. Esse botão foi colocado como uma medida de segurança. Ao ser acionado os *servo-drives* dos eixos saem do estado *Motion* para o estado de *StandStill*, encerrando assim o deslocamento dos eixos.

No centro da tela "Jog Manual", existem dois botões (setas) para cada um dos eixos. A seta que aponta para cima representa um movimento lento no sentido positivo do eixo, já a seta apontando para baixo representa um movimento lento no sentido negativo do eixo. O visor localizado abaixo das setas exibe para o usuário a posição atual do efetuador terminal para cada um dos eixos. O movimento lento da tela "Jog Manual"está relacionado com o bloco de função MC_Jog, com velocidade de deslocamento fixada em 8 mm/s.

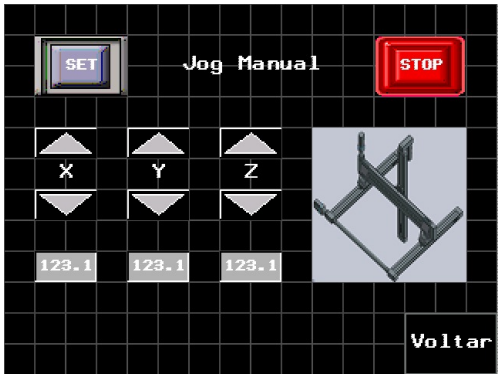


Figura 6.11: Tela de Movimento Lento para os Eixos X, Y e Z.

O segundo botão da tela "Movimentação"(figura 6.10), denominado "Movimento Absoluto", direciona o usuário para a tela de mesmo nome (figura 6.12). Esta dispõe de botões denominados "STOP"e "SET", cujas funções são idênticas às dos botões de mesmo nome exibidos na tela de "Jog Manual". Além disso, a tela possui 6 visores, dois para cada eixo do robô cartesiano, que

exibem a posição atual do eixo e nova posição para onde o eixo irá se deslocar.

Por meio de um toque no visor indicativo da nova posição para cada eixo que o usuário deseja movimentar, uma tela semelhante à da figura 6.13 aparece para que o usuário possa digitar um valor em coordenadas absolutas para movimento do efetuador terminal em cada eixo. Conforme exibido na figura 6.13, o valor digitado pelo usuário deve estar dentro do intervalo de mínimo e máximo definido via software.

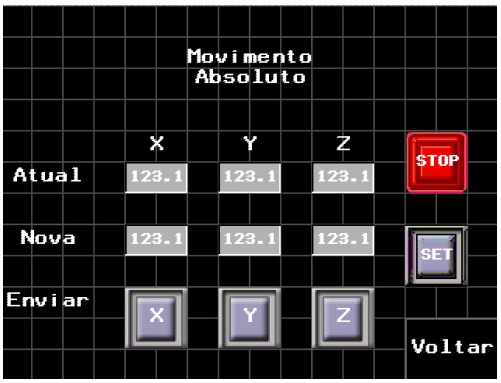


Figura 6.12: Tela de Movimento Rápido em Coordenadas Absolutas.

Figura 6.13: Campo de Entrada Numérica.

O deslocamento para a nova posição ocorre no momento em que o usuário aperta um dos botões "X", "Y"ou "Z". A tela de movimento absoluto está relacionado ao bloco de função MC_MoveAbsolute e portanto a velocidade em que cada eixo irá se movimentar já foi definida no código de função de blocos.

O terceiro botão "Rotinas Pré-Definidas"(vide figura 6.14) são algumas aplicações que foram desenvolvidas em código G para ilustrar o movimento sincronizado dos eixos (X,Y e Z). Ao pressionar o botão "Rotinas Pré-Definidas", o usuário é redirecionado para a tela representada na figura 6.14, na tela existem três botões que executam 3 rotinas diferentes. O primeiro botão executa a rotina de interpolação circular no plano XY, essa rotina foi criada por meio do editor CNC integrado ao SoMachine V4.1. A figura 6.15 mostra o perfil do movimento circular executado. Na imagem, é possível identificar as acelerações representadas pela variação das cores atribuídas ao

longo dos traços em que as cores esverdeadas representam trechos com aceleração positiva e as cores avermelhadas, trechos com aceleração negativa.

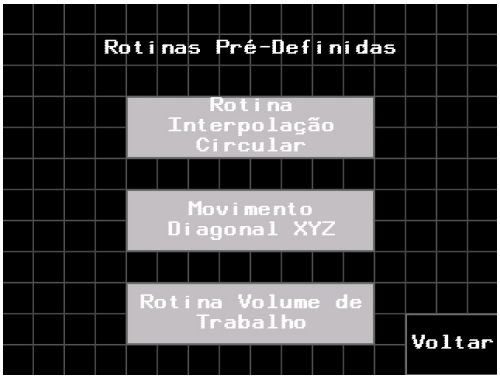


Figura 6.14: Tela de Rotinas Implementadas em Código G.

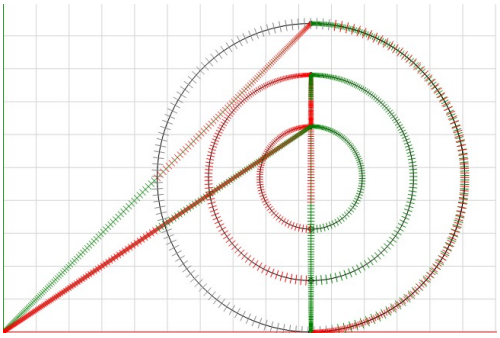


Figura 6.15: Rotina Circular gerada utilizando código G.

O segundo botão (vide figura 6.14) executa uma rotina de interpolação linear nos 3 eixos do robô, realizando um movimento na diagonal do volume de trabalho do robô cartesiano, como dito essa rotina tem por objetivo exibir para o usuário um movimento em que todos os três eixos do robô estão se deslocando para executar um movimento linear de XYZ. O terceiro e último botão da tela "Rotinas Pré-Definidas"(vide figura 6.14) é o botão que faz o mapeamento da área de trabalho do robô cartesiano, essa rotina exibe para o usuário a restrição de atuação do robô cartesiano.

Retornando para o MENU principal (vide figura 6.8) o próximo botão programado para o usuário é o "Auto-Homing", esse botão, conforme o seu rótulo já diz, habilita o bloco de função MC_Home que retorna o robô para a posição de *Home*. Esse botão retorna o fim de curso negativo, conforme definido na referência [5]. Ao executar o "Auto-Homing"a janela da figura 6.16 aparece para que o usuário possa acompanhar o deslocamento do efetuador terminal.

