MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

USO DO DIAGRAMA SEQUENCIAL FUNCIONAL COMO LINGUAGEM DE PROGRAMAÇÃO PARA UM ROBÔ CÍLINDRICO DE 5 GRAUS DE LIBERDADE ACIONADO PNEUMATICAMENTE

por

Pablo Leonardelli

Dissertação para obtenção do Título de
Mestre em Engenharia

Porto Alegre, Junho de 2015

USO DO DIAGRAMA SEQUENCIAL FUNCIONAL COMO LINGUAGEM DE PROGRAMAÇÃO PARA UM ROBÔ CÍLINDRICO DE 5 GRAUS DE LIBERDADE ACIONADO PNEUMATICAMENTE

por

Pablo Leonardelli
Graduado em Engenharia Elétrica

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Mecânica, da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como parte dos requisitos necessários para a obtenção do Título de

Mestre em Engenharia

Área de Concentração: Processos de Fabricação

Orientador: Prof. Dr. Eduardo André Perondi

Aprovada por:

Prof. Dr. Flávio José Lorini, PROMEC / UFRGS

Prof. Dr. Fabiano Disconsi Wildner, PROMEC / UFRGS

Prof. Dr. Marcelo Götz, DELET/UFRGS

Prof. Dr. Luiz Alberto Oliveira Rocha
Coordenador do PROMEC

Porto Alegre, 16, Junho de 2016.

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais por toda motivação para alcançar meus objetivos na vida desde sempre.

A minha namorada pelo apoio e compreensão na fase final do trabalho.

Ao Prof. Eduardo André Perondi pela disposição e orientação de qualidade durante todo trabalho.

Aos colegas do LAMECC pela amizade, companheirismo e apoio.

Ao LAMECC por me acolher e disponibilizar toda estrutura para desenvolvimento do trabalho.

Aos colegas Leonardo Gutérrez e Vitor Valente pela amizade, apoio e ajuda no final do trabalho.

Ao CNPq pelo apoio financeiro.

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de uma estratégia de programação para um robô de cinco graus de liberdade com acionamento pneumático. A proposta para tal estratégia de programação utiliza como base a linguagem SFC (Sequential Function Chart) normatizada pela IEC 61131-3. A principal característica deste tipo de linguagem é a simplicidade na integração com diversos elementos presentes em ambiente fabril, juntamente a garantia do sequenciamento das ações e a facilidade de programação. O estudo foi realizado em três etapas: a primeira, destina-se à criação de sub-rotinas em linguagem SFC para movimentação ponto a ponto, *pick and place*, e paletização. Desta forma, através da definição de alguns dados de entrada, é possível reprogramar o robô de forma gráfica e intuitiva; a segunda etapa do estudo constituiu na criação de um Programa Tradutor em linguagem baseada em *scripts* de Matlab que, através de um servidor OPC (*Ole for Process Control*), faz a interpretação do programa em linguagem SFC e o traduz para a linguagem do sistema de controle do robô; já, a última etapa destina-se à realização de testes utilizando um CLP *Compact Logix* da AllenBradley em conjunto com o software de programação RSLogix 5000, o software Matlab e o sistema de controle do robô pneumático. A partir dos resultados, Conclui-se que a aplicação e utilização este tipo de programação para tarefas de movimentação de robôs é plenamente viável, o que pode vir a simplificar as etapas de programação, e ampliando a integração entre os diversos sistemas fabris, na medida em que os seus elementos poderão trocar facilmente informações necessárias à automação.

Palavras chave: Robótica; SFC; Controlador Lógico Programável; OPC; IEC 61131-3.

ABSTRACT

The present study has as main goal to present a differentiated form of programming for a prototype of a robot of five degrees of freedom with pneumatic drive. This program is based on the language SFC (Sequential Function Chart) standardized by IEC 61131-3. The main feature of this type of language is simplicity in integration with various elements present in the manufacturing environment, ensuring the sequencing of actions and ease of programming. The system used as a test bench consists of a pneumatic robot which currently control actions are carried out through specific programming routines combined with dedicated control boards, working with Matlab software. The study was conducted in three stages: the first, for creating subroutines in SFC language to linear movement, pick and place movement and palletizing movement, thus, by setting some input data it is possible to reprogram the robot for tasks in a graphical and intuitive way; the second stage of the study consisted in creating a translator program in Matlab language based on scripts that, through an OPC server (Ole for Process Control), interpreters the program in SFC language and translates it into the language of the control system robot; the last step was intended for testing this programming approach by using a PLC Compact Logix from Allen-Bradley in conjunction with RSLogix 5000 programming software, Matlab and the control system of the pneumatic robot. It was concluded that the implementation and use of this type of programming for robot handling tasks are both feasible. It simplifies the programming steps and enhances the integration between the various manufacturing systems, since the elements could directly exchange information, because they are in the same language.

Keywords: Robotics; Sequential Function Chart; PLC; OPC; IEC 61131-3.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
GDL	Graus de Liberdade
PROMEC	Programa de Pós-Graduação em Engenharia Mecânica
UFRGS	Universidade Federal do Rio Grande do Sul
LAMECC	Laboratório de Mecatrônica e Controle
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IRF	International Federation of Robotics
OPC	Ole for Process Control
CLP	Controlador Lógico Programável
SFC	Sequential Function Chart
RCP5	Robô Cilíndrico Pneumático com 5 Graus de Liberdade

1 INTRODUÇÃO

No ambiente fabril, juntamente com os robôs industriais, os controladores de lógica programável, mais conhecidos como CLP (Controlador Lógico Programável), desempenham um papel fundamental na produção, sendo responsáveis pela automação da maioria dos processos de manufatura. Para sua efetiva operação na produção, estes sistemas devem apresentar comportamento adequado nas diversas situações operacionais a que são submetidos. Essa característica depende não somente do sistema físico em questão, mas também de sua programação. Com base nisto, surgiu a padronização das linguagens de controladores lógicos programáveis com a norma IEC 61131-3, que estabeleceu padrões para cinco tipo de linguagens: blocos sequenciais (SFC); Ladder (LD); Blocos Funcionais, Linguagem Estruturada e Lista de Instruções.

A programação via SFC baseia-se na representação gráfica do comportamento da parte de comando de um sistema automatizado. Esta representação é constituída por uma simbologia gráfica com arcos orientados que interligam etapas e transições, caracterizadas pela relação lógica combinacional das variáveis de entrada e saída da parte do comando lógico programado relacionadas às ações e regras de evolução que definem formalmente o comportamento dinâmico dos elementos físicos comandados [Silveira e Santos, 2002].

Este trabalho propõe a criação de uma biblioteca com rotinas baseadas na linguagem de programação por Diagrama de Blocos Sequenciais (SFC) para permitir a programação de um robô cilíndrico de 5 graus de liberdade desenvolvido no Laboratório de Mecatrônica e Controle (LAMECC) da Universidade Federal do Rio Grande do Sul. Essa estratégia, quando comparada com as de programação usualmente aplicada a robôs comerciais, apresenta vantagem de permitir que os programadores de CLP com conhecimento em SFC possam também programar manipuladores robóticos sem a necessidade de se aprofundarem nas linguagens específicas de programação de robôs. Além disso, baseando-se na solução para o robô de 5 graus de liberdade, propõe-se no âmbito do presente trabalho uma metodologia genérica de programação baseada em SFC para programar robôs industriais, tornando a programação de robôs menos restrita aos seus fabricantes, deixando, assim, mais simples e acessível a integração dos robôs com os demais elementos de produção.

1.1 Descrição do Problema e Motivação

Dados recentes do IFR (*International Federal of Robotic)*, 2016, indicam que cerca de 240 mil unidades de robôs foram colocadas na indústria mundial no ano de 2015. Hoje existem diversos fabricantes de robôs industriais e, entre os mais importantes, pode-se citar ABB, Stäubli, Fanuc, Epson, Kuka e Hyundai. De forma geral, cada um executa o desenvolvimento completo do sistema, incluindo o controlador e o software de programação do robô. Assim como não há uma padronização sobre qual linguagem de programação deve ser utilizada para um dado robô, cada fabricante opta por desenvolver a sua própria linguagem para programação. A Tabela 1.1 apresenta os principais fabricantes de robôs industriais e as linguagens utilizadas para programá-los.

Tabela 1.1 – Principais fabricantes de robôs e linguagens utilizadas.

Fabricante	**Linguagem**
ABB	ARLA, RAPID
Epson	SPEL
Stäubli	VAL3
FANUC	KAREL
Motoman	Inform 1, Inform 2
Comau	PDL2
Samsung	FARAL II
Kawasaki	AS
Adept	V+

Fonte: Adaptado de Gong, 1998.

Esta grande diversificação das linguagens de programação para robôs gera demanda de suporte para seus fabricantes, pois muitas vezes é difícil encontrar no mercado profissionais devidamente capacitados, tornando mais custoso para as empresas a utilização dos seus robôs. Outro problema que ocorre neste tipo de solução é o tempo envolvido nas atividades de programação. Quando se trata de robótica industrial, sabe-se que um comportamento seguro quanto a falhas, tanto mecânicas, quanto de programação, é um dos principais requisitos para a

operação industrial. Como será visto mais adiante, na Seção 2.5, essa é exatamente uma das características mais atrativas da programação por SFC, já que a mesma é baseada em lógica sequencial intrinsecamente intertravada, garantindo alta confiabilidade na execução das ações sequencializadas. Assim, além de resolver o problema específico de viabilizar a programação de um robô específico em desenvolvimento, uma motivação importante do presente trabalho consiste na necessidade de padronização das linguagens de programação de robôs industriais, de forma que um profissional capacitado em uma linguagem possa facilmente programar diferentes robôs e na possibilidade de utilizar para tanto uma linguagem de programação de CLP (SFC) para aplicação em robótica. Dessa forma, o profissional poderá programar o robô e os demais componentes de uma célula de manufatura por meio de uma mesma linguagem e ambiente de programação.

1.2 Objetivos

O principal objetivo deste trabalho é criar uma estratégia de programação em Linguagem Blocos Sequenciais para programar um robô cilíndrico de 5 graus de liberdade. Essa estratégia de programação, quando generalizada para outras aplicações, visa a tornar mais simples o processo de programação de robôs industriais. Essa estratégia deverá permitir que o usuário possa programar comandos de movimento no espaço de trabalho, atuação do efetuador e fazer uso de tecnologias adicionais, tais como sistema de visão e de segurança e integração com células de manufatura. Os objetivos específicos são:

- Criar sub-rotinas em linguagem de CLP associadas a comandos de movimentação do robô, de forma que as mesmas possam ser importadas e exportadas pelo programador.
- Desenvolver, testar e validar, através de ensaios, um programa interpretador e conversor de linguagem SFC para a linguagem do controlador do robô utilizado nos testes.
- Propor uma comunicação entre o CLP e o controlador do robô através de um protocolo de comunicação industrial existente no mercado.

1.3 Organização do Trabalho

O trabalho está organizado em 6 capítulos. No segundo capítulo é apresentado o estado da arte do desenvolvimento de linguagens de programação de robôs e são apresentados os conceitos teóricos necessários para o entendimento dos assuntos apresentados no trabalho que segue. Já no terceiro capítulo é apresentada a estratégia proposta para a programação de robôs utilizando a linguagem SFC, enquanto que no capítulo 4 é apresentada a aplicação desta estratégia na forma de um estudo de caso utilizando um robô de 5 graus de liberdade acionado pneumaticamente. No Capítulo 5 são apresentados os resultados para dois exemplos práticos, e, finalmente, no Capítulo 6 são apresentadas as conclusões do presente trabalho e propostas para futuros desenvolvimentos.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentados alguns conceitos básicos para melhor entendimento do trabalho desenvolvido. Primeiramente, será apresentado o estado da arte, descrevendo e comentando os principais trabalhos atualmente desenvolvidos na área. Após, serão abordados os assuntos pertinentes ao contexto do trabalho envolvendo robôs industriais tais como seus tipos de linguagens de programação, lógica de controle discreto e binária, e linguagens de diagramas de blocos sequenciais, mais especificamente o *Sequential Function Chart* (SFC).

2.1 Estado da Arte

O estado da arte tem como objetivo contextualizar o presente estudo no cenário atual da pesquisa e desenvolvimento nas áreas de interesse do trabalho. Assim, temas como programação em SFC, a norma IEC 61131-3, robótica e CLP serão focados. Além disso, serão apresentados os principais trabalhos desenvolvidos até o presente momento com relação ao robô pneumático de 5 graus de liberdade, objeto do estudo de caso. Ao longo deste capítulo será possível perceber que alguns estudos propõem o controle de robôs diretamente via CLP, já, outros utilizam técnicas de tradução baseadas em texto ou endereços de memória, acessados fisicamente. Não foram encontrados estudos que fazem uso direto do SFC para programar robôs, sendo, portanto, essa a principal contribuição do trabalho.

2.1.1 CLP, Robótica e IEC 61131-3 (SFC)

As literaturas sobre uso de CLP para controle de robôs é escassa, o mesmo ocorrendo com relação ao uso de SFC em sistemas abertos ou microcontrolados. Em termos de sistemas de controle não proprietários, o trabalho, cronologicamente mais antigo encontrado é o de EL DIN, 1996, o qual propõe um hardware para motor de passo de alta eficiência para manipuladores robóticos controlado por CLP. Eram utilizadas as portas de saída do CLP para indicar a ação e direção dos atuadores rotacionais. Já, Chung, 1998, propôs uma abordagem de baixo custo para fazer uma interface PC-PLC-Robô. Foi proposto um hardware que pudesse converter saídas de tensão de um PLC para a comunicação RS232 do robô, através de uma engenharia reversa no *Teach Pendant* do mesmo.

Ivanescu et al, 2007, fazem uso do SFC como estratégia de programação para micro controladores ATmega 8515 com aplicação em uma máquina CNC, conectada ao micro-

controlador via comunicação Interface Serial Periférica (SPI). Em seu estudo, são criadas funções em linguagem C correspondentes a elementos do SFC como passos, transições e ações. A tradução dos elementos em SFC foi feita utilizando apontadores de funções referentes a endereços da memória do CLP.

Já, Bonfe et al, 2009, realizam um controle de um robô de cadeia fechada similar ao robô comercial IRB 660 da empresa ABB, que possui 4 Graus de Liberdade. Para o controle do robô, foi utilizado um CLP em conjunto com servo drivers controlando 4 motores brushless, com programação foi baseada em texto estruturado. Como resultado, foi possível gerar trajetórias de *pick and place* para tal robô.

Neto et al, 2010, desenvolvem uma interface homem-robô baseada em Projeto Auxiliado por Computador (CAD) para programar robôs de maneira *off-line*. Utilizando uma Interface de Programação de Aplicações (API) do software Autodesk Inventor, é possível que o usuário crie modelos geométricos do robô e que o sistema capture as etiquetas (*tag points*) referentes a movimentação de um ponto do efetuador. Com este sistema foi possível gerar código de trajetória para um robô industrial IRB 2400 da empresa ABB, de maneira mais rápida do que o método tradicional que utiliza o *Teach Pendant*.

Subbaraman et al, 2010, propõem uma estratégia de conversão da linguagem *Ladder* para a Linguagem de Descrição de Hardware (VHDL), criando uma interface gráfica para usuário (GUI) compatível com o *Ladder* e armazenando as variáveis referentes a cada bloco de cada linha da programação em um banco de dados. Tais dados são sintetizados pelo mesmo programa e convertidos para a linguagem VHDL. Como resultado, os autores conseguiram gerar código VHDL, utilizando o *Ladder*.

Pereira et al, 2011, propõem a tradução de códigos escritos em linguagem de lista de instruções baseadas na norma IEC 61131-3 em código compatível com a linguagem de comandos do Matlab (código “m”) para emular um CLP em simulações de algoritmos de controle processados no ambiente Matlab/Simulink®.

Kummar e Sudarshan, 2011, apresentam uma técnica de programação de robôs baseada em demonstração, onde o robô móvel, simulado através do software WEBOTS, é controlado pelo usuário por uma trajetória enquanto os pontos da mesma são adquiridos e armazenados para, posteriormente, e, de maneira automatizada, o robô repetir a mesma trajetória.

Thomas et al, 2013, apresentam uma nova linguagem de programação para robôs, chamada de *LightRocks* (*Light Weight Robot Coding for Skills*), que consiste em uma linguagem

de domínio específico, baseada na Linguagem de Modelagem Unificada UML/P de diagrama de estados, oferecendo 3 níveis diferentes de abstração: a primeira em nível avançado de conhecimento de programação para robôs; a segunda em nível de designação de tarefas, para um estágio de conhecimento técnico intermediário; e, por fim, em um nível de processo, designado para o operário industrial. Como resultado, foi possível gerar código para execução de tarefas de um robô cartesiano com sucesso.

Skulavik et al, 2013, implementaram um sistema de controle do tipo *Fuzzy* em um braço robótico de 3GDL utilizando como linguagem de programação o *Ladder*. O sistema foi implementado utilizando o pacote de software *Siemens Fuzzy Control* ++ e um CLP *Siemens Sematic S7-300*, e se mostrou satisfatório dentro dos critérios requeridos de compensação de massa tanto no modo manual quando automático.

Como estudos mais recentes destaca-se o trabalho de Wu et al, 2015, que desenvolveram um sistema de programação *off-line* baseado em CAD chamado de RobSim. O RobSim foi desenvolvido no Microsoft Visual Studio 2010 para funcionar como uma ferramenta de *add-on* para o software CAD SolidWorks. No mesmo ambiente (*SolidWorks*), foi possível criar, importar ou modificar um objeto no modo convencional de desenho e gerar ou modificar trajetórias de um robô para usiná-lo.

Secil et al, 2015, apresentam um novo método para programar manipuladores industriais, baseada em três estágios: o primeiro é a modelagem da superfície de uma peça industrial através de um *scanner* a laser montado na ponta do efetuador do robô; o segundo consiste na visualização do modelo 3D pelo operador (utilizando a tecnologia PCL-*Point Cloud Library*), onde o mesmo seleciona os pontos de operação; o último estágio é a obtenção do plano de movimentação com base nos pontos selecionados e mover o efetuador através da trajetória gerada. Como estudo de caso foi utilizado o robô de 6 GDL RS005L (Kawasaki Robotics Inc.).

2.1.2 Robô Pneumático de 5 Graus de Liberdade

Quanto à configuração das juntas, o robô pode ser classificado com RPP:RR, ou seja, possui uma junta rotacional em sua base (R), seguido por duas juntas prismáticas e ortogonais (P), dispostas vertical e horizontalmente respectivamente. O punho possui duas juntas rotacionais referentes aos movimentos de arfagem (*pitch*) e rolagem (*roll*).

A estrutura do robô é ilustrada na Figura 2.1 [Sarmanho, 2014].

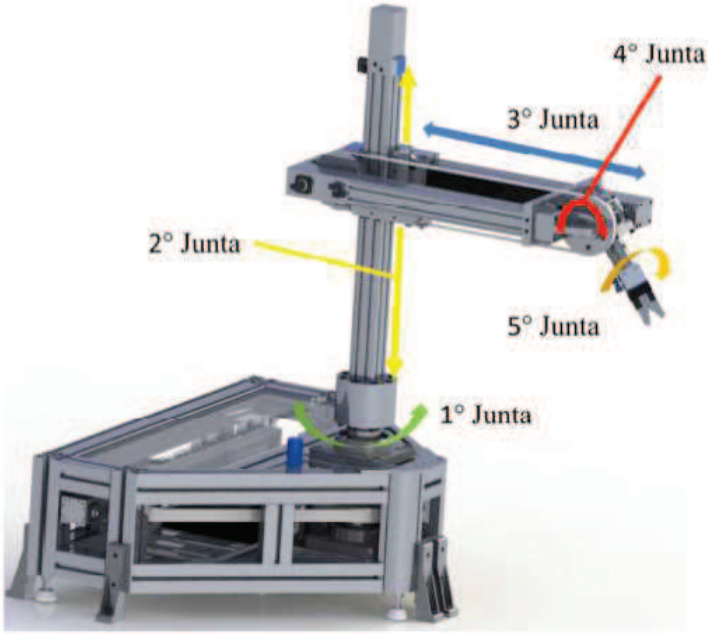

Figura 2.1 – Geometria do robô cilíndrico de 5GDL.

Fonte: Sarmanho, 2014.

A seguir, são apresentadas as descrições dos principais estudos realizados no âmbito do seu desenvolvimento.

Em Rijo, 2013, é apresentado um estudo referente à viabilidade técnica, projeto, construção e avaliação da estrutura mecânica onde se encontram os elementos mecânicos referentes ao acionamento do primeiro grau de liberdade do robô pneumático. É apresentada a simulação realizada por meio do software ANSYS® (modelagem e simulação estrutural por elementos finitos) para as magnitudes de possíveis deformações da estrutura sob condições estáticas e dinâmicas. Rijo, ainda, apresenta um algoritmo de controle através da realimentação de estados (PVA) com parametrização da inércia dos demais elos do manipulador em relação ao primeiro grau de liberdade. Como resultado, o autor obtém erros médios de translação do atuador entre 4,5 a 7 mm, concluindo que a compensação de inercia é eficiente para determinadas aplicações.

Missiagia, 2014, apresenta um conceito de geração de trajetória através da aproximação de pontos por *splines* compostas por polinômios de sétima ordem aplicado ao robô pneumático RCP5. O método proposto para a geração de *splines* possibilita, através do ajuste de parâmetros que podem variar de acordo com exigência de cada aplicação, a obtenção de curvas no espaço

das juntas com valores otimizados de *jerk*, aceleração ou velocidade. Trajetórias projetadas com baixos valores de *jerk* são desejáveis para diversas aplicações robóticas, pois proporcionam às juntas movimento similar ao das articulações humanas, apresentando movimentação suave, diminuindo o desgaste e as vibrações do robô [Piazzi e Visioli, 2000]. Para aplicação na geração de trajetórias para o robô, a interpolação dos pontos é realizada no espaço dos atuadores a fim de fornecer ao controlador as curvas de referência para posição, velocidade, aceleração e *jerk*. O sistema desenvolvido tem como dados de entrada as coordenadas $x, y\ e\ z$ e os ângulos $\theta_4 e\ \theta_5$ dos pontos por onde o efetuador do robô deve passar, e gera a trajetória através das *splines* percorrendo pontos chave. Por fim, são gerados vetores das coordenadas no espaço de juntas incluindo cada ponto a ser percorrido.