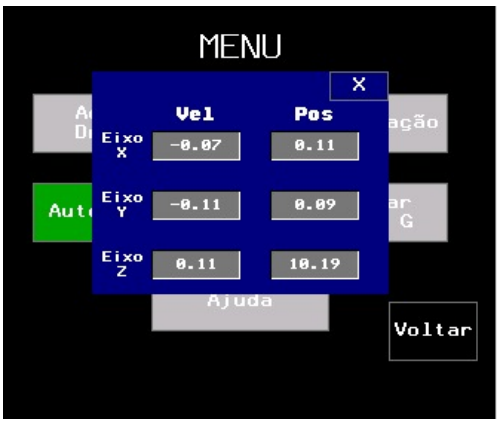


Figura 6.16: Janela com informações sobre Posição e Velocidade do Eixo.

O quarto botão do MENU (vide figura 6.8) é o "Carregar Código G", ao apertar o botão o usuário é redirecionado para a tela da figura 6.17. Na tela existem dois botões, o primeiro é o "Carregar Código G"esse botão é utilizado para fazer a leitura de um código G que se encontra no controlador. Por ser um bloco único, para fazer o carregamento correto, o usuário deve nomear o arquivo que será carregado no controlador como *"gcode.txt"*, isso é uma restrição que foi encontrada ao desenvolver o código. Após o carregamento bem sucedido do código, é necessário apertar o botão "Executar Código", esse botão fará as etapas de decodificação, pré-processamento e execução do código G.

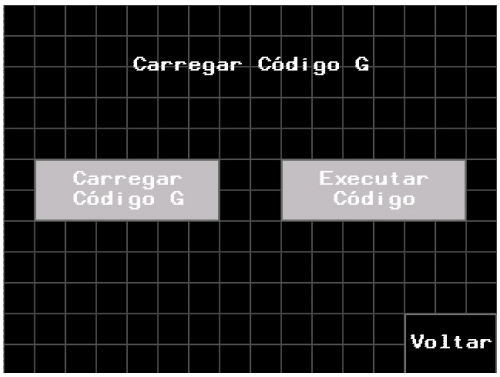


Figura 6.17: Tela de Carregamento do Código G.

O último botão disponível no MENU principal (vide figura 6.8) é o botão "Ajuda"esse botão foi desenvolvido para redirecionar o usuário para o manual de uso dos equipamento que compõem o robô MAXR23-S42-H42-C42.

Capítulo 7

Conclusões

O presente trabalho desenvolveu a integração dos sistemas de controle, de acionamento e de interface com o usuário de um robô cartesiano de 3 eixos. Também desenvolveu o planejamento de trajetória a partir de um programa escrito em linguagem G.

A primeira parte da integração dos sistemas consistiu no estabelecimento da comunicação entre os *servo-drives* Lexium 32M dos eixos X, Y e Z e o controlador LMC058 via rede industrial CANMotion. A correta configuração da comunicação via CANMotion, por meio da definição dos endereços dos dispositivos pendurados na rede, bem como dos parâmetros de comunicação adequados, permitiram a movimentação sincronizado dos eixos.

Configurou-se uma rede ethernet local para estabelecer comunicação entre o controlador, a interface homem-máquina e o computador do operador. Para isso o computador do operador era dotado de 2 portas ethernet independentes, sendo que uma delas ficou dedicada a essa rede local. Verificou-se que a comunicação somente se tornou efetiva ao se ajustarem a máscara de sub-rede conforme o manual e o *gateway*. Descobriu-se que este ultimo correspondia ao endereço padrão do controlador ajustado na fábrica (10.10.159.153). Essa informação não estava disponível nos manuais.

A seleção e programação de diversos blocos de funções da biblioteca SM3_Basic possibilitou o movimento sincronizado dos três eixos do robô cartesiano, permitindo que o usuário realize movimentações por coordenadas absolutas, posicionamento do efetuador terminal em uma coordenada específica e o a busca da origem dos eixos do robô cartesiano (*Homing*).

Programaram-se diversas telas na IHM de modo a tornar fácil a operação básica do robô por meio da execução de diversas rotinas pré-programadas no controlador via botões de comando e campos de entradas de dados implementados nas telas do IHM.

Além da integração dos sistemas, também foi feito o planejamento de trajetória utilizando a biblioteca SM3_CNC. O desenvolvimento do "planejador de trajetórias"consiste ler um trajetória de um sistema de coordenadas imaginário e converter essa informação para as coordenadas do robô cartesiano. O carregamento de um programa com a extensão (.txt) deve ser feito por meio do software SoMachine V4.1, esse foi o único modo encontrado para enviar o arquivo (.txt) para

controlador.

7.1 Trabalhos Futuros

De modo a melhorar a usabilidade do robô sugere-se o desenvolvimento de rotinas de leituras do código G a partir de um *pendrive* diretamente conectado a uma das portas USB disponíveis no controlador ou no IHM.

Para tornar apto o robô cartesiano a realizar o processo de soldagem será necessário o acoplamento da tocha de solda no seu efetuador terminal e integração do controlador com a fonte de soldagem. Também, faz-se necessário realizar o isolamento do ambiente onde está confinado o robô cartesiano para a segurança dos operadores durante a execução do processo de soldagem, instalar uma mesa de suporte da base de deposição com pelo menos 2 graus de liberdade de orientação e avaliar a necessidade de um sistema de arrefecimento para o controle de temperatura da peça em construção.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Moretti, G. H., (2017). Integração dos sistemas de controle de movimento e de temperatura de uma bancada de soldagem 3d. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-no, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 108p.

[2] Ahrens CH, Ferreira CV, Petrush G, Carvalho J, Santos LR, Silva LV, Volpato N. Prototipagem rápida: tecnologias e aplicações. Editora Blucher, São Paulo. 2007.

[3] ANDRADE, R. C., (2013). Desenvolvimento de software de fatiamento de sólidos tipo casca e geração de trajetórias para fabricação de peças por deposição de metal em camadas sucessivas utilizando o processo GMAW. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-no 20/2012, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 82p.

[4] Heyer, C. (2010). Human-Robot Interaction and Future Industrial Robotics Aplications. The 2010 IEEE/RSJ International Conference on Inteligent Robots Systems, (pp. 18-22). Taipei, Taiwan.

[5] LEONCIO DA SILVA NETO, P. (2018). Instalação, configuração e integração dos dispositivos de controle de um robô cartesiano. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 06, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 92 p.

[6] Modenesi, P. J.; Marques, P. V.; Santos, D. B. Introdução à Metalurgia da Soldagem. Belo Horizonte, janeiro de 2012. Disponível em: <http://demet.eng.ufmg.br/wp-content/uploads/2012/10/metalurgia.pdf>. Acesso em maio de 2019.

[7] Gomes, A. A.; Ruppenthal, J. E. Aspectos de higiene e segurança na soldagem com eletrodos revestidos em microempresas do tipo serralheria. XXII Encontro Nacional de Engenharia de Produção. Curitiba – PR, 23 a 25 de outubro de 2002.

[8] FERNANDES, M.O., (2016). Projeto de instalação de um robô cartesiano de 3 graus de liberdade. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-no06, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 85p.

[9] SCHNEIDER ELECTRIC. SoMachine, SoMachine - CANMotion with a Lexium Drive, Example Guide, abril de 2012. Disponível em: <http://www.schneider-electric.com>. Acesso em maio de 2019.

[10] SCHNEIDER ELECTRIC. CanMotion, Modicon LMC058 Motion Controller, Programming Guide, março de 2018. Disponível em: <http://www.schneider-electric.com>. Acesso em maio de 2019.

[11] SCHNEIDER ELECTRIC. Lexium Linear Motion, Linear axes and multi-axis systems, Catalogue, abril de 2011. Disponível em: <http://www.schneider-electric.com>. Acesso em maio de 2019.

[12] SCHNEIDER ELECTRIC. Lexium 32M, Servo Drive, User Guide, novembro, 2017. ISSN: 0198441113767.10. Disponível em: <http://www.schneider-electric.com> Acesso em maio, 2019.

[13] Da Silva, T.; Spohn, M.; Padilha, A. Uma análise comparativa entre os protocolos CANopen, DeviceNet e Smart Distributed System. Revista Brasileira de Computação Aplicada, v. 9, n. 1, p. 2-14, 28 maio 2017.

[14] Galloway, B.; Hancke, G. P. Introduction to industrial control networks.Communications SurveysTutorials, IEEE, v. 15, n. 2, p. 860–880, 2013. ISSN 1553-877X.

[15] Bosh, R. CAN specification version 2.0. 1991. Disponível em: <http://esd.cs.ucr.edu/webres/can20.pdf>. Acesso em: 20 jul. 2014.

[16] Brady, J. Networking with devicenet.The Computer Applications Journal, Circuit Cellar INK, v. 1, n. 98,p. 38–44, 1998. ISSN 0896-8985.

[17] Pfeiffer, O., Ayre, A., Keydel, C. (2008). Embedded Networking with CAN and CANopen. San Clement, CA: RTC Books.

[18] WEG, E. Comunicação CANopen. 2010. Disponível em: <http://ecatalog.weg.net/files/wegnet/ WEG-plc300-comunicacao-canopen-10000849433-manual-portugues-br.pdf>. Acesso em julho de 2019.

[19] WEG, E. Comunicação DeviceNet. 2008. Disponível em: <http://ecatalog.weg.net/files/wegnet/1-2195. pdf>. Acesso em julho de 2019.

[20] HONEYWELL. SDS specification. 1995. Disponível em: <http://www-csr.bessy.de/control/Hard/fieldbus/CAN/SDS/gs052103.pdf>. Acesso em julho de 2019.

[21] Corrigan, S. Introduction to the Controller Area Network (CAN). 2008. Disponível em: <http://www.ti.com/lit/an/sloa101b/sloa101b.pdf>. Acesso em: 20 de julho de 2019.

[22] Barbosa, L. R. G. Rede CAN. Disponível em: <https://manualzz.com/doc/21076378/rede-can-luiz-roberto-guimar%C3%A3es-barbosa–belo-horizonte>. Acesso em: 20 de julho de 2019.

[23] BOTERENBROOD, H.CANopen high-level protocol for CAN-bus. 2000. Disponível em: <https://www.nikhef.nl/pub/departments/ct/po/doc/CANopen30.pdf>. Acesso em: 17 de julho de 2019.

[24] NATIONAL, I.The Basic of CANopen. 2013. Disponível em: <http://www.ni.com/white-paper/14162/en/>.Acesso em: 23 maio 2017.

[25] ASM International. ASM Handbook - Vol. 16-Machining.3.ed.USA: ASM International, 1999.p.1-1089.

[26] H., Bruce. Getting Started With SoMachine. Disponível em: https://www.schneider-electric.com/resources/sites/SCHNEIDER_ELECTRIC/content/live/FAQS/288000/FA288633/en_US/Training%20Manual%20EN.pdf. Acesso em: 23 de junho de 2019.

[27] J. S., Tab. Motion with SoMachine. Training Manual, ACE University, 2012.

[28] Schneider Electric. Human/Machine Interface, HMI configuration software. Disponível em: <https://download.schneider-electric.com/files?p_enDocType=Catalog&p_File_Name=DIA5ED2130614EN.pdf&p_Doc_Ref=DIA5ED2130614EN>. Acesso em: 9 de junho de 2019.

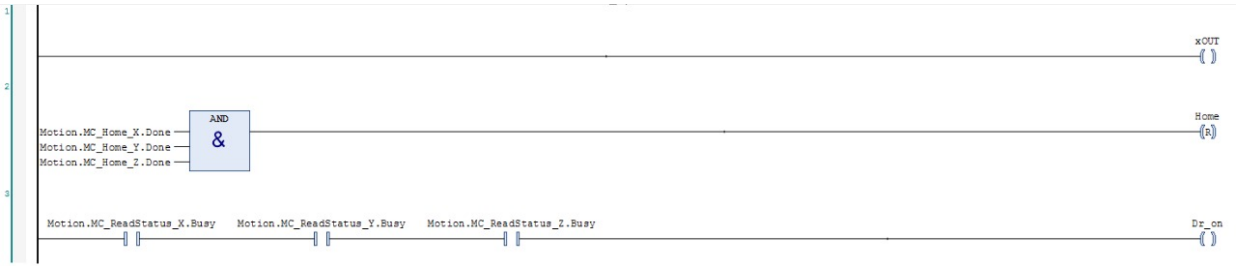
[29] Schneider Electric. Magelis - HMI GTO for Vijeo Designer - User Manual. Disponível em: <https://download.schneider-electric.com/files?p_enDocType=Userguide&p_File_Name=EIO0000001133.05.pdf&p_Doc_Ref=EIO0000001133>. Acesso em: 9 de junho de 2019.