Sarmanho, 2014, propõe uma técnica de controle baseada na Lei de Slotine e Li (Torque Calculado) acrescentando uma parcela de compensação de atrito e utilizando o RCP5 como objeto de testes experimentais. O algoritmo foi desenvolvido em Matlab ®, mais precisamente na ferramenta gráfica Simulink® e compilado para ser processado em tempo real na placa de algoritmos de controle DS1104 da dSPACE®. O sistema faz o controle com base nos dados de pressão e posição de cada atuador e nos vetores de trajetória gerados com o conceito de *splines* do trabalho de Missiaggia, 2014. Como resultado, Sarmanho, 2014, apresentou erros de posição abaixo de 10 mm no 2º e 3º grau de liberdade e abaixo de 0,076 radianos no 1º, 4º e 5º grau de liberdade.

Oliveira, 2015, desenvolveu um sistema de visão computacional para identificação da posição e orientação de peças, gerando coordenadas de uma trajetória de *pick and place* que o robô deve percorrer para atingir a localização e a orientação para manipulá-las. Para realizar o reconhecimento da posição e orientação das peças a serem manipuladas pelo robô, utiliza o método de visão computacional aplicados à uma imagem de acordo com o procedimento descrito por Grassi, 2005, baseado no cálculo dos momentos da imagem e comparação com uma imagem padrão como o faz Hu, 1992. O algoritmo desenvolvido fornece a posição no plano xy do centro de massa do objeto e o ângulo de giro da garra do manipulador necessário para capturar corretamente o objeto.

No Capítulo 4 é feita uma descrição completa do robô, incluindo aspectos como volume de trabalho, cinemática, dinâmica e controle, dentre outros.

2.2 Programação de Robôs Industriais

Na área de automação industrial, muito esforço tem sido dispendido visando a padronizar, desde conexões mecânicas, até sistemas de comunicação digital e, mesmo, linguagens de programação. Essa última padronização é altamente vantajosa, permitindo a migração rápida de programas entre plataformas e fazendo com que um dado aprendizado de linguagem seja útil em uma grande quantidade de equipamentos. Neste contexto, é importante destacar os Controladores Lógicos Programáveis (CLP) que usam, por exemplo, linguagens LADDER e SFC, bem como os Comandos Numéricos Computadorizados (CNC), que empregam em sua maioria a linguagem conhecida popularmente como Código G. Apesar da padronização, há, evidentemente, pequenas diferenças entre equipamentos associadas às características particulares de cada um, que acabam levando à necessidade de realização de adaptações da linguagem utilizada para cada caso [Chaves, 2013].

A programação de um robô pode ser definida como o processo mediante o qual se indica a sequência de ações que o robô deverá cumprir durante a realização de uma dada tarefa. Conforme a norma ISO TR 10562, o Código Intermediário para Robôs (ICR) é um pseudocódigo de baixo nível que possui os elementos básicos para permitir que qualquer linguagem de alto nível seja para ela traduzida. Para que isso seja possível, devem ser desenvolvidos compiladores adequados. Cada tradutor irá fazer uso das capacidades da ICR do modo que lhe for mais conveniente a fim de satisfazer o usuário final [Hernandes, 2002]. Porém, pelo fato de os robôs industriais possuírem configurações significativamente distintas, aliado ao fato de poderem ser empregados para diversos tipos de tarefas, cada fabricante desenvolveu a sua própria linguagem de programação que é válida somente para uso com os seus equipamentos. Hoje existem duas estratégias principais na estruturação da interface homem-máquina. Uma objetiva que a linguagem seja simples, para ser usada pelos próprios operadores sem um treinamento computacional específico. A outra objetiva que a linguagem deva prover requisitos computacionais poderosos e que somente técnicos especialmente treinados devam desenvolver a programação. Um exemplo clássico da primeira estratégia é a linguagem Arla, da ABB; outro exemplo, da segunda corrente, é a linguagem Karel da Fanuc.

Atualmente, existem centenas de linguagens de robôs disponíveis comercialmente. Muitas delas baseadas em linguagens clássicas tais como Pascal, C, Modula-2, BASIC, e Assembler. As linguagens de programação podem ser classificadas de acordo com o sistema de

referência do modelo, o tipo de estrutura de controle utilizada, o tipo de especificação de movimento, a interface com os dispositivos externos e os periféricos a serem utilizados.

2.2.1 Programação Online

A programação *Online* pode ser feita através de dois métodos. O primeiro consiste em mover o robô manualmente, realizando dessa forma a trajetória desejada e guardando informações como velocidade, aceleração e tipo de movimento. Este método funciona como um macro que fica armazenado para que a ação seja repetida, quando necessária. O segundo método consiste em mover o robô para pontos que definem uma trajetória, sendo o movimento executado por meio de um dispositivo de controle manual do robô chamado *Teach Pendant*. Neste caso, o programador indica, com base nos pontos armazenados, o comando apropriado de movimentação a ser utilizado.

Segundo Gong, 1998, na programação *Online*, o programador humano pode ser exposto a um ambiente insalubre ou perigoso e os movimentos ensinados são, na melhor das hipóteses, dependentes das habilidades do programador. Este método é usualmente adotado para operações como *pick-and-place* ou tarefas de manuseio de materiais, pois o movimento depende somente de alguns pontos chave. Porém, para tarefas como soldagem e pintura, se torna menos atrativa, principalmente pelo tempo de programação necessário para ensinar toda a trajetória desejada para o robô.

2.2.2 Programação Offline

Na programação *Offline*, novos programas podem ser preparados em um computador e transferidos para o controlador do robô sem necessariamente interromper a sua operação. A programação pode ser feita via software de programação em linguagem textual ou em um software de simulação gráfica para descrever as ações a serem feitas pelo robô [Gong, 1998]. Segundo Hernandes, 2002, as principais vantagens da programação *Offline* são:

- **Redução do tempo ocioso:** A programação da próxima tarefa do robô não interrompe a tarefa atual.
- **Ambientes perigosos:** Redução do tempo de exposição do operador ao ambiente industrial onde o robô se encontra.
- **Simplificação na programação:** É possível programar o robô sem conhecer as peculiaridades de seu controlador.

- **Integração intrínseca com sistemas CAD/CAM.**
- **Maior facilidade de depuração de programas.**

A principal desvantagem da programação *Offline* é a necessidade de um modelo teórico preciso do robô e do ambiente no qual ele será utilizado. Porém, com as grandes diferenças entre os modelos de robôs existentes no mercado, torna-se difícil desenvolver um software de programação e simulação genérico, compatível com todos robôs. Outro problema é o erro natural existente entre o modelo e o sistema físico real, o que faz com que muitas vezes o robô não alcance um determinado ponto desejado com precisão. Uma opção para solucionar esta deficiência é coletar os pontos com o *Teach Pendant* para depois utilizá-los na programação *Offline* do robô.

2.2.3 Tipos de Comandos

Mesmo com os fabricantes desenvolvendo linguagens próprias de programação que somente são utilizadas em seus robôs, existe uma padronização quando se trata dos comandos utilizados. Basicamente, um programa de robô genérico segue as seguintes estruturas e funções:

Declaração de variáveis: a definição de variáveis normalmente é a primeira tarefa a ser executada em um programa. As variáveis podem ser numéricas, booleanas ou coordenadas geométricas. Os pontos a serem utilizados pelos comandos de movimentação são também declarados.

Funções de contadores: são funções do tipo laço incremental e decremental e normalmente possuem um valor inicial, um incrementador (ou decrementador) e um valor de parada.

Manipulação de interfaces: todo robô deve ser capaz de interagir com o ambiente a sua volta, seja por sensores ou atuadores. Normalmente, funções de manipulação de entradas e saídas digitais estão presentes no controlador do robô. Estas funções são muitas vezes utilizadas na implantação do sistema de segurança do robô.

Funções de temporização: muitas vezes é preciso fazer uso de funções de temporização, principalmente quando o robô precisa ficar aguardando finalizar algum processo antes de realizar a próxima tarefa.

Controle de velocidade e precisão: comandos de movimentação normalmente possuem também variáveis referentes à velocidade e precisão do movimento e/ou de parada, para que o controlador possa calcular a melhor trajetória a ser adotada. Caso não sejam definidos, o programa adota velocidade e precisão padrões.

Controle do efetuador final: é executado geralmente por meio de comandos booleanos, (on / off), associados à execução de ações pelo efetuador do robô.

Movimentação com coordenadas absolutas: é realizada por meio de comandos que utilizam diretamente o sistema de coordenadas do robô.

Movimentação com coordenadas relativas: é realizada por meio de comandos de movimentação com coordenadas relativas, as quais utilizam um ponto de referência e um deslocamento relativo (*offset*). Deste modo, é inicialmente determinado um ponto de partida para o robô no volume de trabalho, definindo os demais pontos da trajetória a partir desta referência.

Tipos de movimentos: movimentos podem ser lineares ou rotacionais. Robôs cartesianos, por exemplo, têm maior facilidade para executar comandos lineares; já, os robôs com juntas rotacionais apresentam maior facilidade para executar comandos de arcos. Existem também comandos de movimento livre, nos quais o controlador decide qual tipo de movimentação é mais simples, em termos do movimento de juntas, de ser realizada.

2.3 Controle Discreto, Comandos Combinatórios e Sequenciais

Segundo Bollmann, 1997, quando há a necessidade da integração de sistemas físicos que fazem uso de um controle contínuo com outros sistemas de natureza física, faz-se uso de técnicas de controle com alguma para se comunicar e tomar decisões com base nos elementos que compõem o meio. Tais sistemas de controle utilizam uma lógica discreta e são baseados em ações lógicas denominadas de comandos. A lógica binária que rege o controle discreto é booleana e suas operações são: SIM, NÃO, E, OU, NÃO E, NÃO OU, EXCLUSIVO E, EXCLUSIVO OU E TEMPORIZADORES.

O comando é definido como o processo mediante o qual uma ou mais grandezas de entrada influenciam uma ou mais grandezas de saída de acordo com as características próprias deste sistema [DIN 19226]. A Figura 2.2 apresenta, esquematicamente, um sistema simples de duas entradas E1 e E2 e uma saídas S1 representando um comando.

Figura 2.2 – Representação elementar de um Comando.

No caso, a Saída 1 é alguma função que depende das entradas Entrada1 e Entrada2, poderia ser uma operação booleana como, por exemplo:

$$Saída\ 1 = (Entrada\ 1)(Entrada\ 2)$$

$$Saída\ 1 = (Entrada\ 1 + Entrada\ 2)(Entrada\ 2)$$

2.3.1 Comandos Binários

Os comandos binários podem ser classificados tanto combinatórios quanto sequenciais, como mostra a Figura 2.3. No primeiro caso, as grandezas de saída, são originadas da combinação de suas entradas através de funções lógicas.

Já, nos comandos sequenciais, as saídas do sistema dependem das funções lógicas entre as entradas e também de variáveis próprias ao sistema físico, ou seja, uma determinada combinação de entradas nem sempre gera a mesma saída.

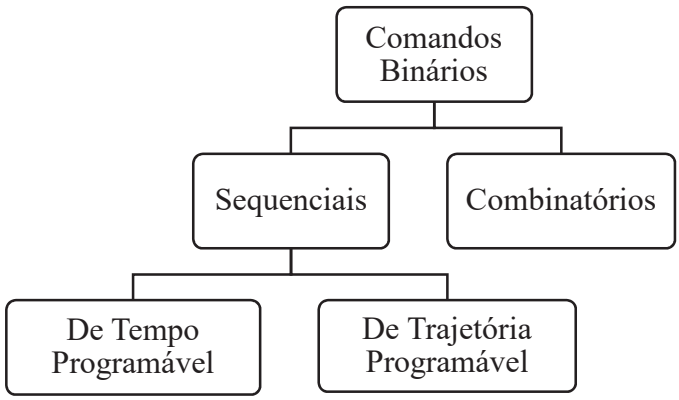

Figura 2.3 – Comandos binários combinatórios e sequenciais.

Fonte: Adaptado de Bollmann, 1997.

2.4 Linguagens de Programação para CLP – Norma IEC 61131-3

Assim como ocorre com os robôs industriais, existem no mercado um grande número de fabricantes de CLP. Como não existe uma normatização geral adotada por todos fabricantes, estes equipamentos não apresentam necessariamente uma compatibilidade de programação ou comunicação, obrigando, via de regra, as empresas a adquirirem pacotes com diversos equipamentos do mesmo fabricante por conta da incompatibilidade com outros equipamentos. Para o fabricante de equipamentos em automação isto é vantajoso, mas para os clientes acaba sendo muitas vezes mais custoso para implementar uma solução em automação em sua empresa.

Em 1992, com o intuito de tentar evitar que ilhas tecnológicas se formassem e por pressão de clientes principalmente para que pequenas empresas e fornecedores tivessem mais acesso à automação, foi publicada pelo IEC *(International Electrotechnical Commission)*, 2016, a norma IEC 61131, que estabeleceu padrões para controladores lógico programáveis. A norma se aplica a Controladores Programáveis e seus periféricos, tais como ferramentas de depuração, equipamentos de testes e Interfaces Homem-Máquina (IHM), não se aplicando, porém, a outros componentes de um sistema de automação.

Segundo a PLCopen, 2015, os objetivos da norma são: estabelecer critérios e características para seleção e aplicação de Controladores Programáveis; especificar os requisitos mínimos para funcionalidades, condições de trabalho, características construtivas, segurança geral e testes aplicáveis para os Controladores Programáveis e seus periféricos; definir regras de semântica e sintaxe para as linguagens de programação mais comuns, para que os fabricantes possam expandir e adaptar estas regras para suas próprias implementações de Controladores Programáveis e; definir a comunicação entre Controladores Programáveis e outros sistemas usando o MMS – *Manufacturing Message Specification*, conforme norma ISO/IEC 9508. Os principais benefícios da norma incluem a redução de custos com implantação devido às diferentes tecnologias existentes; o foco na solução do problema e não na construção do software; a redução de erros e inconsistências na construção de lógicas e; o uso de bibliotecas padrões construídas por diferentes programadores.

A norma é dividida em oito partes, porém, para fins deste trabalho, será abordada somente a terceira parte, que trata das linguagens de programação. Nessa terceira parte, a norma tem por objetivos fornecer metodologias de construção de lógicas de programação e definir cinco

linguagens de programação: *Ladder* (LD), Diagrama de Blocos Funcionais (FBD), Lista de Instruções (IL), Texto Estruturado (ST) e Sequencias Gráficas de Funcionamento (SFC). As linguagens Texto Estruturado e Diagrama de Blocos estão apresentadas com maiores detalhes no Anexo II, já, a linguagem que trata do SFC, é assunto de um tópico específico, já que é tema essencial do trabalho.

2.5 SFC – Sequências Gráficas de Funcionamento (SFC – *Sequential Function Chart*)

Segundo Silveira e Santos, 2002, um diagrama SFC é um modelo de representação gráfica do comportamento da parte de comando de um sistema automatizado. Ele advém da modelagem por "Redes de Petri" e é constituído por uma simbologia gráfica com arcos orientados que interligam etapas e transições, por uma interpretação de variáveis de entrada e saída da parte de comando caracterizado como receptividades de ações, e por regras de evolução que definem formalmente o comportamento dinâmico dos elementos de comando. A Figura 2.4 exemplifica um diagrama SFC.

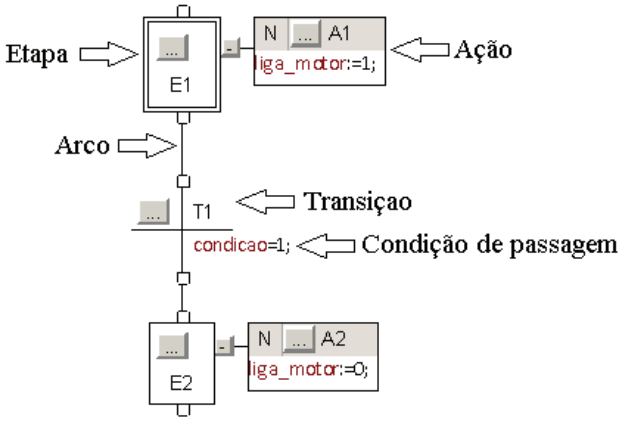

Figura 2.4 – Exemplo de Programação em SFC.

A seguir serão descritos os elementos que compõem o SFC.

2.5.1 Etapa

Uma etapa é representada graficamente por um quadrado e deve ser identificado com números, seguidos ou não por abreviaturas. Quando uma etapa ou mais estão ativas, elas representam o estado do sistema e a elas podem estar vinculadas ações de diversos tipos. A etapa inicial é representada por um quadrado duplo, como mostra a Figura 2.5.

(a) (b)

Figura 2.5 – Exemplos de uma Etapa Inicial (a) e Etapa Intermediária (b).

2.5.2 Transição

A transição (Figura 2.6) é representada graficamente por um traço perpendicular aos arcos orientados e significa a passagem de uma etapa para outra. A uma transição estão vinculadas as condições de evolução do programa. Isto garante o intertravamento entre as etapas, ou seja, em um sistema em que as ações são sequenciais, uma etapa ou ação só ocorre quando todas as etapas ou ações precedentes a ela já tiverem sido concluídas.

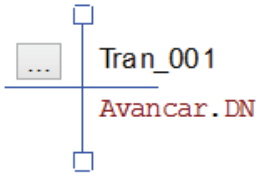

Figura 2.6 – Exemplo de uma Transição.

2.5.3 Arcos Orientados

Os arcos orientados são as linhas pelas quais são interligadas as etapas às transições e, por sua vez, as transições às etapas seguintes. O sentido convencional adotado é de cima para baixo, sendo que, quando não é assim utilizado, deve-se fazer uso de flechas para a indicação do sentido.

2.5.4 Ação

Uma ação, ou uma sequência de ações, é representada graficamente no interior de um retângulo vinculado, normalmente à direita, a uma etapa. São dispostas normalmente em forma de lista de cima para baixo, conforme a prioridade da ação. A Figura 2.7 apresenta um exemplo de ações A1, A2 e A3, vinculadas à uma etapa E3.

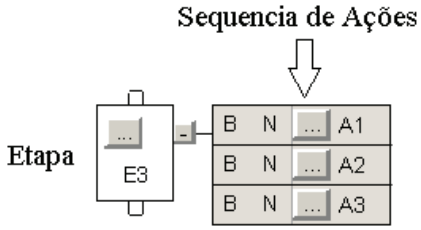

Figura 2.7 – Exemplo de ações A1, A2 e A3, vinculadas à Etapa E3.

Segundo Silveira e Santos, 2002, as ações representam efeitos que devem ser obtidos no sistema que está sendo controlado em uma determinada situação. Pode também representar uma ordem de comando que especifica o "como deve ser feito". Os comandos atuam em elementos como as saídas de um CLP, contadores, temporizadores, memórias e elementos de interface com o usuário, como vídeos, supervisórios, etc.

As ações ainda podem ser do tipo memorizada "S" (*Stored*), com retardo de tempo "D" (*Delay*), limitada no tempo "L" ou de pulso "P".

2.5.5 Intertravamentos dos SFC

Na abaixo Figura 2.8, está apresentado um módulo lógico básico da programação em SFC e sua correspondente lógica representada em blocos lógicos, onde a memória é associada a um Flip-Flop tipo RS com reset dominante. Como pode ser observado, cada passo de um programa executado em SFC é intertravado com relação à execução do passo anterior. Assim, as ações associadas a um determinado passo somente podem vir a ser efetuadas se o passo anterior tiver sido executado e se as condições de transição, geralmente estabelecidas por estados dos sensores, forem concomitantemente satisfeitas. Isso garante que as memórias associadas a cada passo sejam acionadas individualmente e que isso somente ocorra na situação para a qual foi efetivamente programada, de forma que apenas uma delas poderá estar acionada a cada instante (a não ser que seja feita uma programação específica para que isso ocorra), aumentando a confiabilidade de que a sequência irá ocorrer conforme predeterminado. No caso

de sequências que necessitem ocorrer simultaneamente, esta observação vale para cada uma das sequências que deverão ocorrer em paralelo.

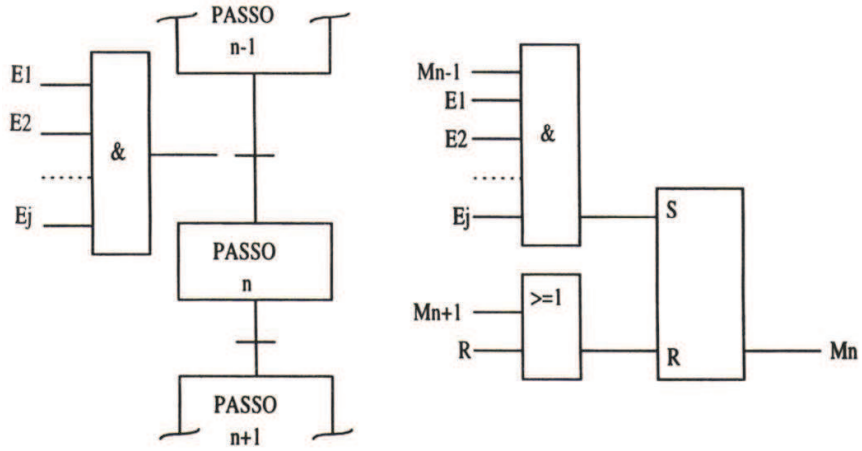

Figura 2.8- a) Módulo lógico básico b) Lógica correspondente.
Fonte: Bollman, 1997.

2.5.6 Seleção entre sequências, paralelismo e saltos

O caminho lógico pelo qual o programa em SFC flui pode ser definido de diversas maneiras:

- **Sequência única:** na qual existe somente um caminho lógico.
- **Seleção entre sequências:** neste caso, uma condição determina o caminho lógico a ser seguido entre dois ou mais caminhos (Figura 2.10 (a)).
- **Paralelismo:** dois ou mais caminhos são seguidos ao mesmo tempo (Figura 2.10 (b)).
- **Saltos:** ocorre um salto no programa quando uma determinada condição é satisfeita (Figura 2.10 (c)).

A Figura 2.9 apresenta exemplos de seleção entre sequências, paralelismo e saltos.