[30] Lenze Automation GmbH. L-force | PLC Designer - SoftMotion. Disponível em: <http://www.schneider-electric.com>. Acesso em maio de 2019.

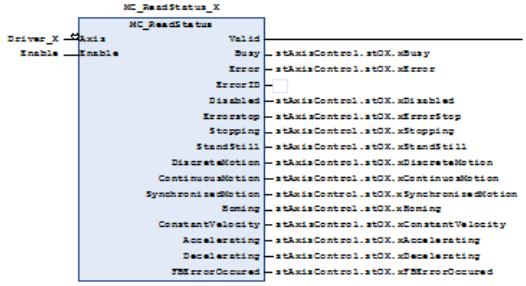
[31] 3S-Smart Software Solutions GmbH. CODESYS Motion + CNC. Disponível em: <https://www.codesys.com>. Acesso em junho de 2019.

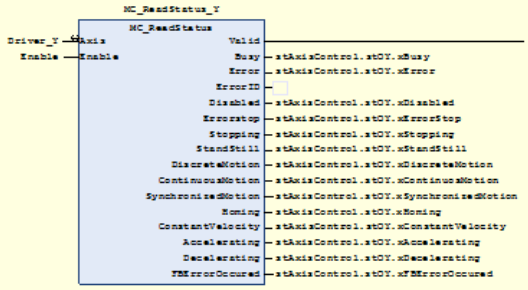
Apêndice A

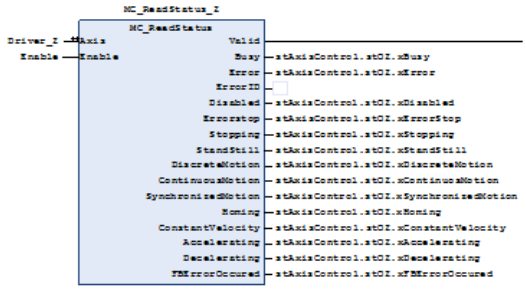
Programa Principal



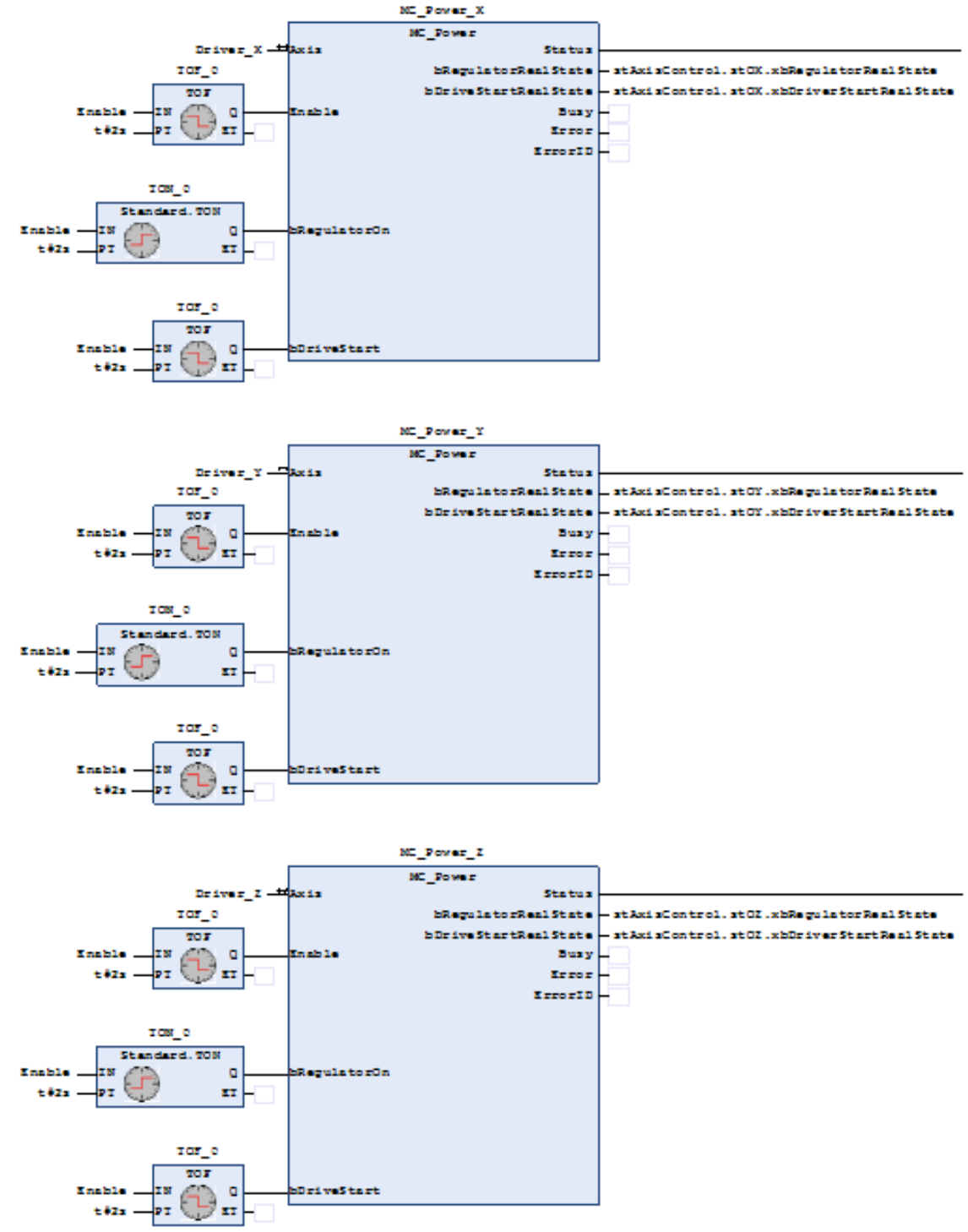
Os blocos MC_ReadStatus tem como função permitir ao usuário verificar vários parâmetros do Driver específico. Precisa ser acionado (ENABLE := TRUE) para habilitar o funcionamento de todos os outros blocos.