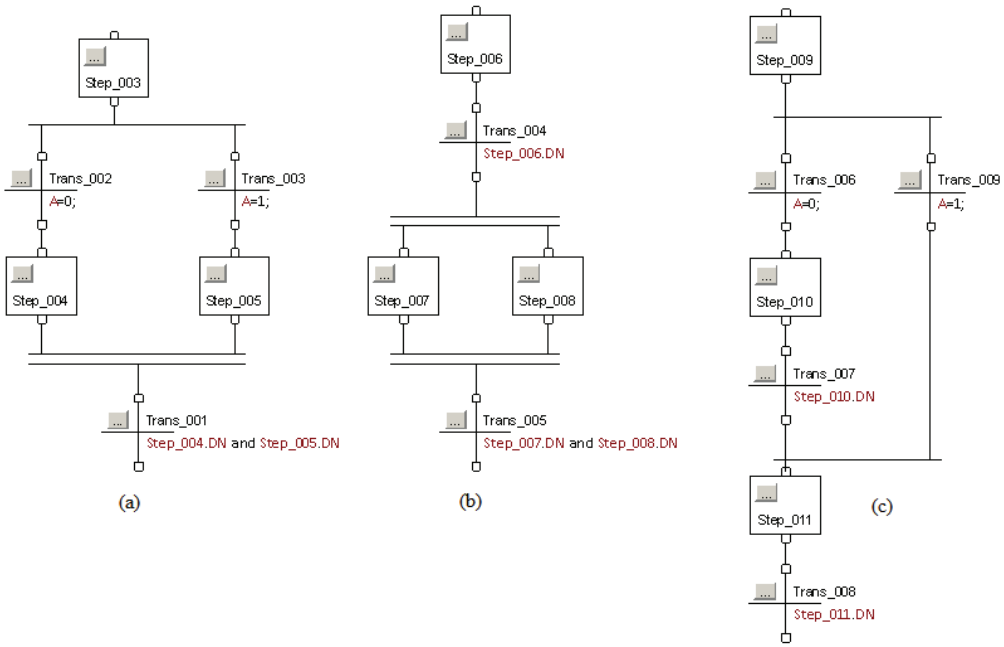

Figura 2.9 – Exemplos de seleção entre sequências (a), paralelismo (b) e saltos (c).

3 SISTEMA DE PROGRAMAÇÃO DESENVOLVIDO

Normalmente, em um ambiente fabril, existem diversos tipos de sistemas mecatrônicos que interagem entre si em sincronia e são comandados por controladores de lógica programável. Pistões, prensas, máquinas de soldagem, esteiras, sensores, entre outros elementos, operam com uma lógica binária e são controlados discretamente. Já, sistemas com monitoramento contínuo, como os robôs, precisam de um controle mais avançado e possuem uma inteligência para tomada de decisões com base nas variáveis do meio no qual estão inseridos. Portanto, pode-se inferir que os robôs precisam de um sistema de controle dedicado. Porém, a sua programação não necessariamente precisa ser exclusiva. A principal proposta do presente trabalho é a de comprovar que é possível utilizar comandos para realizar a programação de um robô juntamente com a de um sistema de controle discreto, tornando, assim, a programação de uma lógica combinatória ou sequencial unificada e simples. Uma das grandes dificuldades de atingir os objetivos desta proposta está na tradução de comandos estruturados em lógica de CLP para a lógica do controlador do robô, que varia para cada fabricante.

Neste capítulo será apresentada a proposta de uma estratégia para programar os movimentos de um robô por meio de uma linguagem usualmente utilizada para programar CLP, permitindo que, por exemplo, uma célula de trabalho automatizada possa ser integralmente programada em um mesmo ambiente e com a mesma linguagem (desde que, além dos robôs, os demais equipamentos sejam controlados por um CLP que atenda à IEC 61131-3). Para tanto, estudou-se mais aprofundadamente as linguagens apresentadas no Anexo II, visando à extração de informações dos programas nestas linguagens para que pudessem ser utilizadas nas linguagens para robôs. A este trabalho segue uma análise da sintaxe utilizada nessas linguagens para permitir a tradução dos comandos para a linguagem do robô a ser utilizado. Finalmente, foram programadas sub-rotinas em SFC específicas para movimentação de um robô e realizado o desenvolvimento de um programa para tradução destas sub-rotinas para o ambiente do sistema de controle do robô chamado de Programa Tradutor. Para melhor entendimento, toda programação do robô realizada no ambiente do SFC será chamada de SFC-R (SFC para robôs). A descrição da aplicação dessa metodologia ao RCP5 serão apresentadas no Capítulo 4.

Na Figura 3.1 está apresentado um esquema da solução proposta no presente trabalho. A estrutura do sistema apresentado foi desenvolvida com base nos recursos disponíveis no LAMECC/UFRGS (software + hardware). Ressalta-se que esta solução não necessariamente é

a única e que, com outros recursos, outras soluções viáveis podem, certamente, vir a também resolver o problema da integração do robô com célula automatizada em um mesmo ambiente por meio de linguagens de programação baseadas em SFC.

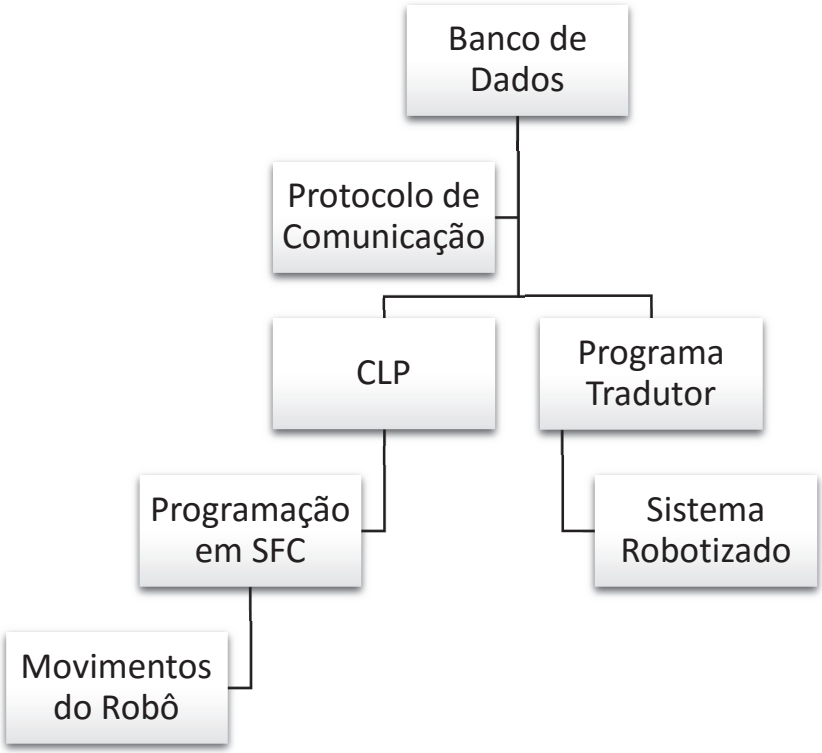

Figura 3.1 – Estrutura do sistema de programação proposto.

Na estrutura proposta, verifica-se a necessidade de um Banco de Dados comum relacionado a um Protocolo de Comunicação, um CLP (dispositivo físico ou lógico) e um Programa Tradutor. O Banco de Dados serve para armazenamento das variáveis visando à integração dos sistemas e possibilitando a troca de informações entre os mesmos, enquanto que o Programa Tradutor realiza a conversão da linguagem de CLP em linguagem do sistema de controle do robô. Finalmente, o CLP é responsável por realizar o controle do sistema físico, integrando as movimentações do robô, sistema de visão e segurança, com outros equipamentos presentes em uma célula automatizada tais como pistões, sensores, etc.

Para a programação do CLP, utilizou-se o software RSLogix 5000 da *Rockwell Automation*, onde é possível combinar rotinas programadas nas 5 linguagens baseadas na norma IEC 61131-3. Como protocolo de comunicação fez-se uso de um servidor OPC- OLE para

Controle de Processos gerenciado pelo software RSLinx, da mesma empresa, que é responsável pelo gerenciamento das conexões PC-CLP e possui suporte para criação e gerenciamento de servidores com tecnologia OPC. As rotinas do Programa Tradutor foram programadas em linguagem baseada em *scripts* do Matlab (linguagem "m"), pois esta linguagem é a mesma utilizada no sistema de controle do robô, fez-se uso, também, de bibliotecas para conexão com servidor OPC e com o sistema de controle do robô deste estudo de caso.

A seguir, os diversos componentes do sistema de programação proposto são apresentados e discutidos.

3.1 OPC–OLE para Controle de Processos

Como foi mencionado anteriormente, a primeira etapa do trabalho consiste na criação de um banco de dados que possa ser acessado tanto pelo CLP quanto pelo Programa Tradutor. Após um estudo preliminar sobre os padrões disponíveis no mercado a solução adotada foi o padrão OPC (*OLE for Process Control*), gerenciada pela OPC Foundation, 2015, segundo a qual, o padrão OPC consiste de uma série de especificações desenvolvidas por fornecedores da indústria, usuários finais e desenvolvedores de software, que definem a interface entre clientes e servidores, bem como entre servidores, incluindo o acesso a dados em tempo real, monitoramento de alarmes e eventos, acesso a dados históricos e outras aplicações.

Quando o padrão foi lançado, o seu objetivo era para a aplicação em protocolos específicos de CLP (como Modbus, Profibus, etc.) em uma interface normalizada, permitindo que sistemas Homem-Máquina fizessem pedidos de leitura e gravação em solicitações específicas do dispositivo (e vice-versa). Como resultado, uma modalidade inteira de novos produtos surgiram, permitindo que os usuários finais pudessem implementar sistemas aplicados, utilizando o OPC em vários setores, incluindo manufatura automação predial, petróleo e gás, energia renovável e serviços públicos, entre outros.

Hoje em dia, fabricantes da área de automação desenvolvem sistemas fechados e com linguagem própria, tornando trabalhosa e dispendiosa a obtenção de compatibilidade com dispositivos de outros fabricantes. O OPC serve como uma ponte entre estes dispositivos para que se possa ter uma maior poder de escolha de fabricante, quando da implantação de um sistema. No presente caso, o OPC serve de "ponte" de ligação entre o CLP e o Programa Tradutor.

Um sistema OPC funciona com conexões cliente-servidor, como mostra o exemplo da Figura 3.2.

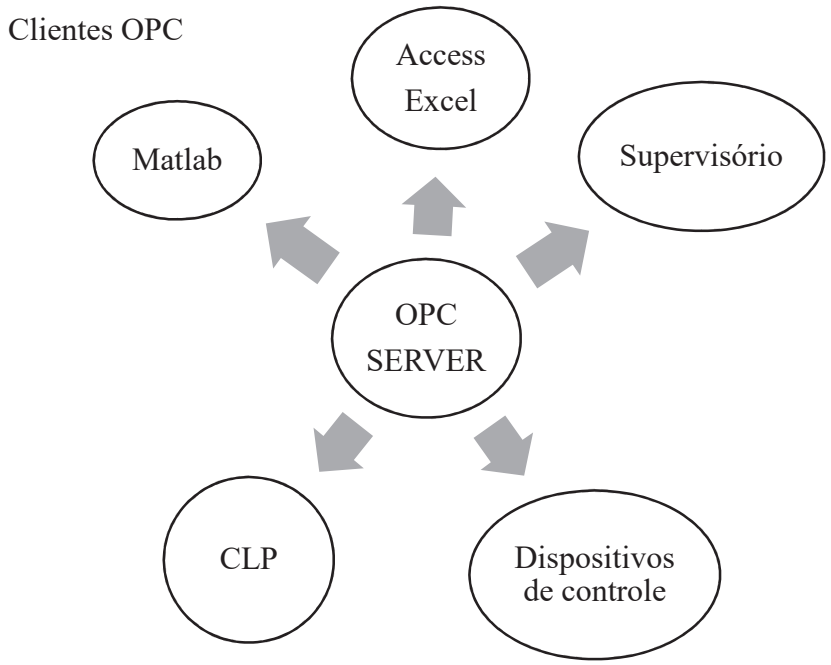

Figura 3.2 – Exemplo de arquitetura OPC.

A Figura 3.2 ilustra, através de um exemplo, a versatilidade da arquitetura OPC. O protocolo Ethernet permite criar uma rede entre vários dispositivos que podem ser acessados por programas de supervisão, geradores de relatório, softwares de controle, tais como Matlab e LabVIEW. Para estruturação do banco de dados, é necessário criar grupos nos quais são inseridos itens referenciados a alguma variável de controle do CLP. A interconexão entre os sistemas via OPC pode ser representada pelo esquema apresentado na Figura 3.3.

Figura 3.3 – Interconexão entre os subsistemas.

Um programa em linguagem SFC é carregado para o CLP que se comunica com o Sistema robótico através de dois subsistemas: um servidor OPC que faz a comunicação entre o CLP e o Programa Tradutor, este por sua vez, traduz a informação para a linguagem do sistema de controle do robô.

3.2 Estratégia de programação

A programação para CLP é geralmente realizada por meio de acesso a variáveis booleanas ou inteiras. As variáveis alfanuméricas (*string* e *char*), que são amplamente utilizadas nas linguagens de programação mais comuns (C, Basic, Pascal, etc.), raramente fazem parte de uma programação em *Ladder* ou SFC. Da mesma forma, as ações programadas em SFC estão usualmente associadas a variáveis booleanas. Porém, atualmente, a maioria dos softwares de programação para CLPs permitem a integração das cinco linguagens da norma IEC. Portanto, é possível, por exemplo, escrever uma rotina em texto estruturado que pode ser chamada de um programa SFC (ou chamar sub-rotinas de uma das outra linguagens normatizadas).

Como a maioria das variáveis que compõem um comando de movimentação de um robô não são booleanas, e, além disso, para controlar a dinâmica do mesmo, é necessário realizar operações de ponto flutuante, optou-se pela utilização de sub-rotinas escritas em texto estruturado chamadas de um programa estruturado em SFC-R, como pode ser visualizado no diagrama esquemático apresentado na

Figura 3.4.

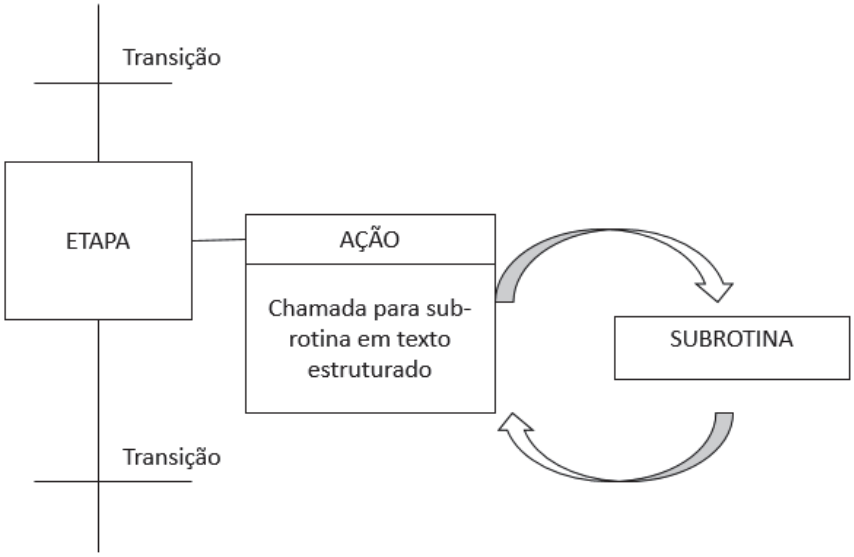

Figura 3.4 – Exemplo da estrutura utilizada para chamada de uma sub-rotina escrita em texto estruturado.

Assim, torna-se possível combinar um comando complexo (como o de movimentação do robô) com comandos mais simples, como atuação de um pistão ou leitura de um sensor. Nos sistemas de programação comerciais, estas sub-rotinas podem geralmente ser importadas e exportadas pelo usuário. Utilizando esta estratégia, é possível estruturar bibliotecas de sub-rotinas configuradas para cálculos de parâmetros para diferentes sistemas robóticos a serem selecionadas para uso de acordo com cada aplicação.

3.2.1 Padronização das variáveis

Diferentemente das linguagens tradicionais de programação (como C e o C++), onde as variáveis são declaradas via texto e podem ser utilizadas como globais ou locais para uma sub-rotina, em programações de CLP geralmente variável deve ser declarada em uma tabela ou banco de dados. Estas variáveis são comumente chamadas de “tags”, e podem ser declaradas em dois contextos: o do Controlador, onde normalmente ficam localizadas as variáveis relacionadas às entradas e saídas físicas do sistema, dependendo do servidor OPC a ser utilizado; o segundo é o de programa, no qual podem existir rotinas em diversas linguagens, todas fazendo parte do contexto de programa.

No caso do presente estudo, há a necessidade de um programa de tradução que possa conversar com o programa SFC-R, ou seja, as variáveis e parâmetros devem ser padronizados para que ambos programas tenham acesso, já que o usuário final não tem acesso à programação do Programa Tradutor. Para a definição de um padrão de variáveis de controle e programação, é necessário dispor-se dos elementos que compõem o movimento de um sistema robotizado, como o apresentado no Capítulo 2. Variáveis, como coordenadas de translação e orientação, são inerentes à programação do movimento de todo robô. No âmbito da estratégia adotada, são necessários também parâmetros que auxiliem a tradução para outras linguagens, como a indicação de um novo comando (ou movimento), o tipo de comando e um indicativo de que o comando (ou movimento) foi executado.

Com base nestes conceitos, a partir de um levantamento das variáveis de movimentação e de controle que são necessárias, foi elaborada uma tabela descritiva com o tipo de variável, nome e estados possíveis. No presente trabalho, os movimentos padronizados são o de translação de um ponto para outro (movimento ponto a ponto); de *pick-and-place* e de

paletização. Na Tabela 3.1, estão apresentas as variáveis relativas ao controle do robô pneumático.

Tabela 3.1 – Tags padronizadas.

Tag:	Tipo	Correspondência Lógica
robot_type	Inteiro	01: Robô do estudo de caso 02: Reservado para uso futuro 03: Reservado para uso futuro 04: Reservado para uso futuro 05: Reservado para uso futuro
robot_ok	Booleana	1: Robô apto para operação.
		0: Robô não apto para operação.
new_cmd	Booleana	1: Existe um novo comando para o robô
		0: Não existe um novo comando
cmd_type	Inteiro	01: Movimento Linear
		02: Movimento Pick and Place
		03: Movimento de Paletização
		10: Movimento de parada
cmd_executed	Booleana	1: Comando foi executado pelo robô
		0: Comando não foi executado pelo robô
cmd_accepted	Booleana	1: Comando válido aceito pelo Programa Tradutor
		0: Comando válido ainda não lido pelo Programa Tradutor ou comando inválido

Suas variáveis são descritas abaixo:

- **robot_type:** indica qual o tipo de robô que está sendo utilizando. Atribuiu-se um código (valor 01) para o robô do estudo de caso (reservou-se mais 4 códigos para uso futuros).
- **robot_ok**: indica que a inicialização do robô está concluída e, que o robô foi para um ponto inicial padrão, a comunicação está funcionando corretamente e o robô pode começar a operar novamente.
- **new_cmd**: indica uma ação (comando) do programa do CLP relacionada ao robô.
- **cmd_type**: esta variável serve como uma chave (switch) para que o programa possa, para cada tipo de movimento, tomar as decisões cabíveis.
- **cmd_executed**: é uma variável usada pelo Programa Tradutor na troca de informações entre o CLP e o robô, que indica que o robô completou seu movimento. Quem modifica tal variável é o próprio programa de conversão e controle. Esta variável é normalmente usada em movimentos lineares para um ponto.

- **cmd_accepted**: é uma variável usada na sincronia do Programa Tradutor e o programa estruturado em SFC-R, que indica de que o Programa Tradutor conseguiu ler com sucesso as variáveis necessárias para executar o comando.

Além disso, na estratégia adotada, criou-se uma classe denominada "*Target*" para facilitar o armazenamento das coordenadas de um determinado ponto no volume de trabalho do robô. Conforme mostra a Tabela 3.2, os membros da variável "Target" são as coordenadas de translação (x, y e z) e as coordenadas de rotação (*roll, pitch* e *yaw)*. Na implantação adotada, cada elemento pode ser editado através da chamada da classe seguido do caractere "." e o membro da classe que se deseja editar.

Tabela 3.2 - Membros da classe "*Target*".

Classe Target	
Membro	Tipo
x	DINT
y	DINT
z	DINT
roll	DINT
pitch	DINT
yaw	DINT

3.2.2 Programação em SFC-R

Para que haja sincronia na troca de informações, a programação em SFC-R deve levar em consideração o sequenciamento de ações e as verificações dos estados para que uma determinada sub-rotina seja chamada. Dessa forma, para que uma sub-rotina associada a um determinado passo seja chamada, as tarefas associadas ao passo anterior devem ter sido adequadamente finalizadas.

A Figura 3.5 apresenta um fluxograma descrevendo as etapas e ações necessárias para a chamada de uma sub-rotina de movimentação do robô utilizando a estratégia de programação adotada no presente trabalho.

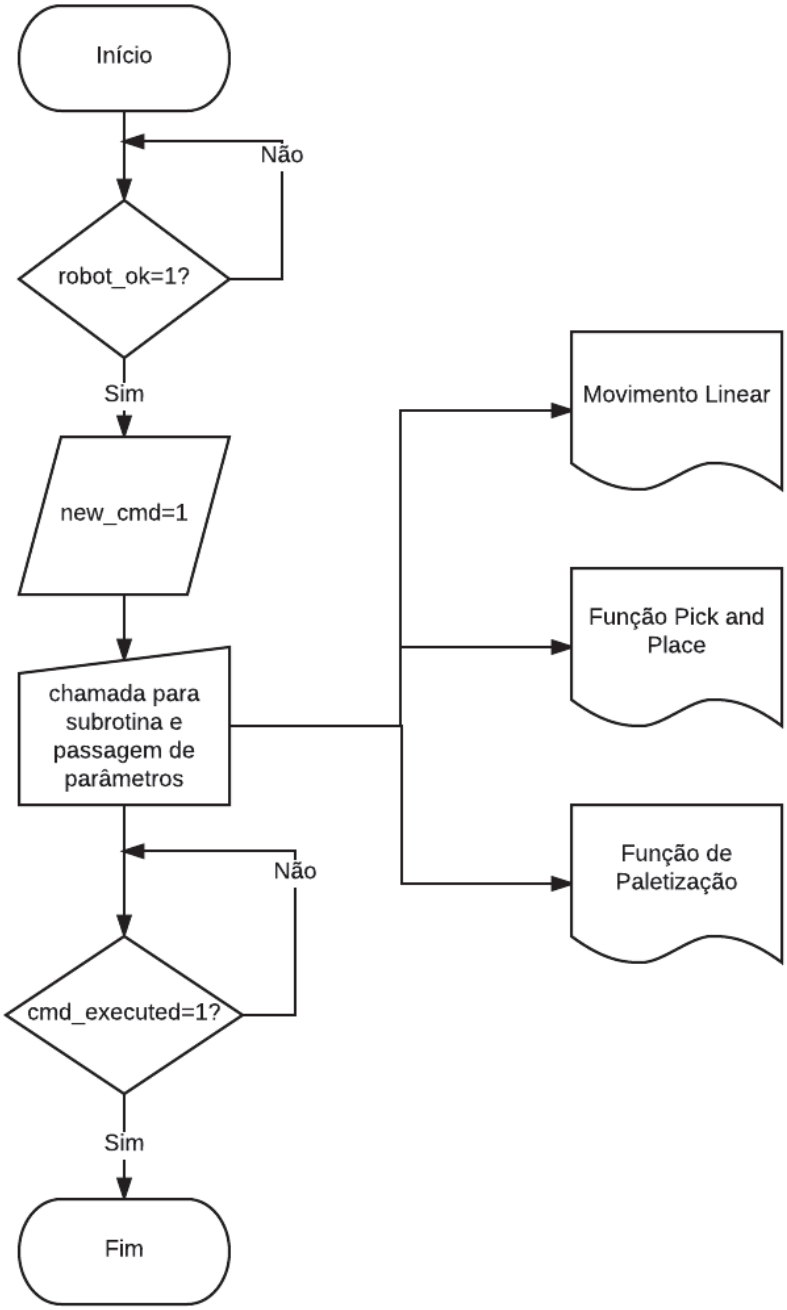

Figura 3.5 – Fluxograma referente a chamada de uma sub-rotina em SFC, caso geral.