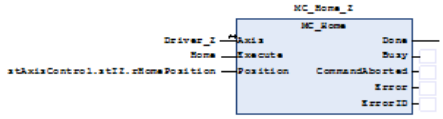


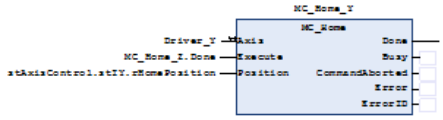


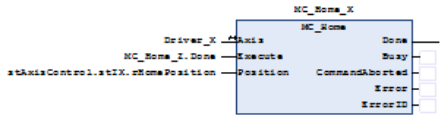
Os blocos MC_Power tem como função habilitar que os drivers entre em modo de operação remoto (Modo run).



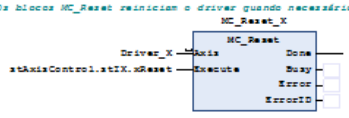
Os blocos MC_Home, quando acionados, ativam os motores para que os eixos se desloquem para o sensor de fim de curso do Limite Negativo do robô. Todas as variáveis de posição são zeradas após o bit Done se tornar verdadeiro.

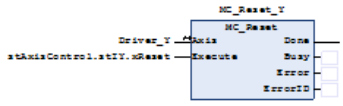


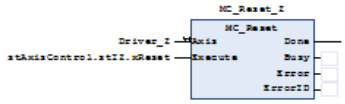




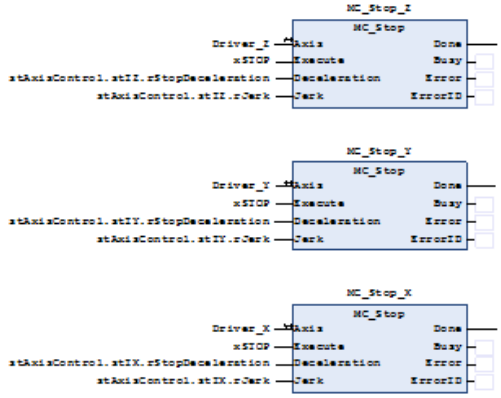
Os blocos MC_Reset reiniciam o driver quando necessário. Deve ser acionado quando há algum sinal de erro em algum dos eixos.



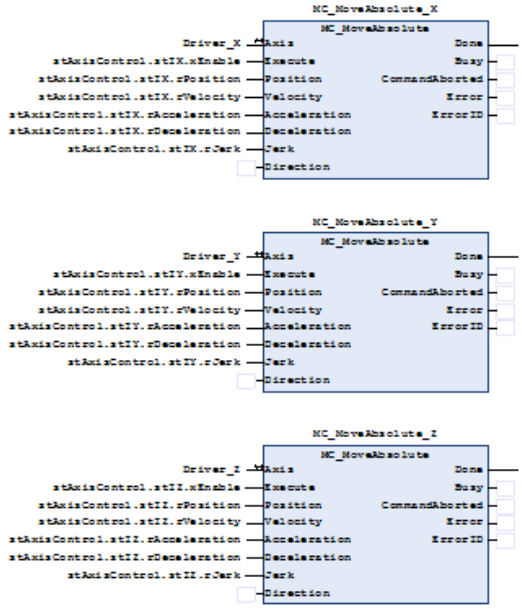




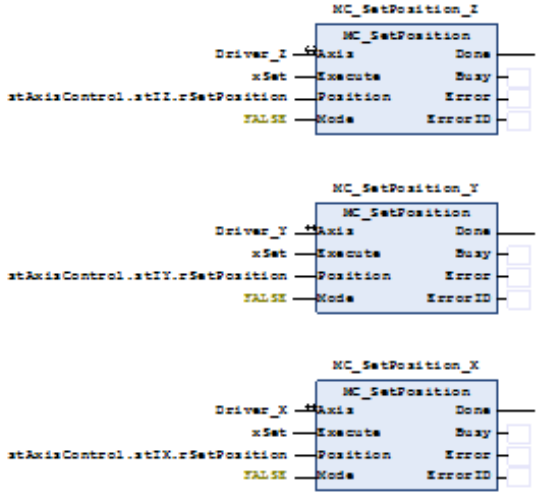
O bloco MC_Stop interrompe qualquer atividade que esteja sendo realizada por algum dos eixos, até mesmo movimentação. Quando acionado, ativa o sinal de erro ErrorStop.



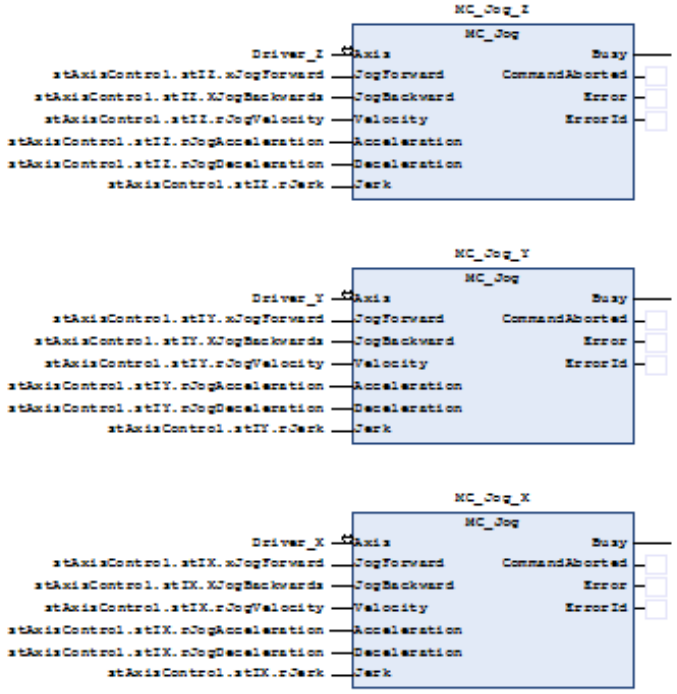
Os blocos MC_MoveAbsolute permitem movimentar o robô para alguma determinada posição dentro do seu limite de trabalho.



Os blocos de set Position permitem definir um novo sistema de coordenadas para o robô cartesiano forçando o zeramento das coordenadas.