Antes de realizar a chamada para uma sub-rotina de movimentação do robô, o programa em SFC-R deve, primeiramente, verificar se o robô está apto para operar e se as tarefas anteriores foram finalizadas. Após, deve indicar para o Programa Tradutor um novo comando de movimentação do robô e, em seguida, a chamada da sub-rotina e passagem dos parâmetros necessários para o mesmo. Por fim, deve verificar o êxito na execução do movimento para continuar a processar o programa.

3.3 Sub-rotinas SFC-R de comandos de movimentos.

As sub-rotinas de comandos de movimentos foram programadas em texto estruturado, principalmente devido à quantidade de operações matemáticas em ponto flutuante necessárias. Tais sub-rotinas podem ser acessadas (chamadas) pelos comandos programados em SFC-R. As sub-rotinas fornecem os dados necessários para que o Programa Tradutor gere as informações necessárias para comandar o robô, de acordo com cada movimento desejado, como, por exemplo, calcular a cinemática inversa e posições necessárias paras ações de paletização.

Para este trabalho, desenvolveu-se sub-rotinas para 3 tipos de movimentos comuns aos sistemas robotizados: movimento ponto a ponto; movimento completo de *pick and place* e movimento de paletização. Para cada tipo de movimento foi programada uma sub-rotina, sendo, evidentemente, os parâmetros enviados para o Programa Tradutor dependentes de cada caso.

A maioria do softwares de programação possui uma chamada para uma rotina auxiliar. Como mencionado no início do presente capítulo, para programação das sub-rotinas, foi utilizado o software RSLogix 5000 da Rockwell Automation, cujas chamadas para sub-rotinas são feitas através do comando "JSR", em inglês: *Jump to Subroutine* da seguinte forma:

JSR(nome da subrotina, número de argumentos, lista de argumentos)

A seguir são apresentados alguns exemplos de aplicação deste comando.

3.3.1 Movimento linear (cmd_type=01)

O movimento ponto a ponto é um dos mais utilizados em programação de robôs. Os parâmetros necessários para tal movimentação normalmente incluem as coordenadas de translação (x,y e z) e de rotação (*roll*, *pitch*, *yaw*) do ponto desejado, velocidade e estado do efetuador.

Portanto, a chamada para a sub-rotina de movimento linear, utilizando o método descrito na Seção 3.3 segue a seguinte sintaxe:

JSR(mov_linear, **3, Ponto desejado, tempo de execução, estado do efetuador)**

onde o comando "JSR" indica uma chamada para uma sub-rotina seguida dos parâmetros: mov_linear que é o nome da sub-rotina desejada; o número 3 que indica a quantidade de

parâmetros que serão passados; o ponto desejado, do tipo “Target” (descrita na Seção3.2.1); o tempo previsto para a execução do movimento (este é um parâmetro de entrada da função de geração de trajetória do sistema de controle do robô, por este motivo não é utilizada a velocidade como parâmetro de entrada); e o estado do efetuador.

Na Figura 3.6, é possível visualizar um exemplo de programação feita em SFC-R, chamando um movimento linear para um ponto.

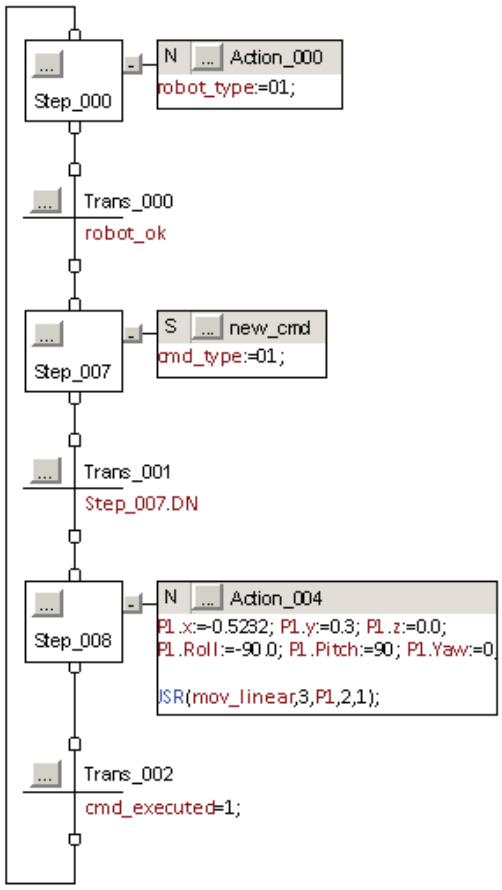

Figura 3.6 – Exemplo de uma chamada para sub-rotina de movimentação linear.

A figura apresenta um programa simples para a chamada da sub-rotina mov_linear, a etapa “Step_000” indica o tipo de robô que está sendo utilizado por meio de um valor numérico correspondente ao robô, o qual deverá estar associado a um banco de dados com as características cinemáticas (incluindo as dimensões) de cada robô a ser programado. Ligando a etapa “Step_000” à etapa seguinte, está uma transição “Trans_000” onde a condição de passagem de uma etapa para outra é a variável “robot_ok”, indicando se o robô está apto para operar. Na etapa “Step_007”, a primeira ação corresponde à passagem da variável new_cmd

para nível alto, indicando que está sendo iniciado um novo conjunto de comandos para programação do robô. A segunda ação indica qual tipo de movimento será utilizado. Conforme mostra a Tabela 3.1, estabeleceu-se para variável “cmd_type”, associada à movimentação ponto a ponto, o valor de 01. Na terceira ação, são declaradas as coordenadas de rotação e translação do ponto desejado e a subrotina mov_linear é chamada através do comando JSR com os seguintes parâmetros: mov_linear (o nome da subrotina), o número 3 indicando que serão 3 parâmetros de entrada e, por último, os parâmetros propriamente ditos (em ordem: ponto P1, tempo de execução e estado do efetuador).

3.3.2 Movimento de *pick and place* (cmd_type =02)

Conforme a Tabela 3.1, a sub-rotina de *pick and place* possui o código do tipo de movimento (variável “cmd_type”) igual a 02. Neste caso, os parâmetros de entrada do sub-programa são: ponto inicial e final, altura, e tempo de execução. Para a passagem dos pontos inicial e final decidiu-se por criar a classe apresentada na Seção 3.2.1, chamada de “*Target*”, enquanto que a altura da movimentação possui um valor *default* de 10 cm no eixo z, mas pode ser personalizada. A chamada para a sub-rotina de movimentação de *pick and place* é feita através da seguinte sintaxe:

JSR(mov_pnp, **4, ponto inicial, ponto final, altura, tempo de execução)**

onde o comando “JSR” indica uma chamada para uma sub-rotina seguida dos parâmetros: mov_pnp que é o nome da sub-rotina desejada; o número 4 indica a quantidade de parâmetros que serão passados; o ponto inicial e o ponto final desejados, do tipo “Target”; a altura da movimentação em metros; e o tempo previsto para a execução do movimento, em segundos.

Como é apresentado na Figura 3.7, a etapa “Begin1” é a etapa inicial do programa seguida da transição que depende da variável “robot_ok”. Quando a transição assume o valor verdadeiro, ou seja, o robô está apto para operar, duas ações associadas à etapa Step_01 são realizadas: indica-se para o Programa Tradutor um novo comando (modifica a variável “new_cmd” para valor verdadeiro) e qual o tipo de comando (variável “cmd_type=02”). Na etapa seguinte (Step_02) são declarados os parâmetros de entrada (pontos P1 e P2) e a chamada da sub-rotina “mov_pnp”. Por fim, uma transição verifica se o comando foi executado (variável

"cmd_executed") para executar a etapa Step_04, onde está associada a ação de modificar o estado da variável "new_cmd" para falso.

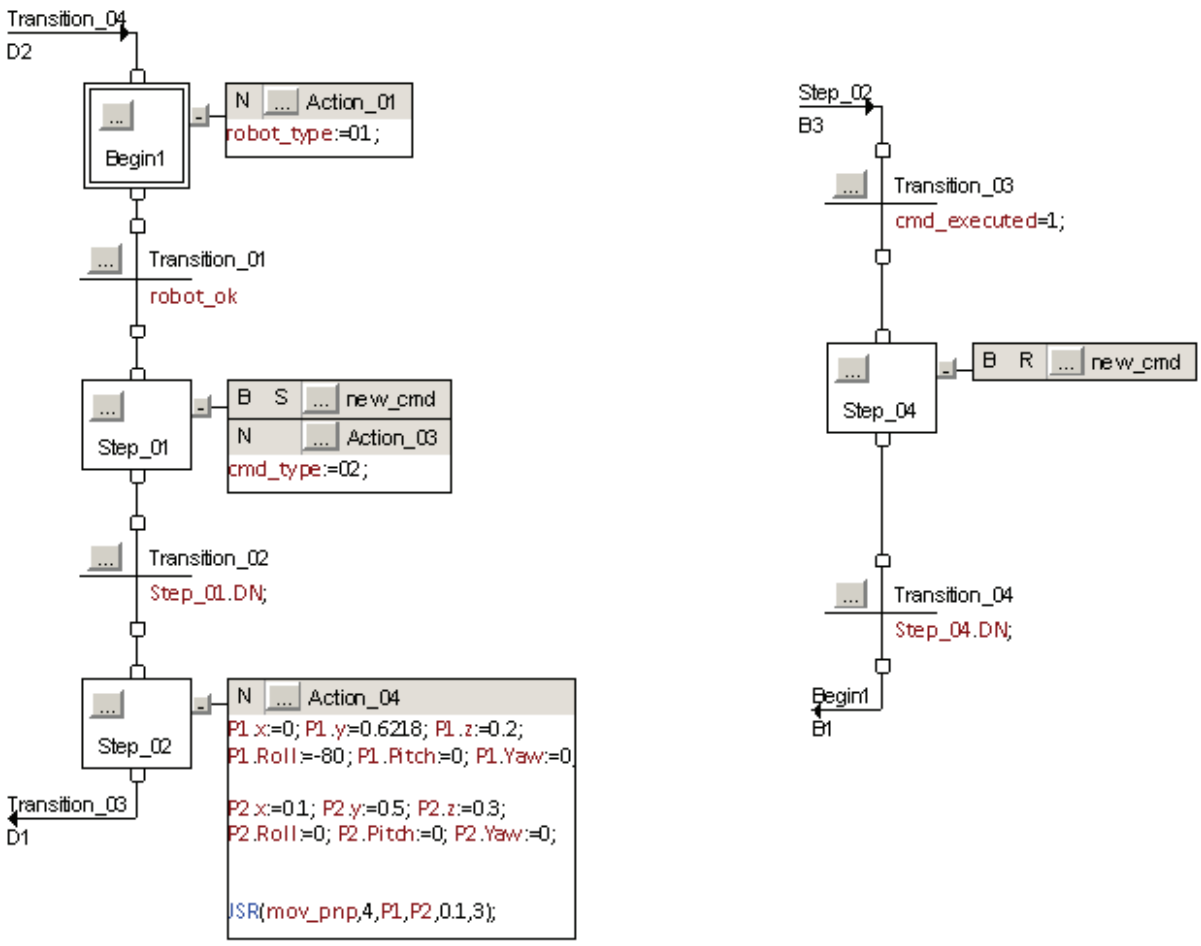

Figura 3.7 – Exemplo de chamada para uma sub-rotina de movimentação *pick and place*.

3.3.3 Movimento de paletização (cmd_type=03)

A paletização é uma movimentação comum em ambiente fabril. Ela envolve uma quantidade grande de parâmetros, tais como dimensões do pallet, dimensões do produto, entre outros. Conforme a Tabela 3.1, o código associado à sub-rotina do movimento de paletização é o "03". Os parâmetros de entrada são: o ponto desejado para captura do objeto e o ponto relativo à origem do pallet; as dimensões do produto (x, y e z) em metros, incluindo uma "folga" de segurança e considerando sempre o produto como paralelepípedo; o número de colunas, linhas e andares do pallet. Como nos casos anteriores, a partir desses parâmetros, uma sub-rotina programada em texto estruturado se encarrega do cálculo dos pontos que definem a trajetória

das juntas do robô. Assim como na movimentação de *pick and place*, a paletização tem variáveis de controle de início e parada de movimento, chamadas de pallet_start e pallet_stop. A chamada para a sub-rotina (Figura 3.8) segue a seguinte sintaxe:

JRS(mov_pallet**, 8, ponto de captura, ponto relativo à origem do sistema de coordenadas do pallet, dimensão do produto_x, dimensão do produto_y, dimensão do produto_z, colunas, linhas, andares)**

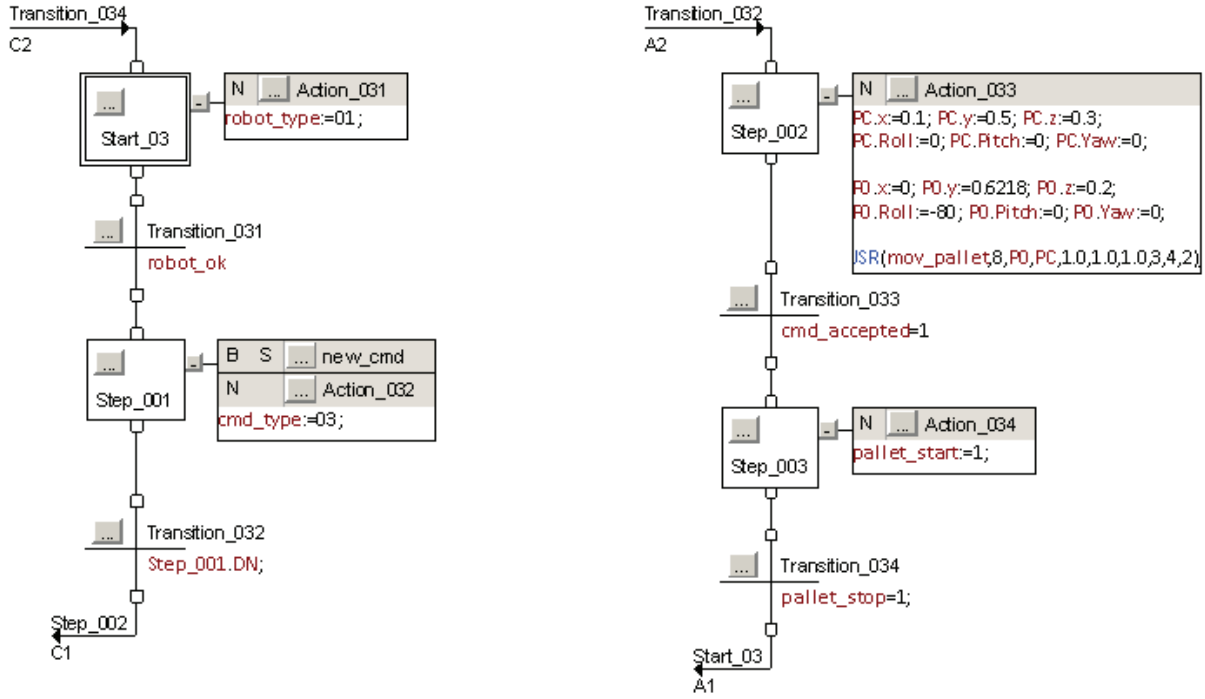

Figura 3.8 – Exemplo de chamada para sub-rotina de paletização.

Assim como nos exemplos anteriores, o programa começa com uma etapa inicial (Start_03) e uma transição que depende do robô estar apto a operar (variável “robot_ok”). Em seguida, duas ações estão associadas à etapa Step_001: a indicação de um novo comando (variável “new_cmd”) e seu tipo (variável “cmd_type=03”). No próximo passo (Step_002) são declarados os pontos de captura e origem do sistema de coordenadas do pallet (P0 e PC, no caso) e é chamada a sub-rotina “mov_pallet”. Em seguida, verifica-se se o comando foi aceito pelo Programa Tradutor através da transição Transition_033 e ao próximo passo (Step_003) está associada uma ação para início do movimento. Por fim, o programa verifica se o comando

foi executado pelo estado da variável “pallet_stop”, que muda para o valor “1” quando o Programa Tradutor identifica o final do movimento de paletização.

3.4 Programa Tradutor

O Programa Tradutor tem como objetivo converter a linguagem de programação, baseada na norma IEC 61131-3, para a linguagem utilizada pelo robô. Pela praticidade, e pelo fato de que o estudo de caso foi realizado em um robô onde a sua estrutura de controle e geração de trajetória é feita via Matlab®, decidiu-se por utilizar tal linguagem de programação também para o Programa Tradutor. Além disso, o Matlab® possui uma biblioteca (*Toolbox*) específica para gerenciamento e conexão OPC. As rotinas são inicialmente programadas em linguagem baseda em *scripts* de Matlab (linguagem “m”) e posteriormente compiladas para um programa executável em ambiente Windows.

Funções e rotinas programadas para operação com o programa SFC-R são apresentadas a seguir.

3.4.1 Rotina de definição do robô

Essa rotina permite a entrada de dados do robô a ser utilizado. O usuário deve informar o número codificado para o robô e a rotina deve ser capaz de carregar os dados de tal robô previamente salvos em um banco de dados que deve conter todos os parâmetros necessários para programação e controle do robô, tais como, parâmetros Denavit Hartenberg, pressões máximas e mínimas utilizadas, espaço de trabalho, pressão de trabalho, etc. O programa também possui uma rotina para definir um novo robô através de um código e um arquivo de dados no formato “m”.

A Figura 3.9 apresenta o esquema de inter-relação das ações necessárias para a rotina.

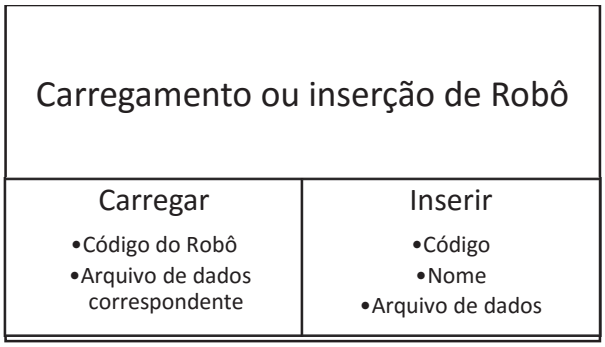

Figura 3.9 – Esquema representativo da rotina de escolha do robô.

Como apresentado na Figura 3.9, a rotina dispõe ao usuário duas opções: inserir um novo robô, através de um código, nome e arquivo de dados ou carregar um robô de uma lista através do seu código e arquivo de dados correspondente.

3.4.2 Rotina de Conexão com servidor OPC

A rotina de conexão com o servidor OPC utiliza funções presentes na toolbox OPC do Matlab®. Para conectar-se, o usuário deve fornecer os dados de IP e Nome do Servidor. Após conectado, a rotina cria um grupo de leitura e adiciona todas as variáveis ou itens presentes do servidor OPC, os quais deseja-se utilizar no Programa Tradutor. O esquema apresentado na Figura 3.10 mostra as funções que a rotina deve executar.

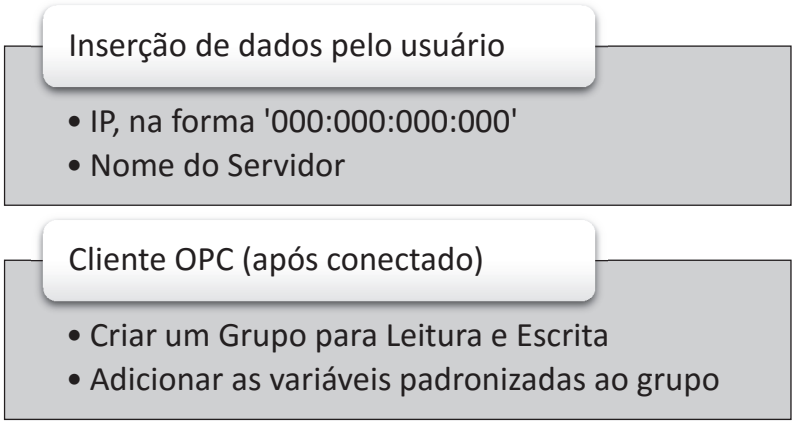

Figura 3.10 – Funções da sub-rotina de conexão com o OPC.

Como apresentado na Figura 3.10, a rotina de conexão OPC possui uma etapa de inserção de dados pelo usuário, onde o mesmo fornece o IP e nome do servidor ao qual deseja-se conectar e uma etapa após a conexão, onde a rotina cria um grupo para leitura e escrita das variáveis e adiciona as variáveis padronizadas da Tabela 3.1 ao mesmo.

3.4.3 Rotinas de Movimento

Caso o robô a ser programado seja de arquitetura de controle fechada, sem acesso direto aos códigos de controle da dinâmica do robô, segundo a estratégia proposta, a programação de um movimento pode ser realizada *off-line* através de linhas de código em texto. Neste caso, após a leitura dos dados fornecidos pelo programa em SFC sobre a movimentação desejada, o Programa Tradutor, desde que munido das rotinas referentes à linguagem do robô, pode gerar

automaticamente o código correspondente ao movimento linear respectivo. Um exemplo de aplicação para um robô de arquitetura fechada é o robô articulado de 6 GLD da empresa ABB, que utiliza o RAPID como linguagem de programação. Por exemplo, no caso de um movimento linear ponto a ponto, os pontos são descritos no cabeçalho e o código para a movimentação linear baseado no comando MOVEL. A rotina, para o robô da ABB, deve ser capaz de escrever em um arquivo de texto as coordenadas do ponto desejado e o código referente ao movimento (programado em SFC-R).

Já, para as arquiteturas abertas, como é o caso do robô do estudo de caso, a rotina deverá, inicialmente, calcular a trajetória de movimentação. No presente estudo, utiliza-se os resultados do trabalho de Missiaggia, 2014, e Sarmanho, 2014, (descritos na Seção 4.1). Assim, com base nos dados de coordenadas, é possível enviar o comando para o robô executar a movimentação e verificar se o comando foi executado.

A Figura 3.11 descreve o esquema associado às funções executadas pela rotina de movimento ponto a ponto.

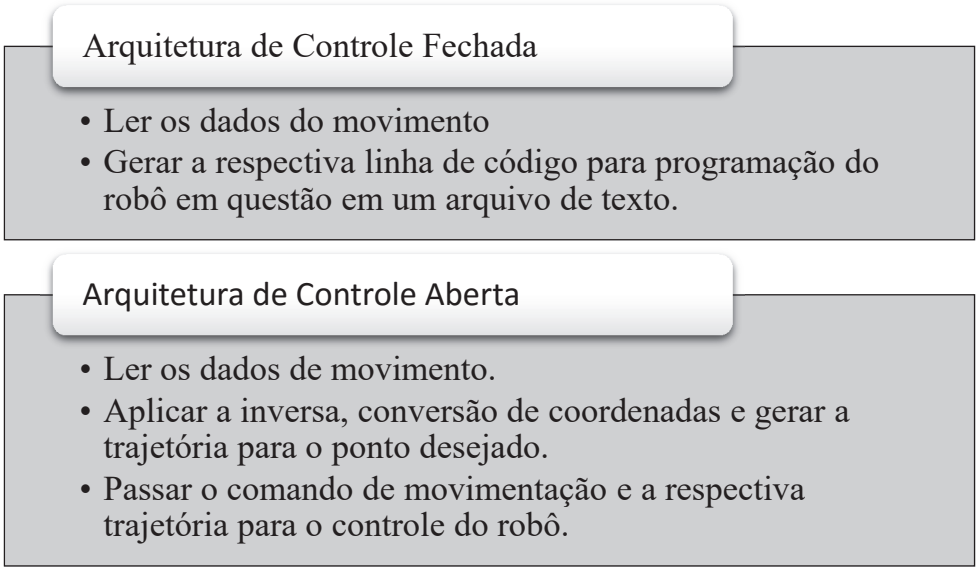

Figura 3.11 – Esquema de funções executadas pela rotina de movimentação ponto a ponto.

Como na Figura 3.11, quando o robô possui uma arquitetura de controle fechada, a rotina respectiva ao movimento linear ponto a ponto deverá ler os dados de movimentação (programados em SFC-R) e escrever o código referente a tal movimentação em um arquivo de texto. Por outro lado, quando este robô possui uma arquitetura de controle aberta, além da leitura dos dados de movimentação, a rotina respectiva ao movimento linear ponto a ponto deve

também realizar cálculos de trajetória, com base na dinâmica do robô, e enviar essa trajetória, juntamente com o comando de movimentação para o sistema de controle do robô.

Conforme já comentado, além da rotina de movimentação linear ponto a ponto, foram implantadas as rotinas de movimento de *pick and place* e de paletização. Essas rotinas possuem a mesma lógica da de movimentação linear no que diz respeito a sistemas abertos e fechados. Em um sistema fechado, a rotina lê os pontos de partida e chegada, bem como o tempo de parada entre os pontos desejados e a altura do movimento. Com base nesses parâmetros, o código a ser escrito pode ser uma sucessão de movimentos do ponto de partida para o ponto de chegada com uma trajetória típica de *pick and place* (combinação de movimentos verticais, horizontais e arcos) com relação a alguma variável externa, como, por exemplo, a de indicação de um procedimento em que, cada vez que um sensor indicar um novo objeto, a movimentação deve ser repetida.

No caso de uma arquitetura aberta, a rotina tem as seguintes funções: ler os parâmetros de movimentação gerados via SFC-R através do servidor OPC; gerar a trajetória para o primeiro ponto (ou ponto de pegar a peça); gerar a trajetória correspondente e executar a movimentação do ponto de captura de peça (*pick*), para o ponto onde a mesma será posicionada (*place*). Maiores detalhes sobre as especificações das trajetórias são apresentadas na Seção 4.1.2.

No caso da rotina de paletização, são calculados os pontos de paletização com base nas informações fornecidas pelo usuário na sua programação SFC-R. O código programado combina comandos de ações para movimentações verticais e horizontais para que a peça possa ser posicionada de acordo com as trajetórias descritas na Seção 4.1.2.

3.4.4 Rotina Principal

A rotina principal (*Main*) deve gerenciar todas as outras rotinas e chamá-las no momento adequado conforme o tipo de comando. Ela segue o sequenciamento descrito na Figura 3.12.

O primeiro passo é a seleção do tipo de robô e subsequente carregamento dos dados do mesmo presentes no banco de dados. Em seguida, a rotina de comando faz a chamada das sub-rotinas de conexão, tanto com o servidor quanto com o sistema de controle do robô, movimentando posteriormente o robô para um ponto de espera no espaço de trabalho. Após, a rotina de comando executa um laço no qual espera um novo comando do programa SFC-R, e, assim que o recebe, verifica seu tipo e chama a sub-rotina para a movimentação desejada.

Quando o robô executa o movimento, indica para o programa SFC-R através da variável “cmd_executed” e a rotina recomeça o laço de espera de um novo comando.

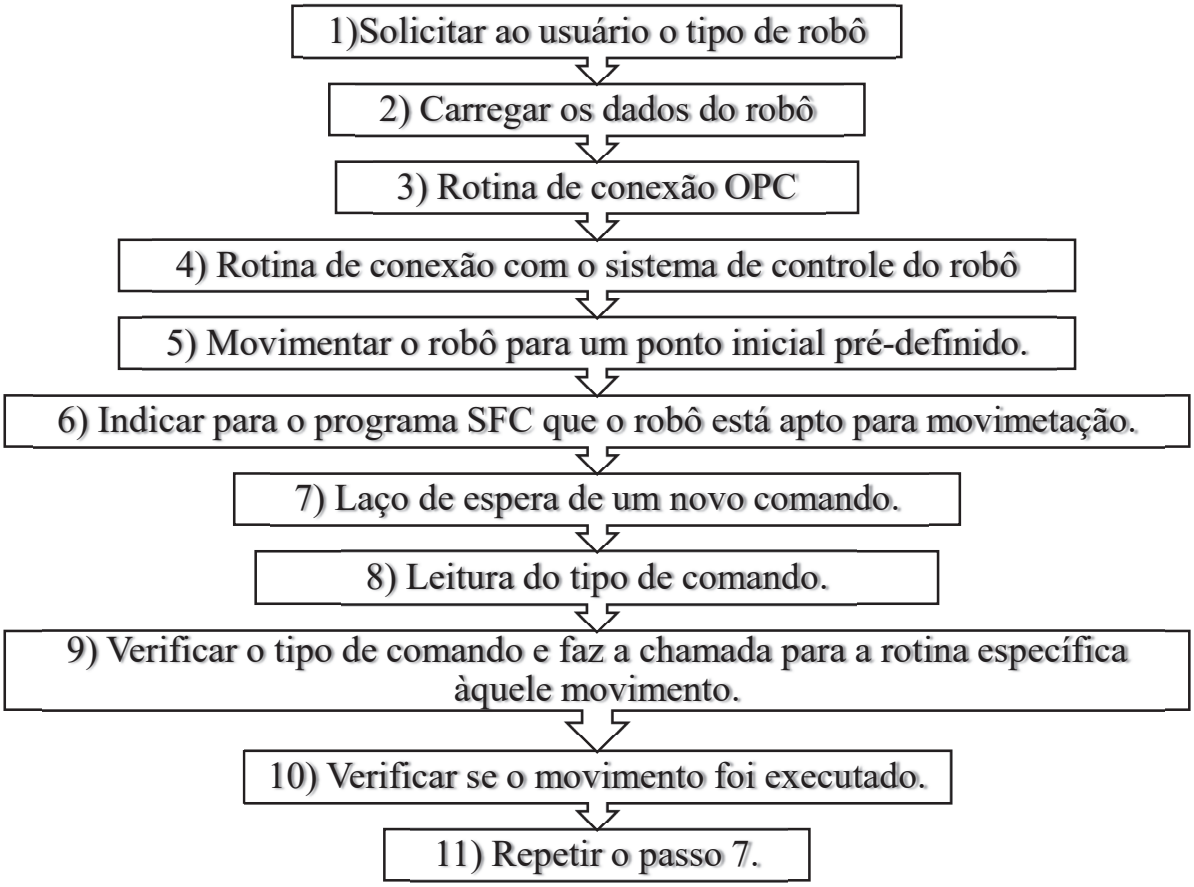

Figura 3.12 – Sequenciamento da rotina de comando.

4 PROGRAMAÇÃO DO ROBÔ PNEUMÁTICO DE 5 GRAUS DE LIBERDADE

No Capítulo 3 foi apresentada uma proposta de procedimento que torna possível a programação de robôs através da combinação da linguagem SFC e texto estruturado, ambas baseadas na norma IEC 61131-3. Tal estratégia é baseada no uso de um sistema cliente-servidor OPC para troca de informações entre um controlador lógico programável e de um programa que realiza a tradução de comandos da linguagem SFC para linguagem de programação de um dado robô.

Neste capítulo, com o intuito de testar a estratégia de programação proposta, é apresentado um estudo de caso realizado utilizando o robô cilíndrico de 5 GDL acionado pneumaticamente (RCP5). Primeiramente, é descrito o funcionamento do robô. Após, são apresentadas as rotinas escritas em SFC-R e o Programa Tradutor. No Capítulo 5, são apresentados exemplos completos de funcionamento do sistema implantado incluindo discussões sobre seus respectivos resultados.

4.1 Descrição do Robô

Conforme já comentado, o robô em estudo pode ser classificado como RPP:RR (junta rotacional na base, duas juntas ortogonais prismáticas e dois graus de liberdade rotacionais no punho (Figura 2.1). A Tabela 4.1 apresenta os valores máximos e mínimos de deslocamento para cada junta, [Sarmanho, 2014].

Tabela 4.1 – Valores máximos e mínimos de deslocamento por junta.

GDL	Deslocamentos	
	Mínimo	Máximo
1	-2,4652 rad (-141,21°)	+2,4652 rad (+141,21°)
2	0 m	0,45 m
3	0 m	0,30 m
4	-1,9809 rad (-113,5°)	+1,9809 rad (+113,5°)
5	-2,3562 rad (-135°)	+2,3562 rad (+135°)

Fonte: Adaptado de Sarmanho, 2014.

Com base nos valores da Tabela 4.1, Sarmanho, 2014, definiu o espaço de trabalho do robô pneumático (Figura 4.1). No domínio do espaço de trabalho, foi também definida uma região, como mostra a Figura 4.2, que pode ser representado por um paralelepípedo com dimensões de 1,0 x 0,4 x 0,55 m. Esses valores referem-se ao volume de trabalho relativo a um trabalhador braçal sentado operando confortavelmente sobre uma plataforma horizontal, segundo a ANBT 9050:2004 [ABNT,2004].

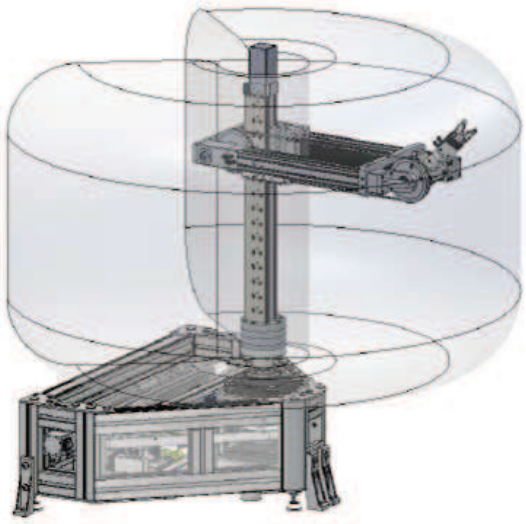

Figura 4.1 – Espaço de trabalho do robô pneumático.

Fonte: Adaptado de Missiaggia, 2014.

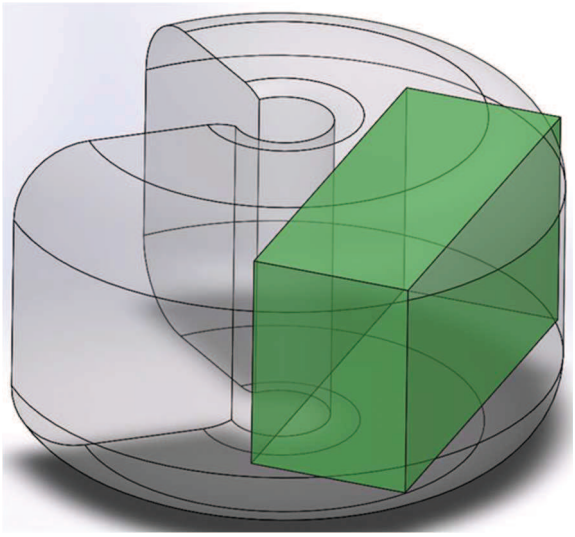

Figura 4.2 - Espaço de trabalho relativo a um trabalhador braçal.

Fonte: Sarmanho, 2014.

4.1.1 Cinemática

Esta seção é focada na análise cinemática do robô cujos resultados serão usados posteriormente no trabalho. A cinemática (direta e inversa) do RCP5 foi estudada por Rijo, 2013, Sarmanho, 2014, e Missiaggia, 2014, utilizando o método de Denarvit-Hartenberg, que consiste no método sistemático de representação das relações cinemáticas entre um elo e seus elos adjacentes, onde são necessários quatro parâmetros por elo para a definição completa do mesmo [Fu et al, 1987].

Esses estudos foram baseados em modelos geométricos simplificados do robô, como o apresentado na Figura 4.3. Oliveira, 2015, realizou um estudo experimental identificando as diferenças entre os parâmetros calculados por Missiaggia, 2014, e Sarmanho, 2014, e o sistema real a partir de medições. Nesse trabalho, verificou-se que Missiaggia, 2014, considerou a distância d_1 como o ponto onde o robô faz contato com o solo e que a distância d_5 é a distância até a coordenada central da garra utilizada como efetuador. Já, Sarmanho, 2014, considera d_1 como o ponto próximo ao solo onde inicia o segundo atuador e a distância d_5 como sendo a coordenada onde inicia o efetuador. Oliveira, 2015, considerou as duas abordagens em seu trabalho (focado em visão computacional) e, para facilitar, utilizou a referência d_1 proposta como Sarmanho, 2014, e a referência d_5 assim como proposta por Missiaggia, 2014.

Nos trabalhos supra citados, a cinemática direta foi equacionada a partir da aplicação do método de Denarvit-Hartenberg sobre o modelo simplificado apresentado na Figura 4.3. Como resultado, obteve-se os valores apresentados na Tabela 4.2.

Tabela 4.2 – Parâmetros de Denavit- Hartenberg.

GDL	$\alpha[rad]$	a	$\theta[rad]$	$d[m]$
1	0	0	θ_1	0
2	$-\pi/2$	0	0	d_2
3	$+\pi/2$	0	$+\pi/2$	0,1945+ d_3
4	$-\pi/2$	0	θ_4	0
5	0	0	θ_5	0,2273

Fonte: Adaptado de Oliveira, 2015.

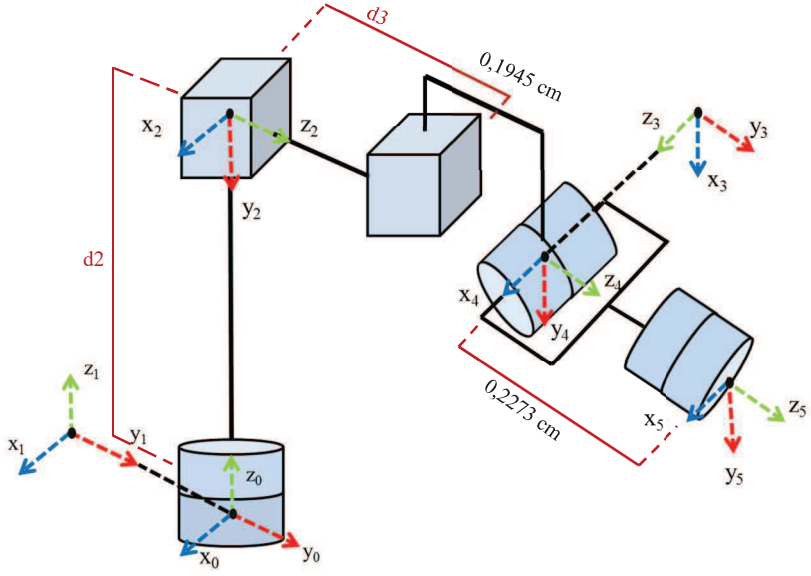

Figura 4.3 - Modelo simplificado dos elos do robô e representação gráfica dos parâmetros de Denavit-Hartenberg.

Fonte: Adaptado de Sarmanho, 2014.

Segundo Missiaggia, 2014, para determinar as coordenadas no efetuador do robô com relação ao sistema de coordenadas de sua base, multiplica-se as matrizes de transformação homogêneas de cada elo, como mostra a Equação (4.1).

$$ {}^{5}_{0}T = {}^{1}_{0}T\,{}^{2}_{1}T\,{}^{3}_{2}T\,{}^{4}_{3}T\,{}^{5}_{4}T \tag{4.1} $$

onde, ${}^{5}_{0}T$ é a matriz de transformação homogênea que representa a transformação do eixo de coordenadas do seu último elo até a sua base, enquanto que as matrizes ${}^{i+1}_{i}T$ para $i = 0,1 \ldots 4$, representam, cada uma, a transformação do sistema de coordenadas do elo i para o elo $i+1$.

Tal matriz fornece os dados de posição e rotação no efetuador como segue:

$$ {}^{5}_{0}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2} $$

onde os termos d_x, d_y e d_z representam a origem do sistema de coordenadas localizado no efetuador, enquanto que os temos r_{ij}, que representam a rotação com relação ao sistema de coordenadas cartesiano, são explicitados com relação aos ângulos de rotação dos elos (coordenadas de junta θ_1, θ_4 e θ_5) por meio da Equação 4.3 [Sarmanho, 2014].

$$\begin{bmatrix} -c\theta_1 s\theta_5 - c\theta_5 s\theta_1 s\theta_4 & s\theta_1 s\theta_4 s\theta_5 - c\theta_1 c\theta_5 & -c\theta_4 s\theta_1 & -0{,}2273 c\theta_4 s\theta_1 - (d_3 + 0{,}1945) s\theta_1 \\ c\theta_1 c\theta_5 s\theta_4 - s\theta_1 s\theta_5 & -c\theta_5 s\theta_1 - c\theta_1 s\theta_4 s\theta_5 & c\theta_1 c\theta_4 & 0{,}2273 c\theta_1 c\theta_4 + (d_3 + 0{,}1945) c\theta_1 \\ -c\theta_4 c\theta_5 & c\theta_4 s\theta_5 & s\theta_4 & 0{,}2273 s\theta_4 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_0^5 \quad (4.3)$$

O procedimento da cinemática inversa visa a encontrar os valores para as variáveis de junta, tendo em mãos a posição e a orientação do efetuador [Spong e Vidyasagar, 1989]. No presente trabalho, ela será usada para transformar os pontos de uma trajetória desejada para as coordenadas de junta do robô. Existem dois tipos de métodos de solução para a cinemática inversa, um baseado na interpretação analítica e outro baseado em uma solução numérica. Segundo Missiaggia, 2014, os termos da cinemática inversa do robô foram calculados por meio das relações trigonométricas apresentadas nas equações 4.7 a 4.9, baseadas no modelo simplificado da cadeia cinemática do robô, apresentado na Figura 4.4.

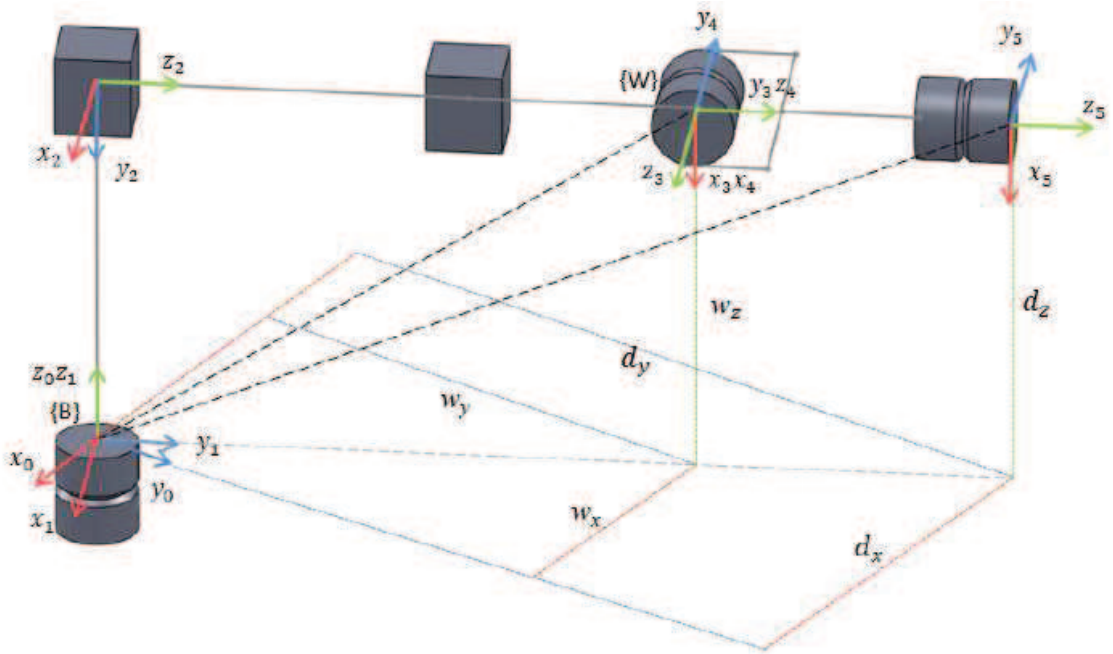

Figura 4.4- Representação simplificada da cadeia cinemática.

Fonte: Missiaggia, 2014.

As equações (4.7) a (4.9) foram obtidas por meio do método analítico, com base na Equação 4.3 e no modelo da Figura 4.4.

$$\theta_1 = -arcsen\left(\frac{d_x}{\sqrt{{d_x}^2 + {d_y}^2}}\right) \tag{4.7}$$

$$d_2 = d_z - 0{,}263 - 0{,}2273 sen\theta_4 \tag{4.8}$$

$$d_3 = \sqrt{{d_x}^2 + {d_y}^2} - 0{,}1943 - 0{,}2273 cos\theta_4 \tag{4.9}$$

4.1.2 Geração de Trajetória

Como já mencionado, Missiaggia, 2014, propôs uma estratégia para geração de trajetória com aproximação de pontos por *splines*, resultando em curvas de movimentação compostas por polinômios de 7ª ordem. Este procedimento foi realizado com o objetivo de suavizar a movimentação do robô, facilitando seu controle. A Figura 4.5 apresenta um exemplo de uma trajetória em posição gerada por Sarmanho, 2014, baseado no trabalho de Missiaggia, 2014, enquanto que na Figura 4.6 estão apresentadas as trajetórias de junta para os cinco graus de liberdade referentes ao movimento no espaço tridimensional apresentado na Figura 4.5.