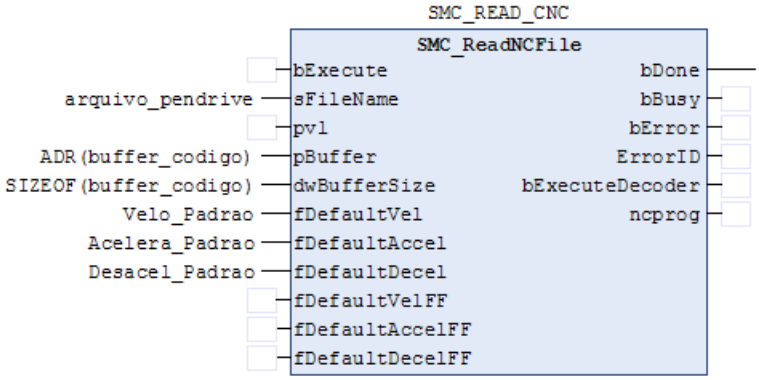


Os blocos de função MC_Jog são responsáveis pela movimentação lenta dos eixos XYZ.

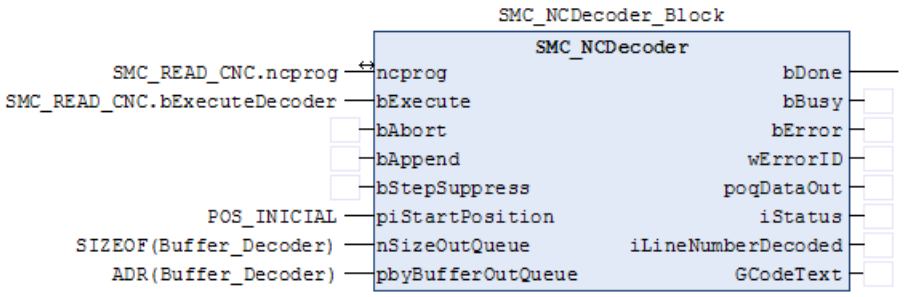


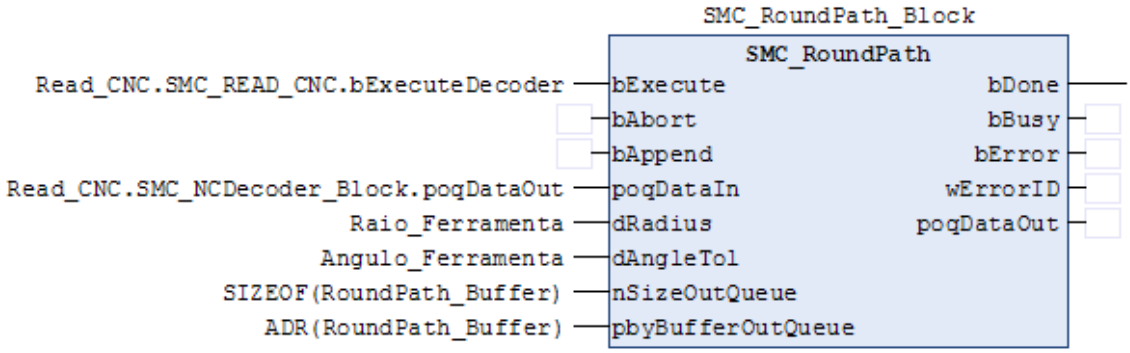
```
Essa linha faz a leitura da posição do driver X.

   Driver_X.fActPosition ——— stAxisControl.stDX.rActPosition

Essa linha faz a leitura da posição do driver Y.

   Driver_Y.fActPosition ——— stAxisControl.stDY.rActPosition

Essa linha faz a leitura da posição do driver Z.

   Driver_Z.fActPosition ——— stAxisControl.stDZ.rActPosition

Essa linha faz a leitura da velocidade do driver X.

   Driver_X.fActVelocity ——— stAxisControl.stDX.rActVelocity

Essa linha faz a leitura da posição do driver Y.

   Driver_Y.fActVelocity ——— stAxisControl.stDY.rActVelocity

Essa linha faz a leitura da posição do driver Z.

   Driver_Z.fActVelocity ——— stAxisControl.stDZ.rActVelocity
```

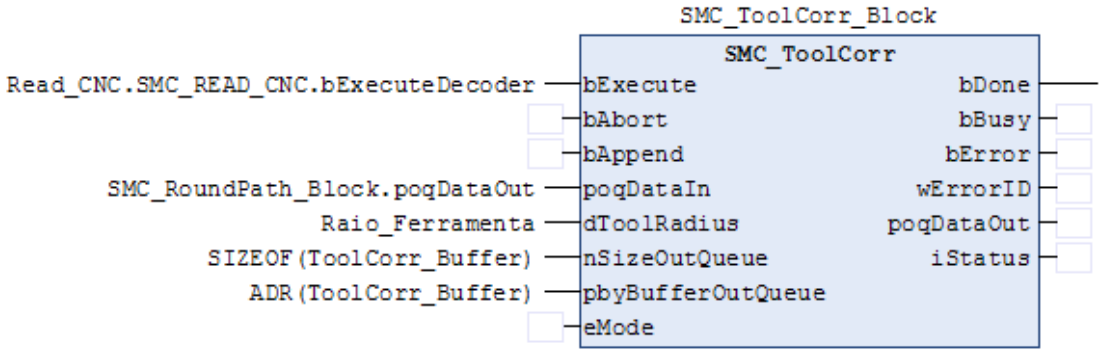
Esse bloco é usado para fazer a leitura de um arquivo de código G que está na memoria do controlador.