No primeiro trecho da trajetória, o vetor de posições iniciais das juntas é $q_0 = [\pi/2 \;\; 0{,}04 \;\; 0{,}04 \;\; -\pi/2 \;\; -\pi/2]^T$ e o de posições finais é $q_f = [\pi/2 \;\; 0{,}42 \;\; 0{,}27 \;\; \pi/2 \;\; \pi/2]^T$. O segundo trecho é até $q_f = [0 \;\; 0{,}225 \;\; 0{,}15 \;\; 0 \;\; 0]^T$ completando assim um ciclo da trajetória.

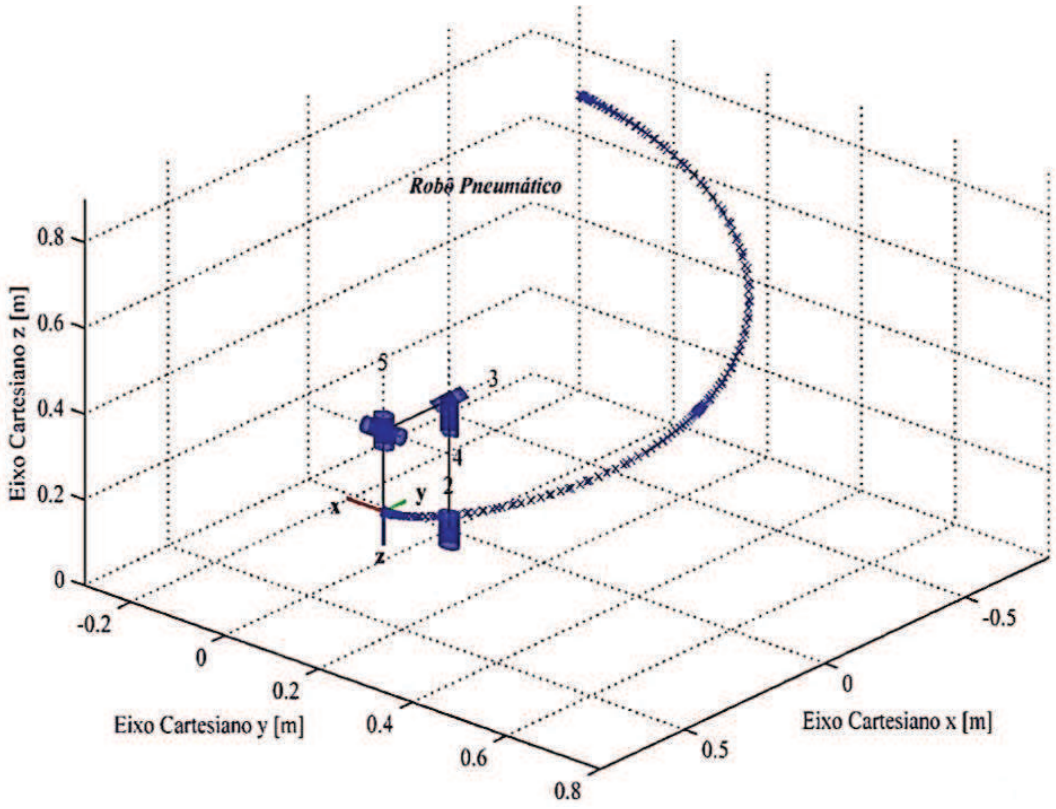

Figura 4.5- Trajetória Gerada por *splines*. (Fonte: Sarmanho, 2014.)

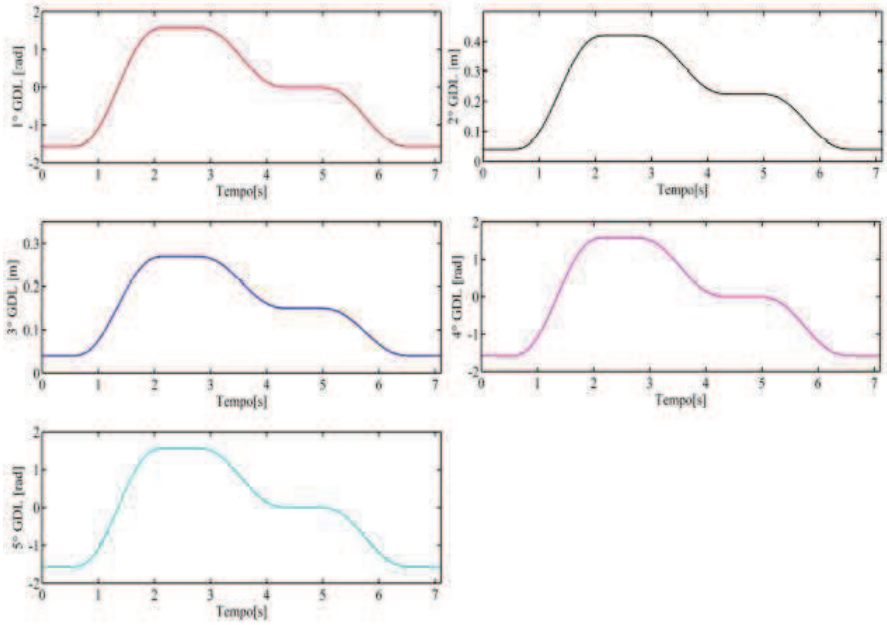

Figura 4.6-Referências de posição para os 5 graus de liberdade relativos a trajetória espacial apresentada na Figura 4.5.

Já, Oliveira, 2015 baseando-se no trabalho de Missiaggia, 2014, apresentou uma estratégia para automatizar a geração de trajetória para uma movimentação *pick and place* composta por 5 estágios de movimento. O 1º estágio consiste de uma trajetória vertical a partir do ponto inicial, a qual é seguida de uma trajetória em forma de arco para posicionar o robô horizontalmente (2º estágio). Na próxima etapa, o robô segue uma trajetória horizontal em direção à posição projetada do ponto final (3º estágio). Quando próximo a essa posição, é realizada mais uma trajetória em forma de arco para posicionar o robô acima do ponto final (4º estágio). Finalmente, o 5º estágio consiste na realização de uma trajetória vertical de descida até o ponto final. A Figura 4.7 apresenta um esquema representativo deste conceito de geração de trajetória *pick and place*.

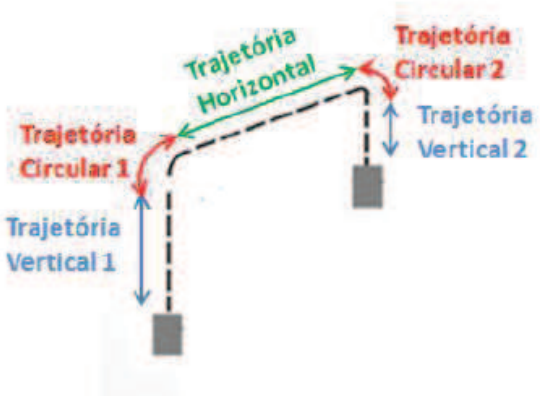

Figura 4.7 – Trajetória de *pick and place*.

Fonte: Oliveira, 2015.

Tal conceito é importante para o presente trabalho pois a movimentação *pick and place* faz parte da estratégia de programação adotada. Além disso, o estudo de visão computacional realizado por Oliveira, 2015, poderá ser, futuramente, integrado ao sistema de programação do robô pneumático utilizando a linguagem SFC-R.

4.1.3 Programação *off-line*

Sarmanho, 2014, propôs a aplicação de um controlador baseado em uma estratégia em cascata com compensação de atrito. Este controlador foi implantado experimentalmente por meio de uma estrutura de blocos utilizando o módulo Simulink Matlab®. Tal algoritmo, por

sua vez, é compilado na linguagem de processador (C) e carregado para o módulo de prototipagem de algoritmos de controle DS1104 da empresa dSPACE®, o qual é responsável por processar em tempo real o ciclo de controle do robô. Através de uma interface CAN[1] e SPI convertidas em um protocolo RS485, a plataforma DS1104 realiza a comunicação com as placas de condicionamento de sinais dedicadas para cada grau de liberdade do robô. Estas placas, chamadas de UCAM[2], são responsáveis por fazer o gerenciamento da leitura dos dados dos sensores de pressão e posição, bem como o acionamento das servo-válvulas pneumáticas que comandam a movimentação. A Figura 4.8 apresenta um esquema da interface de controle.

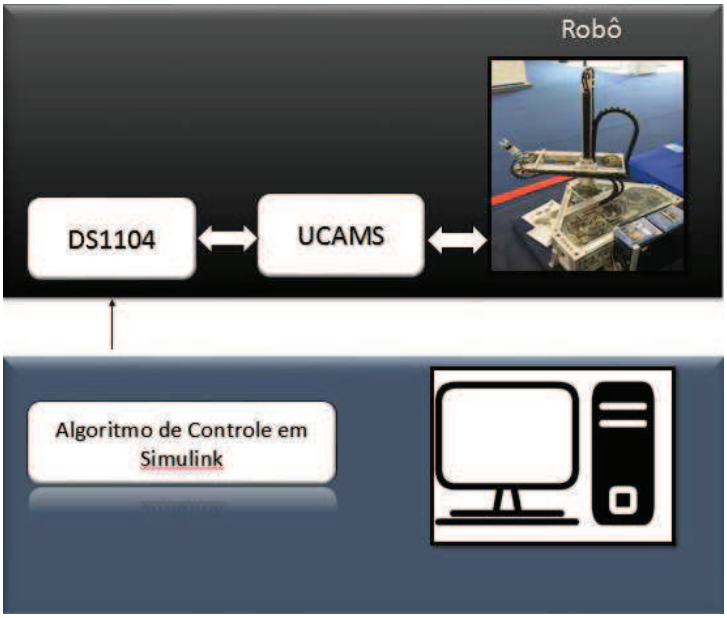

Figura 4.8- Esquema da arquitetura de controle do robô.

Uma vez compilado o código, o controlador passa a executar suas tarefas independentemente do Simulink/Matlab® e, em uma aplicação padrão, os parâmetros de controle só podem ser lidos e modificados através do software Control Desk. Portanto, para a programação do robô usando recursos usuais de operação, toda e qualquer programação deve ser realizada de forma *off-line*. O usuário, ou programador, fornece uma quantidade de parâmetros (como a quantidade de pontos desejados para trajetória, tempo entre pontos e estado do efetuador) e uma rotina em Matlab® cria um vetor contendo as coordenadas de junta para

[1] Interface CAN (Controller Area Network)

[2] UCAM (Unidade de Controle e Acionamento Microcontrolado)

cada instante de tempo da movimentação. Essas informações devem ser computadas e transmitidas juntamente com o código do controlador para a placa de controle. Dessa forma, para cada vetor de trajetória novo, o usuário deve compilar novamente o algoritmo de controle e o enviar para a DS1104. Essa forma *off-line* de programação desenvolvida por Sarmanho, 2014, utiliza os recursos *stardard* da DSpace e não permite, por exemplo, alteração online da trajetória além de necessitar do usuário expertise no uso da DSpace.

Diante de tais deficiências quanto à possibilidade de atualização das trajetórias desejadas, buscou-se uma maneira de realizar escrita e leitura *online* dos parâmetros do algoritmo de controle através de rotinas em linguagem "m" e em tempo real, ou seja, parâmetros como estabelecimento de uma nova trajetória, por conta de alteração das condições de operação, ou mudança *online* de referência de posicionamento ou de trajetória para, por exemplo, capturar uma peça identificada por visão computacional, poderão dessa forma ser realizados sem a necessidade de uma nova compilação do algoritmo de controle. A solução encontrada para tal desafio é apresentada, a seguir, na Seção 4.2.

4.2 Programação *Online*

Para demonstrar a capacidade de programação da estratégia proposta, por meio de estudo de caso, desenvolveu-se o sistema esquematizado na Figura 4.9, que baseia-se na integração do sistema de controle e programação do robô desenvolvido por Sarmanho, 2014, com o CLP, a programação em SFC e o servidor OPC.

Como elementos de hardware, além do CLP, utilizou-se um computador pessoal padrão IBM-PC compatível e a placa DS1104 da empresa dSPACE, além do RCP5. Para programação e integração entre os componentes, utilizou-se o software de programação para CLP RSLogix 5000, o software RSLinx para o gerenciamento OPC e o Matlab® para o desenvolvimento do Programa Tradutor.

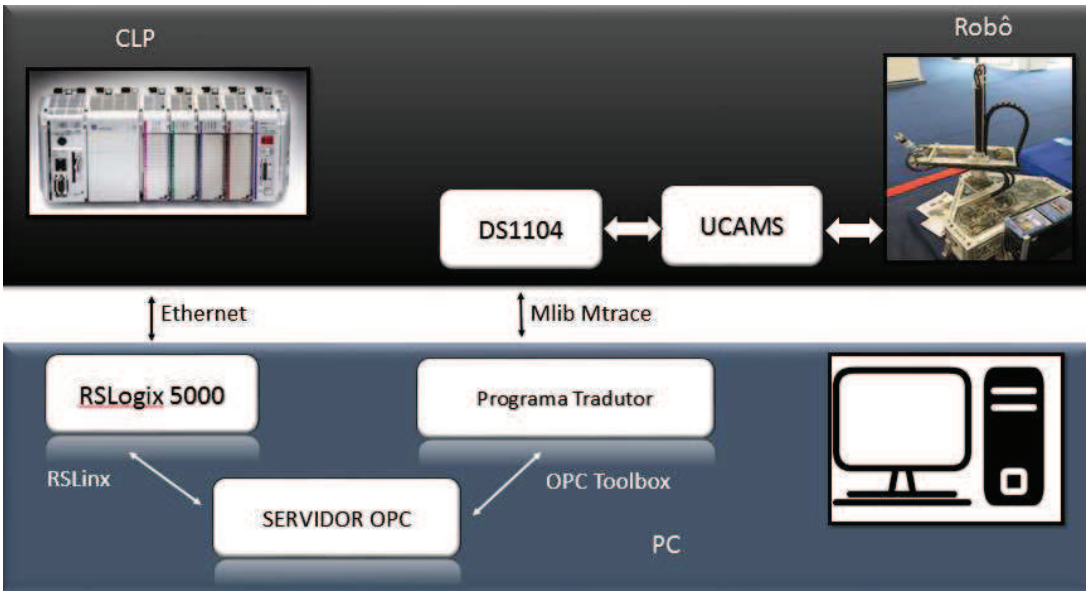

Figura 4.9 – Esquema do sistema de programação proposto.

O CLP se conecta à rede local utilizando o protocolo Ethernet por meio de um Protocolo de Internet (IP fixo). Conectado à mesma rede encontra-se o PC que opera com sistema operacional Windows 7. Nesse PC são processados o RSLogix 5000, o Matlab® e o RSLinx. A programação é feita em SFC utilizando o RSLogix e carregada no CLP. O servidor OPC, por sua vez, faz a conexão com o CLP por meio da rede Ethernet e utiliza um sistema de banco de dados para troca de informações com outros sistemas. Conectado ao mesmo servidor OPC está o cliente do Matlab®, onde se encontra o Programa Tradutor, que é responsável por ler e processar o programa em SFC, calculando as trajetórias necessárias às movimentações desejada. O mesmo programa, via Interconector de Barramentos Periféricos (PCI), encaminha as informações da trajetória para a placa de controle DS1104.

Os principais elementos que compõem a estratégia de programação proposta são detalhados nas seções que seguem.

4.2.1 Interface PC-Robô

A programação em SFC-R pode conter ações para movimentações diferentes. Portanto, para cada movimentação, uma nova trajetória deve ser transmitida para o algoritmo de controle do robô. Na estratégia de programação adotada por Sarmanho, 2014, cada nova ação proposta para o robô demanda um tempo considerável, tendo em vista que todo o algoritmo de controle

deve ser compilado e encaminhado para a placa de controle para cada diferente trajetória, inviabilizando decisões *on-line* a respeito da trajetória. Não é possível, por exemplo, implantar um sistema de manipulação de peças que chegam ao volume de trabalho em posições aleatórias (como no caso típico de uma esteira transportadora que têm suas coordenadas identificadas de maneira *on-line*).

Por isso, investigou-se maneiras de transmitir informação automaticamente para o controlador (o programa Control Desk®, que é o software disponibilizado pela dSPACE para este tipo de tarefa, permite apenas esta ação manualmente), tendo-se encontrado a biblioteca MLIB/MTRACE ® da própria dSPACE. Através de um conjunto de funções programadas em código "m", é possível ter acesso, via barramento PCI, às variáveis do algoritmo de controle previamente compilado para plataforma DS1104. Isto se dá porque, quando compilado, o algoritmo de controle gera um arquivo SDF (*System Data Format*) contendo as variáveis de controle do mesmo e suas respectivas atribuições nos registros de memória. A biblioteca MLIB possui as seguintes funções:

1) Verificação do estado da placa de controle: identifica quais placas estão conectadas ao computador.
2) Seleção da placa de controle: realiza a seleção de qual placa deseja-se trabalhar com as rotinas MLIB/MTRACE.
3) Vinculação para variáveis de um Arquivo Padrão de Informações (SDF): cria vínculos ou *links* entre variáveis do Matlab e variáveis do sistema de controle que está sendo processado na placa de controle.
4) Leitura e escrita de variáveis: através dos vínculos, é possível realizar a leitura e escrita de variáveis.
5) Captura de dados: a biblioteca também possui uma opção de realizar amostragem e armazenamento no disco rígido do PC das variáveis processadas na placa de controle.

Com estas funções, é possível programar um código que tem acesso às variáveis do algoritmo de controle que está sendo processado em tempo real na placa de controle, podendo-se alterá-las *on-line*, permitindo assim atualizar as trajetórias desejadas sem a necessidade de compilar o programa de controle com as trajetórias tratadas como vetores estáticos pré-programados.

4.2.2 CLP

O CLP utilizado para o trabalho foi o Compact Logix da Rockwell Automation, que possui 16 entradas e saídas analógicas, além de 16 entradas e saídas digitais. A comunicação com o PC pode ser realizada através de conexão USB ou Ethernet. Para o trabalho utilizou-se a conexão Ethernet, atribuindo um IP fixo ao CLP para que o mesmo pudesse ser acessado de qualquer computador conectado à rede Ethernet. O CLP é utilizado no presente trabalho para processar o programa em SFC-R e para tornar possível a integração do robô pneumático com outros elementos no meio de trabalho, tais como sensores, esteiras, pistões, etc.

4.2.3 Servidor OPC

O software utilizado para o gerenciamento do servidor OPC foi o RSLinx, o qual também é responsável pelo gerenciamento das conexões entre CLP e um PC hospedeiro da placa de controle. Com isso, através de poucos cliques, é possível criar um banco de dados no servidor OPC e associá-lo a um CLP correspondente. Através desta conexão, o servidor OPC tem acesso a todas as variáveis presentes no contexto do CLP, as quais são organizadas na hierarquia apresentada na Figura 4.10.

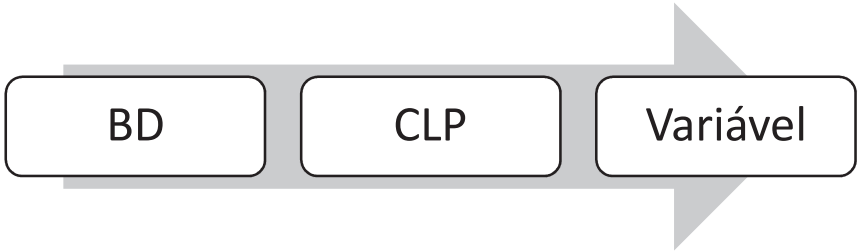

Figura 4.10- Hierarquia de dados no servidor OPC.

A um banco de dados está associada uma tabela correspondente a um CLP. Os itens desta tabela são as variáveis (*tags*) do programa que está sendo processado no mesmo.

Para o presente trabalho, criou-se um servidor OPC chamado de “RSLinx Remote Server”. Neste mesmo servidor, criou-se um banco de dados chamado de “OPCServer”, que está associado ao CLP do presente estudo, denominado de “CLP_ROBO” nesta estrutura de dados. Portanto o cliente OPC, quando devidamente conectado ao servidor “RSLinx Remote Server”, tem acesso às variáveis através da seguinte sintaxe:

“[OPCServer]CLP_ROBO.Nome_da_variável”

4.2.4 Sub-rotinas programadas em SFC-R

As sub-rotinas programadas em SFC-R foram configuradas visando a atender as necessidades do RCP5. Caso seja utilizado outro robô, outras sub-rotinas devem ser programadas e importadas para o ambiente de programação do CLP. Como mencionado anteriormente, usou-se a linguagem de texto estruturado para programar as sub-rotinas de ações de movimentação. Elas possuem funções como conversão de variáveis, cálculo de cinemática inversa, cálculo de pontos das trajetórias, entre outras. A seguir, na seção 4.2.4.1, essas sub-rotinas são detalhadas.

4.2.4.1 Sub-rotina mov_linear

Como descrita na seção 3.3.1, a sub-rotina de movimentação linear possui como parâmetros de entrada as coordenadas de posição e orientação tendo como origem o ponto extremo do efetuador. Também são parâmetros de entrada, o tempo de duração do movimento e o estado do efetuador (onde 1 representa um estado de garra fechada e 0 aberta). A sub-rotina escrita em texto estruturado apresenta uma lógica baseada na utilização de variáveis locais, ou seja, quando é realizada a chamada da sub-rotina, os parâmetros podem ser passados na forma de valores numéricos e a sub-rotina trata de atribuir os valores calculados para as variáveis locais.

A sub-rotina de movimentação linear realiza as seguintes funções:

1) Conversão para radianos: para facilitar a interface com usuários não especializados, os parâmetros de orientação do efetuador (θ_4, θ_5 e θ_6) são passados para a sub-rotina mov_linear com unidade em graus. O Programa Tradutor, por sua vez, utiliza como unidade angular o radiano, assim, é utilizada a função “RAD” para converter as variáveis.
2) Cálculo da cinemática inversa: como o algoritmo de controle do robô trabalha com coordenadas de junta, é necessário realizar o cálculo da cinemática inversa, descrita na Seção 4.1.1. Este cálculo, pode ser realizado opcionalmente por meio de uma sub-rotina do SFC-R escrita em texto estruturado ou vinculado ao próprio algoritmo de controle. No presente estudo, optou-se pela realização deste processamento através de uma rotina programada em texto estruturado, podendo, porém, sem prejuízo à operação do sistema proposto, ser executada pelo Programa Tradutor no PC.