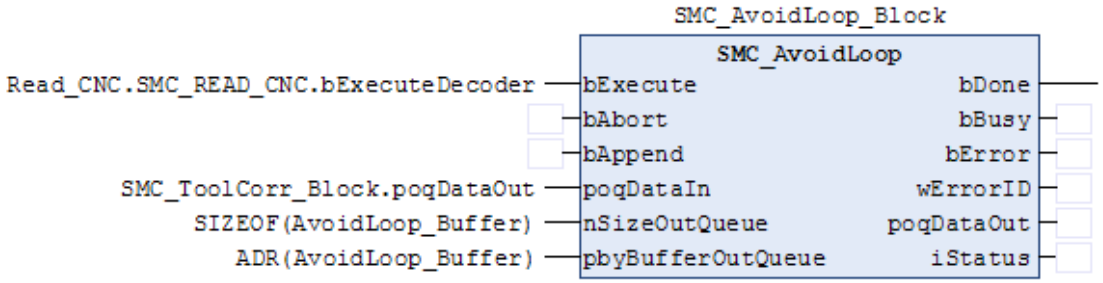


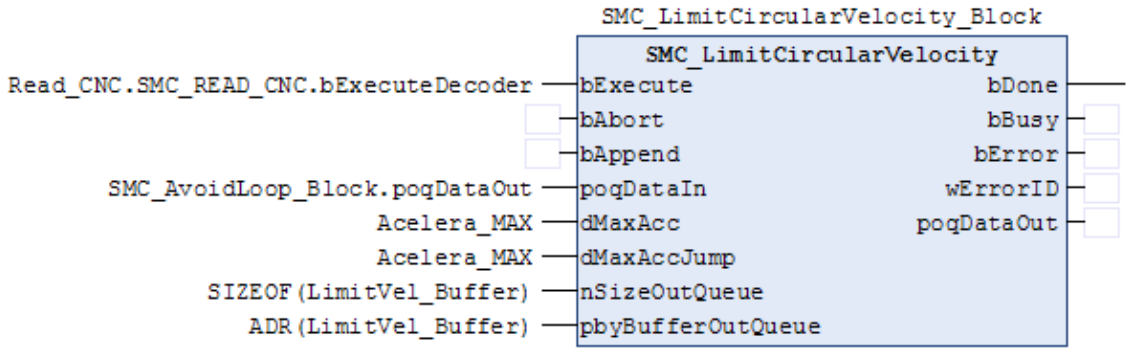
Esse bloco é usado para fazer a decodificação do código G lido.



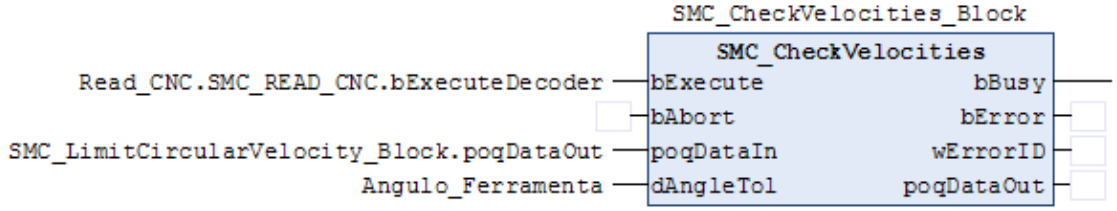

SMC_RoundPath_Block
SMC_RoundPath
Read_CNC.SMC_READ_CNC.bExecuteDecoder
bExecute
bDone
bAbort
bBusy
bAppend
bError
Read_CNC.SMC_NCDecoder_Block.poqDataOut
poqDataIn
wErrorID
Raio_Ferramenta
dRadius
poqDataOut
Angulo_Ferramenta
dAngleTol
SIZEOF(RoundPath_Buffer)
nSizeOutQueue
ADR(RoundPath_Buffer)
pbyBufferOutQueue

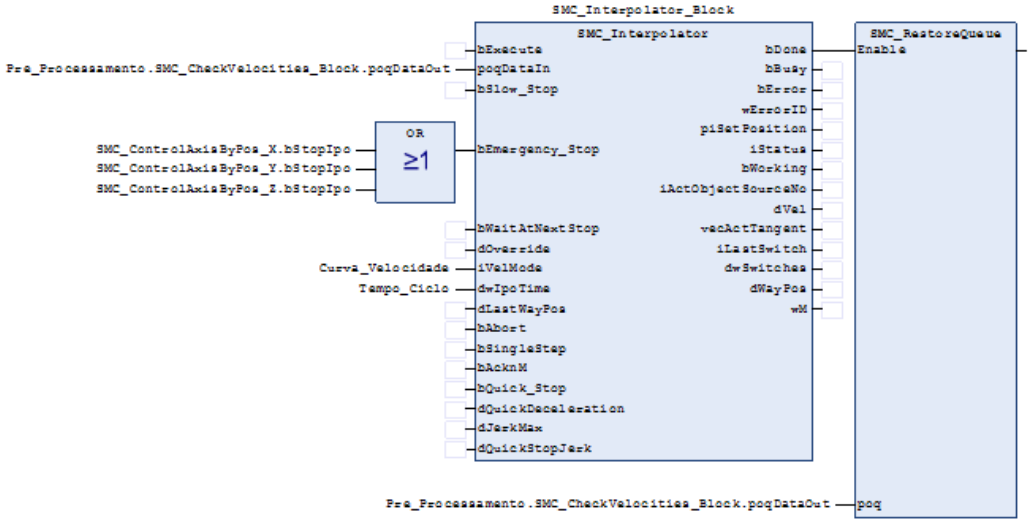

SMC_ToolCorr_Block
SMC_ToolCorr
Read_CNC.SMC_READ_CNC.bExecuteDecoder
bExecute
bDone
bAbort
bBusy
bAppend
bError
SMC_RoundPath_Block.poqDataOut
poqDataIn
wErrorID
Raio_Ferramenta
dToolRadius
poqDataOut
SIZEOF(ToolCorr_Buffer)
nSizeOutQueue
iStatus
ADR(ToolCorr_Buffer)
pbyBufferOutQueue
eMode

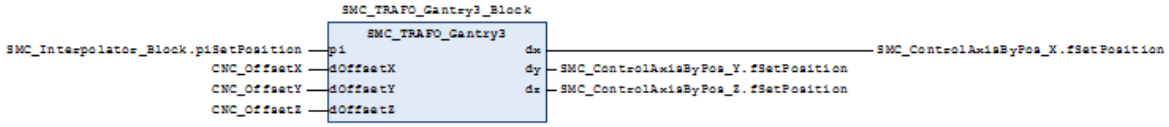

SMC_AvoidLoop_Block
SMC_AvoidLoop
Read_CNC.SMC_READ_CNC.bExecuteDecoder
bExecute
bDone
bAbort
bBusy
bAppend
bError
SMC_ToolCorr_Block.poqDataOut
poqDataIn
wErrorID
SIZEOF(AvoidLoop_Buffer)
nSizeOutQueue
poqDataOut
ADR(AvoidLoop_Buffer)
pbyBufferOutQueue
iStatus

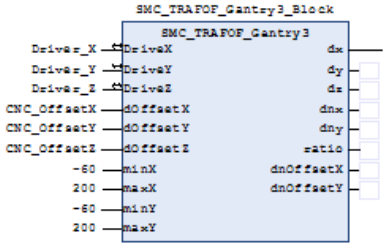


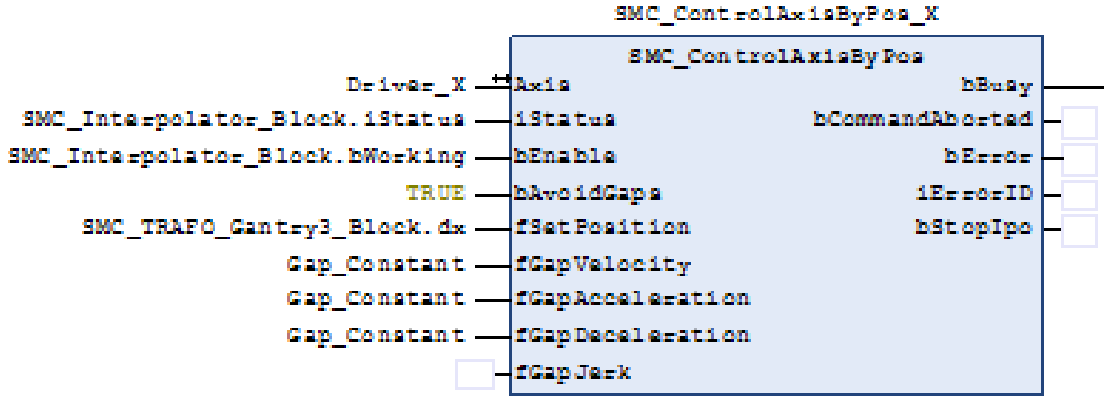
Esse bloco é utilizado para fazer a checagem da velocidade dos pontos no código G.

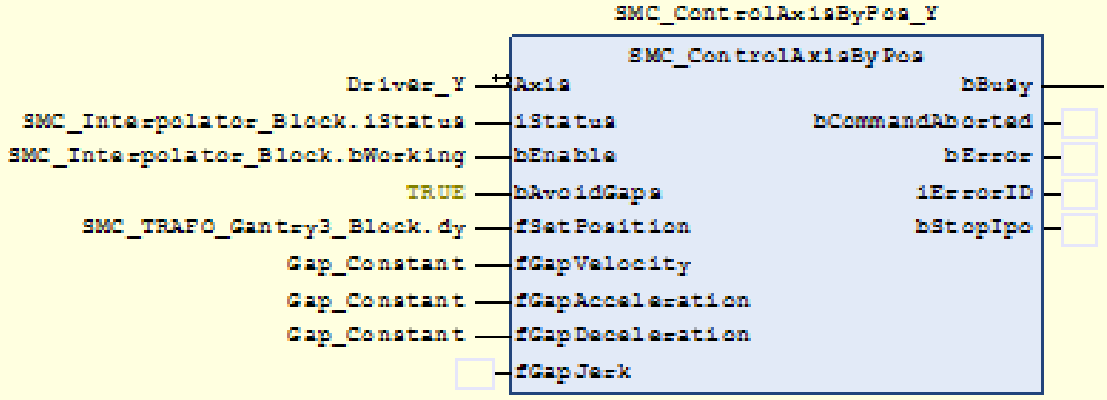


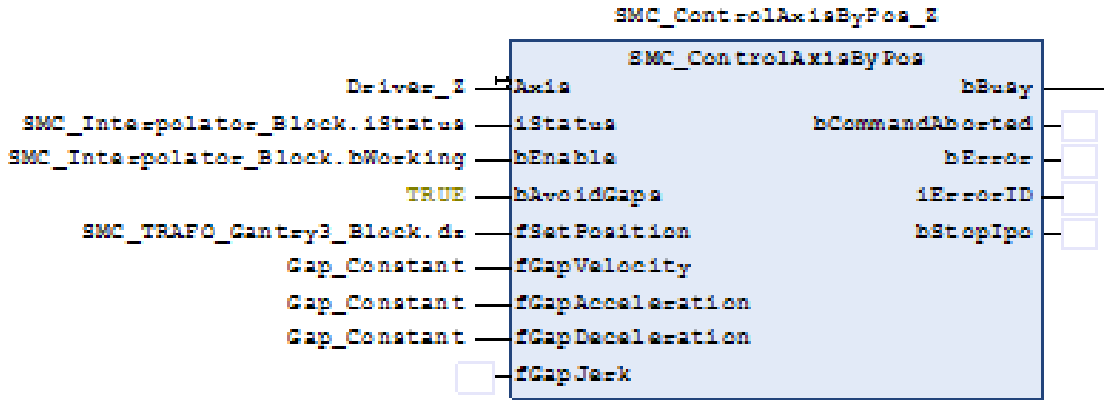







SMC_ControlAxisByPos_X
SMC_ControlAxisByPos
Driver_X
Axis
bBusy
SMC_Interpolator_Block.iStatus
iStatus
bCommandAborted
SMC_Interpolator_Block.bWorking
bEnable
bError
TRUE
bAvoidGaps
iErrorID
SMC_TRAFO_Gantry3_Block.dx
fSetPosition
bStopIpo
Gap_Constant
fGapVelocity
Gap_Constant
fGapAcceleration
Gap_Constant
fGapDeceleration
fGapJerk


SMC_ControlAxisByPos_Y
SMC_ControlAxisByPos
Driver_Y
Axis
bBusy
SMC_Interpolator_Block.iStatus
iStatus
bCommandAborted
SMC_Interpolator_Block.bWorking
bEnable
bError
TRUE
bAvoidGaps
iErrorID
SMC_TRAFO_Gantry3_Block.dy
fSetPosition
bStopIpo
Gap_Constant
fGapVelocity
Gap_Constant
fGapAcceleration
Gap_Constant
fGapDeceleration
fGapJerk


SMC_ControlAxisByPos_Z
SMC_ControlAxisByPos
Driver_Z
Axis
bBusy
SMC_Interpolator_Block.iStatus
iStatus
bCommandAborted
SMC_Interpolator_Block.bWorking
bEnable
bError
TRUE
bAvoidGaps
iErrorID
SMC_TRAFO_Gantry3_Block.dz
fSetPosition
bStopIpo
Gap_Constant
fGapVelocity
Gap_Constant
fGapAcceleration
Gap_Constant
fGapDeceleration
fGapJerk