3) Parâmetros de saída: os parâmetros de saída da sub-rotina mov_linear são as coordenadas do ponto de destino no espaço de juntas do robô: gl_01, gl_02, gl_03, gl_03, gl_04 e gl_05, o tempo previsto para realizar o movimento e o estado do efetuador.

4.2.4.2 Sub-rotina mov_pnp

A sub-rotina de movimentação *pick and place* possui 4 parâmetros de entrada: o ponto onde uma determinada peça será capturada, o ponto onde ela será deixada e o tempo previsto para trajetória entre os pontos. Para a geração de trajetória *pick and place*, a sub-rotina mov_pnp utiliza as seguintes funções:

1) Cálculo da cinemática inversa: o subprograma realiza o cálculo da cinemática inversa dos 4 pontos necessários ao movimento de *pick and place*. O ponto inicial e final são fornecidos pelo usuário. Por outro lado, os pontos auxiliares (localizados em uma posição customizada localizada acima dos pontos inicial e final) são declarados na sub-rotina já como coordenadas de junta. O valor *standard* é um *offset* de $+10\ cm$ no eixo z.
2) Conversão para radianos: assim como no subprograma de movimentação linear mov_linear, é necessário fazer a conversão dos ângulos (em graus) para radianos utilizando a função RAD.
3) Parâmetros de saída: os parâmetros de saída são as coordenadas no espaço de juntas para os 4 pontos descritos na Tabela 4.3, o tempo desejado para movimentação entre os pontos.

A nomenclatura adotada para referenciar os pontos de uma movimentação *pick and place* é, genericamente, Gl_0ij, onde $i = 1 \ldots 5$ representa o grau de liberdade e $j = 1 \ldots 4$ a referência aos 4 pontos Pj que definem o movimento.

As coordenadas de junta do robô são declaradas separadamente, pois o software RSLogix 5000 não permite a possibilidade de se trabalhar com variáveis do tipo vetor de variáveis (arrays).

Tabela 4.3- Nomenclatura adotada para referenciar os pontos da movimentação *pick and place*.

Grau de Liberdade	P1	P2	P3	P4
1º	Gl_011	Gl_012	Gl_013	Gl_014
2º	Gl_021	Gl_022	Gl_023	Gl_024
3º	Gl_031	Gl_032	Gl_033	Gl_034
4º	Gl_041	Gl_042	Gl_043	Gl_044
5º	Gl_051	Gl_052	Gl_053	Gl_054

4.2.4.3 Sub-rotina mov_pallet

O cálculo dos pontos de paletização envolvem operações matriciais e, por este motivo, optou-se por realizá-los no Programa Tradutor. Dessa forma, a sub-rotina mov_pallet permite que o usuário possa indicar os parâmetros do pallet, que são as dimensões do produto, o número de linhas, colunas e andares, e o ponto de referência inicial (P0), mas não realiza qualquer cálculo ou operação com estes parâmetros.

4.2.5 Programa Tradutor em Matlab®

Utilizando a estratégia apresentada no Capítulo 3, foram programadas diversas rotinas em linguagem “m” para gerenciar as conexões entre o CLP, o PC e o Robô e permitir a execução do controle dos movimentos. O objetivo da programação de cada comando por meio de rotinas separadas é tornar o Programa Tradutor modular, de forma que, posteriormente, estas rotinas possam servir para programar outros robôs. As rotinas programadas para o RCP5 são apresentadas a seguir na Seção 4.2.5.1.

4.2.5.1 Rotina Principal

A rotina principal (*Main*) é a que faz o gerenciamento do Programa Tradutor e comunicação com os demais componentes do ambiente de trabalho. Ela realiza o monitoramento das variáveis de controle e as chamadas para as rotinas auxiliares, quando necessário. É dividida em duas partes: a primeira realiza a seleção do robô e carregamento de dados do mesmo, conexão com o módulo dSPACE, conexão com o servidor OPC e

movimentação para um ponto de espera. O programa como um todo foi projetado para o robô do estudo de caso, havendo a necessidade de fazer alterações na rotina de comando para que a mesma possa operar com outros sistemas.

A primeira parte da rotina *Main* apresenta o seguinte sequenciamento:

1) Executa a rotina de inicialização e carregamento de dados.
2) Executa a rotina de conexão com o servidor OPC.
3) Executa a rotina de conexão com o a Placa de Controle.
4) Realiza a movimentação do robô até o ponto de espera por meio do comando:

$$P_{espera}(x, y, z, \theta_4, \theta_5) = (0, (0{,}6218), (0{,}4630), 0, 0)$$

que representa o ponto :

$$P_{espera}(\theta_1, d_2, d_3, \theta_4, \theta_5) = (0, (0{,}2), (0{,}2), 0, 0)$$

no espaço de juntas.

5) Avisa ao usuário que a inicialização está completa e solicita ao mesmo que forneça um comando para iniciar a execução da segunda parte do programa.

Já, a segunda parte da Rotina de Comando apresenta o seguinte sequenciamento:

1) Coloca a variável “robot_ok” em 1, indicando que o robô está apto para um novo movimento.
2) Uma função do tipo “repete enquanto” (do-while) executa recursivamente o laço, aguardando até que a variável “new_cmd” seja igual a 1.
3) Faz a leitura da variável “cmd_type”.
4) Passa para o valor 1 a variável “cmd_accepted” e para o valor 0 a variável “robot_ok”.
5) Chama uma função do tipo “seleção de caso” (ou switch case) com base no valor da variável “cmd_type”.

   - Caso seja 1, faz a chamada para rotina de movimentação linear;
   - Caso seja 2, faz a chamada para rotina de movimentação de *pick and place*;
   - Caso seja 3, faz a chamada para rotina de paletização;
   - Caso seja 10, realiza a movimentação para o ponto de espera, encerrando o laço e solicitando ao usuário um comando indicativo se o mesmo deseja encerrar o programa ou iniciar uma nova ação.

6) Executa novamente a ação a partir da etapa 1), a não ser que o comando dado através da variável “cmd_type” seja o de parar o robô e ir para o ponto de espera (cmd_type=10).

4.2.5.2 Rotina de Inicialização e Carregamento de Dados: *Robot_Data*

Ao iniciar o Programa Tradutor, a rotina *RobotData* é a primeira a ser executada, permitindo que o usuário possa escolher entre as seguintes opções: carregar, de uma lista pré-definida, uma estrutura de dados associados ao tipo de robô; inicializar o programa; ir para o modo de configurações, onde o usuário pode inserir um novo robô e suas rotinas; e, por último, o usuário tem a opção de ir para o modo manual, onde é possível movimentar o robô através de botões associados à movimentos de cada grau de liberdade e adquirir pontos.

Por enquanto o único robô disponível para o carregamento de dados é o RCP5. Dependendo do tipo de robô, outras rotinas de movimentação, gerenciamento do sistema de controle, aquisição da posição, dentre outras, devem ser acrescentadas. O sistema está configurado para que outros robôs possam ser integrados ao sistema.

A rotina aquisição de pontos foi programada em código “m”. Por meio de uma interface gráfica, o usuário pode mover individualmente com cliques cada grau de liberdade do robô e salvar cada ponto desejado no arquivo onde se encontram as “tags” da programação SFC-R. Assim, quando o usuário importar as “tags”, levará junto todos os pontos adquiridos por meio deste processo.

4.2.5.3 Rotina de Conexão e Configuração da Plataforma dSPACE: *Connect_dSPACE*

A rotina de conexão e configuração do módulo dSPACE executa as seguintes funções:

1) Verifica quais as plataformas conectadas ao PC hospedeiro e atribui nomes às mesmas.
2) Seleciona placa de controle (DS1104) e realiza a conexão com a mesma.
3) Por último, através da função “GetTrcVar”, as variáveis do algoritmo de controle desejadas são conectadas às variáveis no ambiente Matlab.

Desta forma, é possível monitorar, ler e escrever nas variáveis de contole do algoritmo de programação em tempo real, o que só era possível no software Control Desk utilizando recursos padrões da DSpace.

4.2.5.4 Rotina de Conexão e Configuração do Servidor OPC: *Connect_OPC*

A rotina *Connect_OPC* executa as seguintes funções:

1) Solicita que o usuário forneça o nome e IP do servidor OPC com o qual se deseja realizar a conexão.
2) Com base nos dados fornecidos pelo usuário, o subprograma realiza conexão. Caso aconteça algum erro, o programa retorna uma mensagem indicando sua ocorrência.
3) Adquire *namespace* do servidor OPC, que é o conhecimento (*acknowledge*) de todas as variáveis presentes no banco de dados.
4) Cria, automaticamente, um grupo de leitura e escrita, de forma a adicionar somente as variáveis de interesse do Programa Tradutor. Este grupo é denominado de "grp".
5) Ao grupo "grp" são adicionadas todas as variáveis necessárias ao movimento, descritas na Tabela 3.1, além das descritas como parâmetros de saída para cada sub-rotina do SFC-R. O caminho para referenciar uma variável no banco de dados segue a seguinte estrutura:

"Nome do servidor OPC: Nome do CLP: Tipo de variável: Nome da variável"

Foram atribuídos nomes no ambiente Matlab para as variáveis do servidor OPC para uso na programação. O tipo é o *struct*, ou seja, para atribuir ou ler o valor da variável, é necessário utilizar a estrutura: "Nome_da_variável.Value".

4.2.5.5 Rotina de Ação de Movimento: *Goto*

A rotina *Goto* é responsável por fazer o gerenciamento da movimentação do robô. Ela realiza a operação de ida para um ponto, atualizando a trajetória e tipo de referência executando as seguintes funções:

1) O vetor contendo os dados referentes às posições, velocidades e acelerações da trajetória gerada é atualizado na placa de controle (DSpace).
2) O tipo de referência do robô é alterado de estática para dinâmica, para que o mesmo passe a operar com o vetor de posições, velocidade e acelerações atualizados. Em cada ciclo de controle que ocorre durante o movimento do robô, um novo vetor de referências é atualizado.

3) Verifica se a movimentação finalizou. Quando isto ocorre, o tipo de referência é alterado novamente para estático, fixando o último valor de referência de posição velocidade e aceleração.

4.2.5.6 Rotina de Movimentação Linear: *Go_Linear*

1) Indica que o comando foi aceito, colocando nível lógico 1 na variável "cmd_accepted".
2) Lê as posições e orientações de junta do ponto final desejado através do servidor OPC.
3) Captura, através de funções do MLIB, as posições e orientações de junta atuais do robô e toma estes valores como ponto inicial para geração de trajetória.
4) É realizada a chamada da função de geração de trajetória e rotina de ação de movimento.
5) Indica o término da movimentação colocando em nível lógico 1 a variável "cmd_executed".

4.2.5.7 Rotina de Movimentação Pick and Place: *Go_pnp*

1) Indica que o comando foi aceito, colocando a variável "cmd_accepted" no nível lógico 1.
2) Lê os parâmetros de saída da sub-rotina SFC-R através do servidor OPC.
3) Lê a posição atual do robô para tomar como ponto de partida.
4) Executa o laço de ações de movimentação descrito na Figura 4.11:

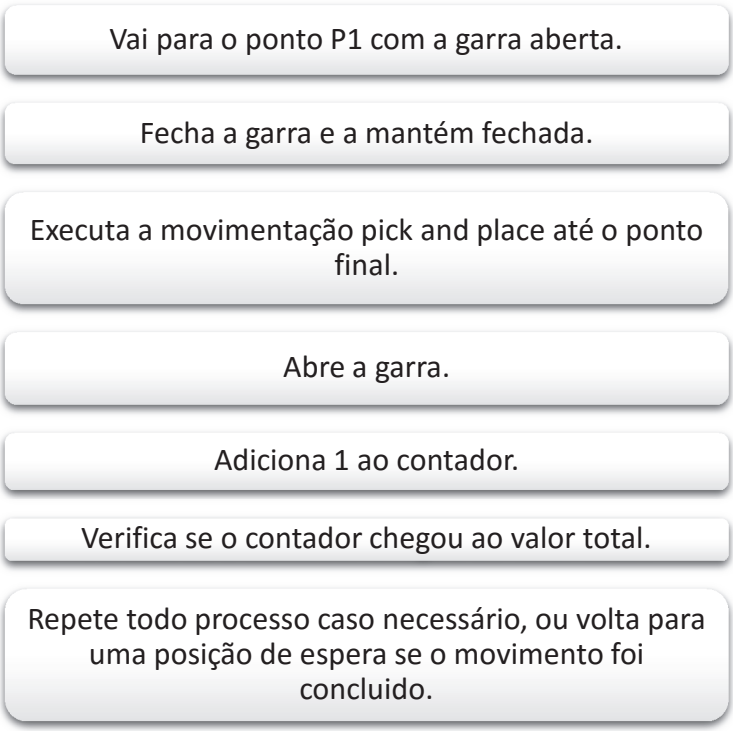

Figura 4.11- Laço de movimentação

5) Indica o término da movimentação colocando em nível lógico 1 a variável "cmd_executed".

4.2.5.8 Rotina de Paletização: *Go_Pallet*

A rotina *Go_Pallet* é a responsável pelo cálculo dos pontos do pallet com base nos parâmetros passados pelo usuário. Além disso, ela ordena as movimentações para os pontos de forma que o robô entregue a peça ou produto sempre por cima, assim como em uma trajetória de pick and place.

A Figura 4.12 apresenta o exemplo de um pallet 2x2x2 [Fonte: Dreamstime, 2016] com caixas de 90 cm com 10 cm de folga. O ponto que representa o vértice da origem do sistema de coordenadas do *pallet* pode ser adquirido por meio do processo manual descrito na Seção 4.2.5.1, ou indicado diretamente pelo usuário como parâmetro em programa SFC.

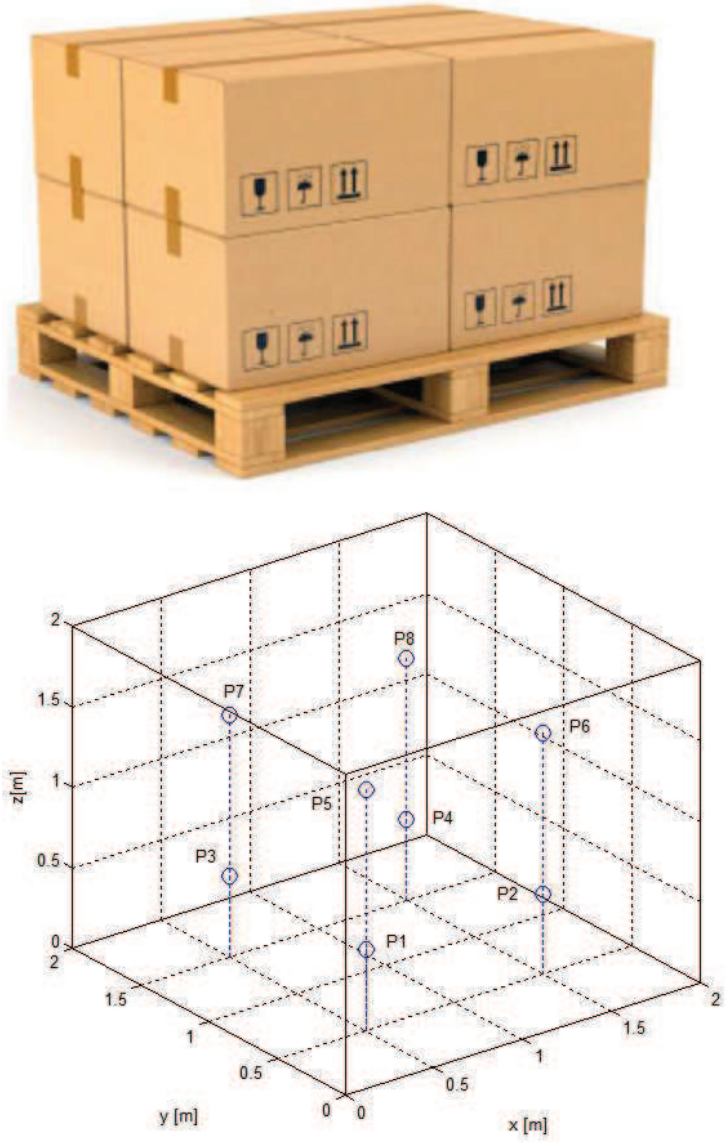

Figura 4.12 – Exemplo dos *keyoints* desejados para um pallet 2x2x2.

Os parâmetros linhas, colunas e andares servem de base para montagem do vetor de pontos do pallet da seguinte maneira:

Em x:

$$x[n] = \frac{(dimProdx)}{2} + (n)(dimProdx) \ .: n = 0,1,2 \dots linhas - 1 \tag{4.10}$$

Em y:

$$y[n] = \frac{(dimPrody)}{2} + (n)(dimPrody) \ .: n = 0,1,2 \dots colunas - 1 \tag{4.11}$$

Em z:

$$z[n] = \frac{(dimProdz)}{2} + (n)(dimProdz) \ .: n = 0,1,2 \ldots andares - 1 \qquad (4.12)$$

onde $dimProdx, dimPrody \ e \ dimProdz$ são as dimensões do produto.

A rotina *Go_pallet* apresenta o seguinte sequenciamento:

1) Indica que o comando foi aceito, por meio da passagem para nível lógico 1 da variável "cmd_accepted".
2) Lê os parâmetros de saída da sub-rotina *mov_pallet* através do servidor OPC.
3) Com base nos parâmetros acima, calcula os pontos chaves da ação de paletização.
4) Executa a seguinte ordem de movimentação para cada elemento a ser armazenado.
   - Dirige-se ao ponto de captura do elemento.
   - Movimenta-se para o ponto correspondente do elemento utilizando uma trajetória do tipo *pick and place*.
5) Quando terminado, indica que o comando foi executado passando para nível lógico 1 a variável "cmd_executed".

5 EXEMPLOS, RESULTADOS E DISCUSSÕES

Este capítulo é destinado à apresentação de aplicações do sistema desenvolvido visando a demonstrar a efetividade da metodologia proposta no presente trabalho e do sistema desenvolvido para programação do RCP5. Em um primeiro exemplo, é apresentada a programação de um caso que permite utilizar os três tipos de movimentação disponibilizados. Já, em um segundo caso, é simulado o funcionamento de uma célula fabril, onde elementos como pistões e sensores são integrados com o RCP5 que executa uma operação de *pick and place*.

5.1 Exemplo 1 - Escolha de movimentação

Este exemplo tem como objetivo descrever com detalhes a aplicação dos comandos relativos aos três movimentos anteriormente apresentados. Para condensar a programação, ao invés de compor 3 algoritmos diferentes, utilizou-se um caso onde há um nó de decisão (logo após a etapa “example_Step_001”). Assim é processado o código de uma determinada ramificação baseado em algum parâmetro determinado de seleção. As figuras 5.1 a 5.5 representam graficamente os módulos que compõem programa em SFC-R deste caso.

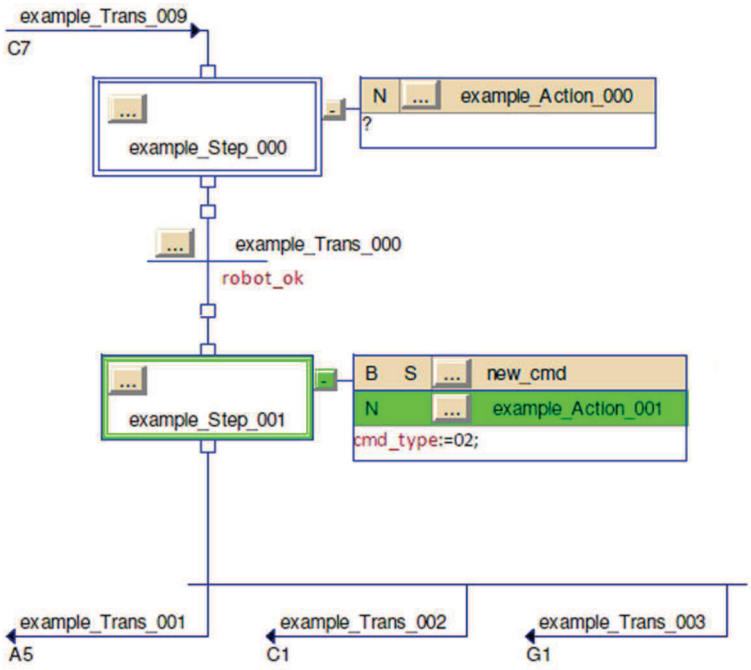

Figura 5.1- Início do programa.

A primeira parte, apresentada na Figura 5.1, é composta pela etapa inicial representada por um retângulo com uma borda, seguida de uma transição que depende da variável “robot_ok”, que é comandada pelo Programa Tradutor. Após, é processada uma etapa onde duas ações são executadas: a primeira aciona a variável “new_cmd”, responsável por indicar um novo comando; já, a segunda ação indica qual tipo de movimentação é desejada através da variável “cmd_type”. Por fim, tem-se um nó de seleção indicando as três possíveis transições descritas a seguir. A primeira refere-se à seleção do movimento linear, de acordo com a Figura 5.2.

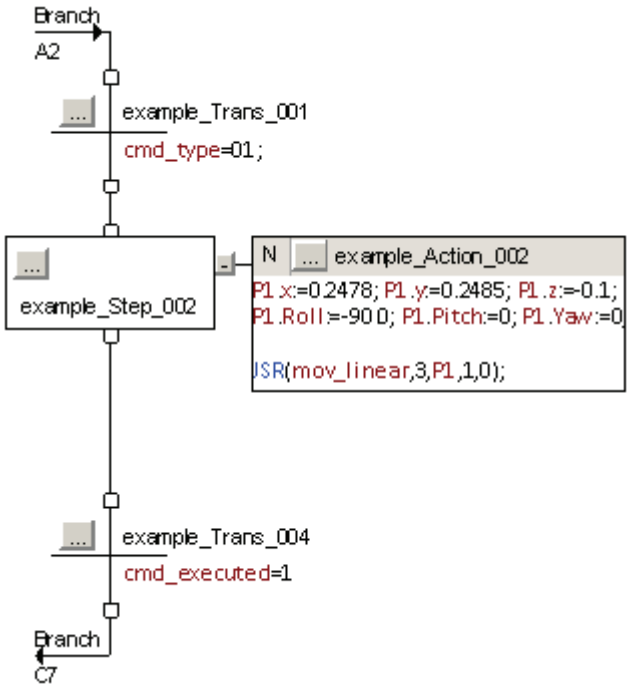

Figura 5.2- Movimento linear.

Conforme a Figura 5.2, a transição para a rotina de movimentação linear é comandada pela variável “cmd_type=01”, ou seja, quando o processamento alcançar o nó de seleção, o mesmo irá seguir pelo caminho apresentado na Figura 5.2 quando a variável descrita assumir o valor 01. Seguida da transição, encontra-se a ação de chamada para a sub-rotina que executa o movimento. Primeiramente, é fornecido o ponto desejado (que pode, por exemplo, ser adquirido via Programa Tradutor) associado às coordenadas de posição, as quais, no exemplo proposto,

são $x = 0.2478\ cm$, $y = 0.2485\ cm$ e $z = -0.1\ cm$, enquanto que as de orientação são: $\theta_4 = -90°$, $\theta_5 = 0°$ e $\theta_6 = 0°$, que através das equações 4.7 a 4.9 resultam nas coordenadas de junta: $q[gl1\ gl2\ gl3\ gl4\ gl5] = [-0{,}78\ \ 0{,}05\ \ 0{,}15\ \ -1{,}57\ 0]$. É então acionada a sub-rotina mov_linear com o fornecimento dos seguintes parâmetros que definem características do movimento, conforme a Seção 3.3: a primeira informação é o número de parâmetros de entrada da sub-rotina que está sendo chamada, no caso do movimento linear são 3 (ver Seção 3.3.1); a segunda até a sexta informação referem-se à estrutura de dados da posição desejada (Ponto_1); a terceira informação é a do tempo desejado para o movimento (1 segundo); a quarta informação é o estado da garra ("0" aberta e "1" fechada, no caso a garra será aberta, usando-se, portanto, "0"). Após esse processamento, uma transição garante que o programa só prossiga caso o comando seja executado pelo robô.

A segunda ramificação do nó de seleção da Figura 5.1 pode ser visualizada na Figura 5.3. Ela representa um exemplo de movimentação *pick and place*. Analogamente à primeira ramificação, o programa só seguirá por esta rotina caso a variável "cmd_type" seja igual ao valor "02". Após a transição, é processada a etapa onde esta sub-rotina é chamada. Os pontos para ação de movimentação são declarados e a sub-rotina "mov_pnp" é chamada com a passagem de 4 parâmetros: o ponto inicial P1 e final P2, a altura (10 cm) e o tempo previsto para a movimentação (3 segundos). Em seguida, uma transição garante que o programa só prossiga para as próximas etapas do laço principal caso o movimento seja executado.

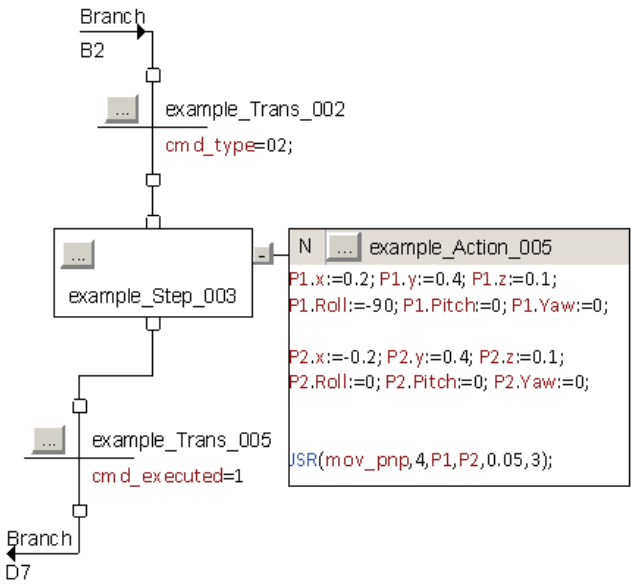

Figura 5.3 –Movimento *pick and place*.

A terceira ramificação do nó de seleção representa uma movimentação completa de paletização, cuja programação está representada por meio do diagrama ilustrado na Figura 5.4. Da mesma forma que nos casos anteriores, a transição para esta sub-rotina ocorre quando a variável “cmd_type” tem o valor igual à “03”. Nessa rotina, a primeira etapa corresponde à declaração das coordenadas do ponto inicial do pallet (Ponto_0) e do ponto de captura de objeto (Ponto_C). Em seguida, ocorre a chamada para a sub-rotina “mov_pallet” com a passagem de 8 parâmetros: o primeiro é o ponto inicial P0, o segundo é o ponto de captura PC. Os três seguintes indicam as dimensões do produto (no caso, um cubo de 1m, incluindo folga). Os últimos 3 parâmetros definem a topologia da matriz de paletização (no caso, uma base de 2x2 e dois andares). A seguir é feita uma verificação se o comando foi recebido, sendo acionada a etapa de início do movimento através do estado da variável “pallet_start”. A transição que segue esta etapa garante que o programa só siga seu fluxo quando o Programa Tradutor informar que a movimentação foi interrompida ou finalizada.

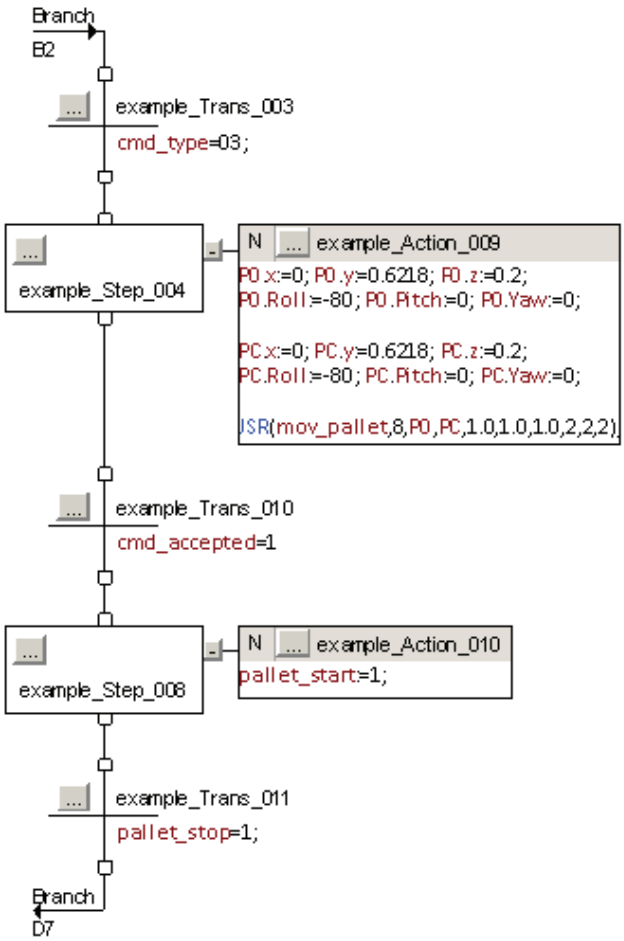

Figura 5.4 –Movimento de paletização.

A parte final do programa SFC-R (Figura 5.5) consiste no encontro das três ramificações após a movimentação ter sido realizada. Essa etapa é responsável por fazer a reinicialização da variável “new_cmd” e o retorno para a etapa inicial.

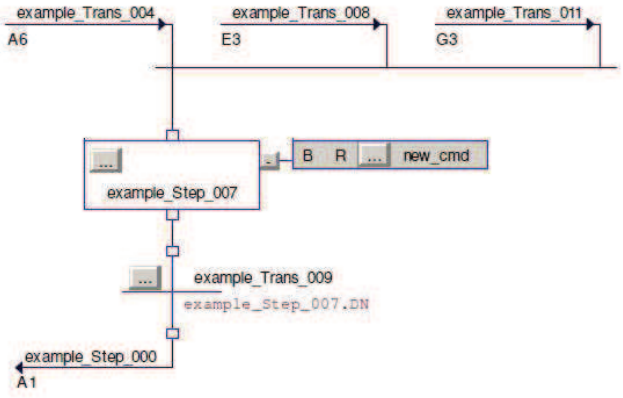

Figura 5.5 – Etapa final do código SFC-R.

A Figura 5.6 apresenta os gráficos de posição do robô em comparação com a trajetória de referência gerada pelo Programa Tradutor, para o caso do movimento linear.

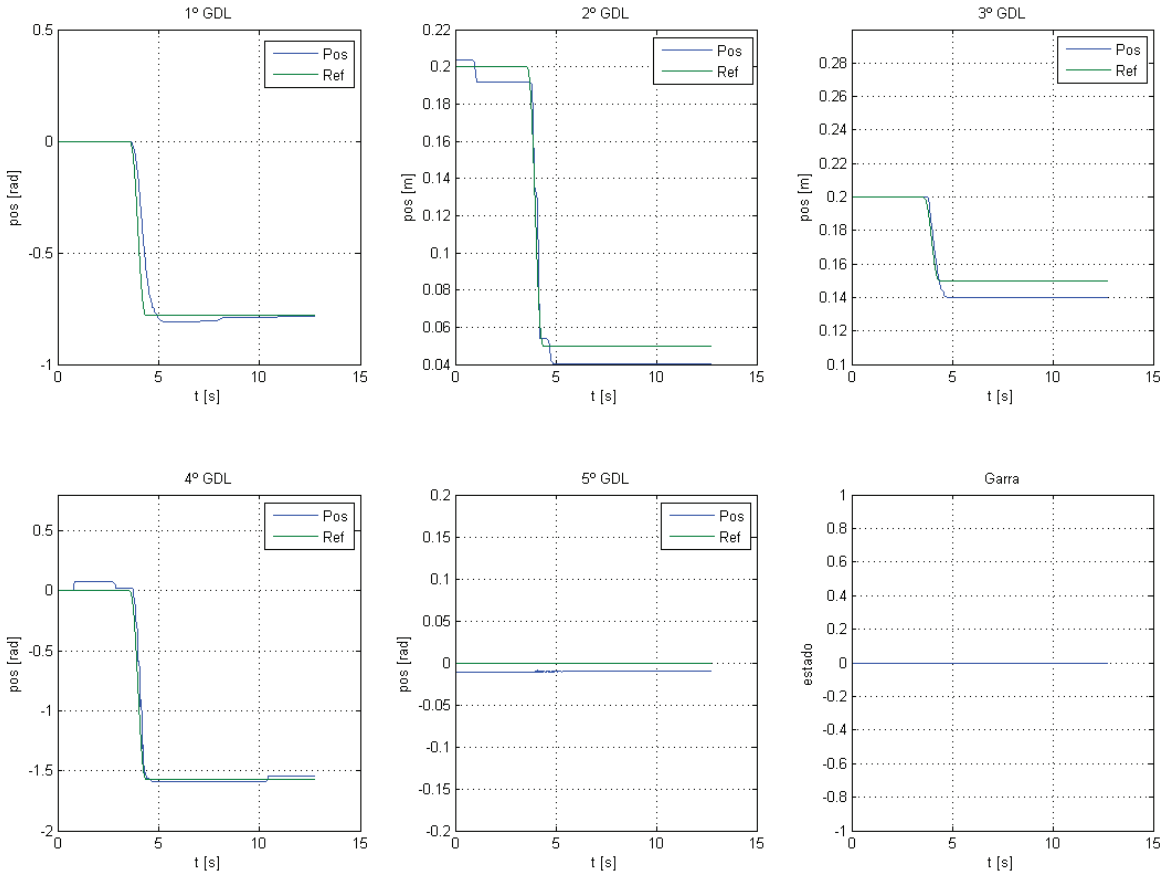

Figura 5.6 – Gráficos de posição em comparação com referência para cada grau de liberdade.

O que se pode perceber pela Figura 5.6 é o Programa Tradutor gerou e passou corretamente a trajetória com origem nas coordenadas do ponto de espera do robô até as coordenadas do ponto desejado para a movimentação programada em SFC-R.

A Figura 5.7 apresenta os gráficos que ilustram as referências de posição em comparação com posição do robô para cada grau de liberdade associados a trajetória de *pick and place* gerada pelo Programa Tradutor, em sua interpretação da programação feita em SFC-R.

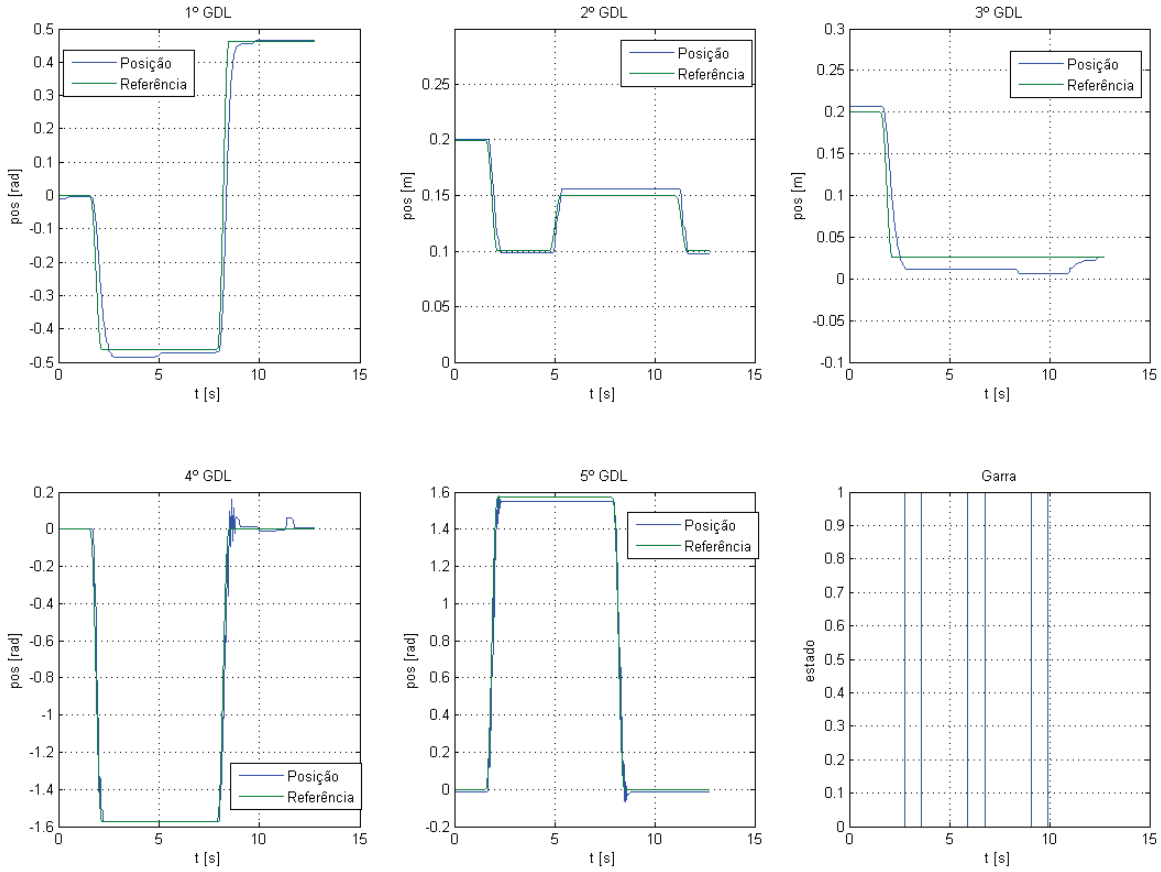

Figura 5.7 – Gráficos de comparação de posição do robô com suas respectivas referências para movimento de *pick and place*.

Nota-se pela Figura 5.7 que o Programa Tradutor gerou e passou corretamente a trajetória de movimentação para o robô. As coordenadas de posição para os pontos 1 e 2 foram conferidas através das equações 4.7 a 4.9 e estão corretas. No gráfico do segundo grau de liberdade é possível perceber que o robô desce da coordenada relativa ao ponto de espera para a coordenada relativa ao ponto 1 e, após um tempo, sobe 5 cm no eixo z e permanece nesta posição até o robô chegar próximo ao ponto 2 e então desce os mesmo 5 cm no eixo z até chegar na coordenada

relativa ao ponto 2, completando a movimentação. Isto mostra que o Programa Tradutor interpretou corretamente o valor da altura passada pela programação SFC-R.

Finalmente, a Figura 5.8 apresenta os gráficos que ilustram as referências de posição do robô para cada grau de liberdade associados a trajetória de paletização gerada pelo Programa Tradutor, em sua interpretação da programação feita em SFC-R. Neste caso, optou-se por não adquirir os dados dos sensores de posição para cada grau de liberdade do robô por questões visuais, visto que este tipo de tarefa conta com um número grande de movimentações.

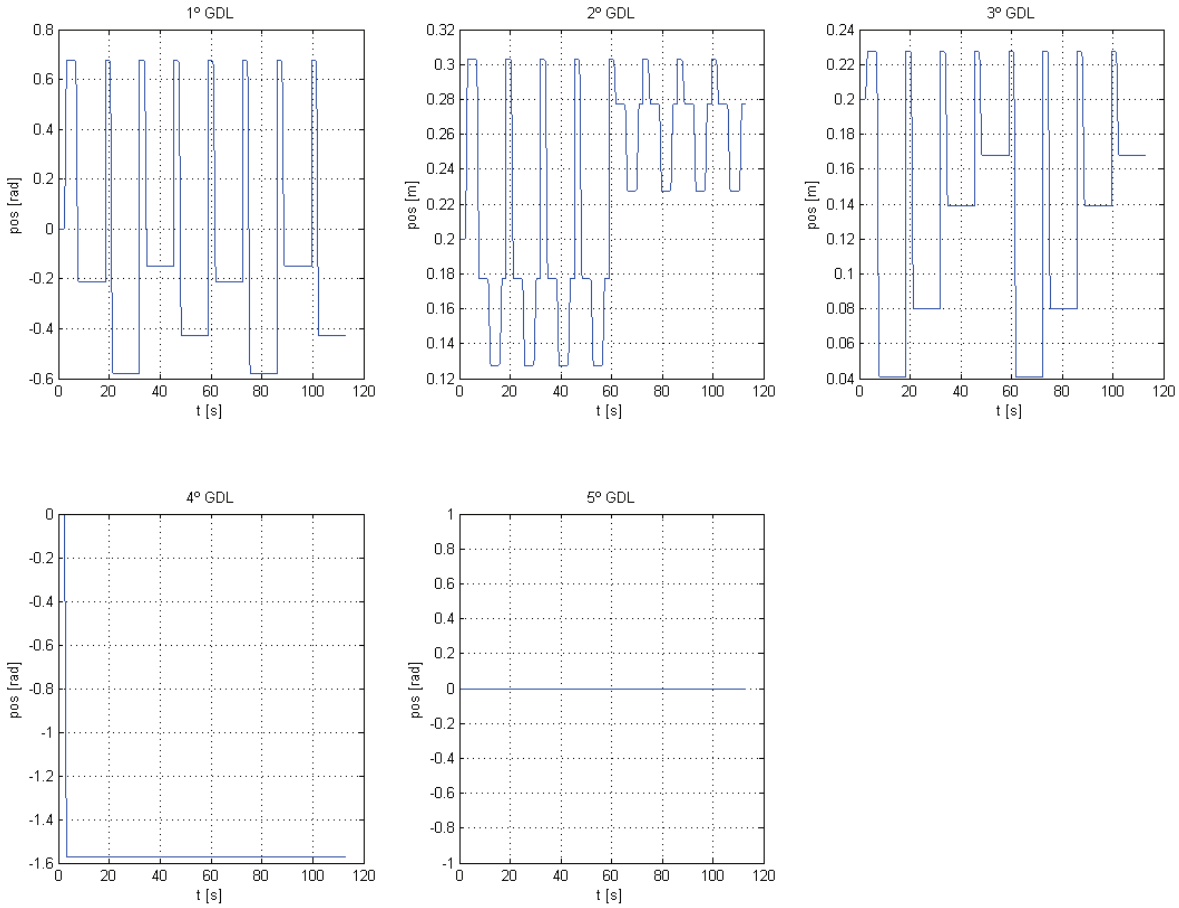

Figura 5.8-Gráficos de posição para cada grau de liberdade do robô para trajetória de paletização desejada.

Utilizando as equações 4.10 a 4.12, em conjunto com as equações da cinemática inversa do robô (equações 4.7 a 4.9), verificou-se as coordenadas posição de junta para o robô associadas ao pallet com os parâmetros apresentados na programação SFC-R que pode ser visualizada na Figura 5.4 e, comparando com os gráficos de posição de junta apresentados na

Figura 5.8, pode-se concluir que o Programa Tradutor gerou com sucesso a trajetória do movimento de paletização.

5.2 Exemplo 2 - Simulação de uma célula fabril

Este exemplo foi idealizado para demonstrar a capacidade de comandar um robô integrado a um sistema automático fabril por meio de um único código programado utilizando o SFC-R. Assim, elementos como pistões, sensores e esteiras tem seu comportamento emulado em uma situação típica de chão-de-fábrica. É primeiramente apresentado a formulação verbal do problema, descrevendo sequencialmente as ações a serem efetuadas. Em seguida, são apresentadas as representações gráficas através de um diagrama trajeto-passo e de diagrama de estados, os quais servem de apoio à realização da programação SFC-R e da sua fácil compreensão. Por último, são apresentados os resultados da simulação utilizando um CLP, o RCP5 e um circuito eletro-eletrônico que emula a comunicação dos sensores da planta. A Figura 5.9 apresenta um esquema do sistema proposto.

5.2.1 Formulação verbal do problema

De acordo com o esquema apresentado na Figura 5.8, o sistema automático pode ser descrito pelas seguintes considerações:

1) Uma esteira M1 é iniciada deslocando uma peça a ser manipulada até o ponto A.
2) Um sensor S1 de presença indica se a peça chegou ao ponto A.
3) Um pistão pneumático A1 é acionado deslocando a peça até o ponto B.
4) Quando a peça estiver posicionada no ponto B, caso que ocorre quando o pistão atinge o fim de curso S2, e a gaiola de segurança estiver fechada (Sensor S4=1), o robô inicia a movimentação visando a capturá-la.
5) O robô executa a movimentação de *pick and place* até o ponto C e solta a peça.
6) Outro sensor S3 identifica a presença da peça e aciona o pistão A2 para deslocar a peça para outra célula da linha de montagem.

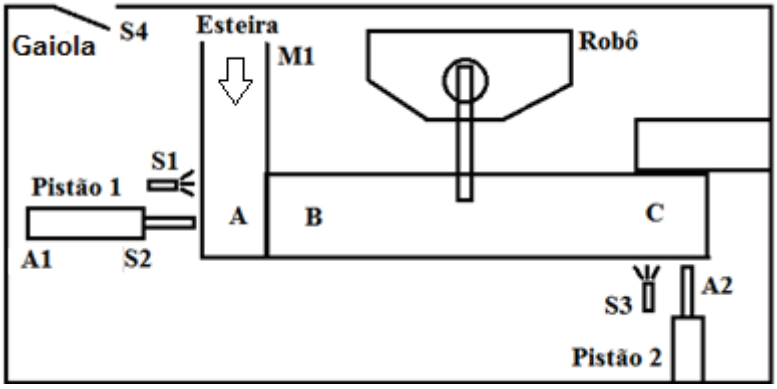

Figura 5.9- Diagrama esquemático representativo de uma planta automática de manufatura controlada por CLP.

Para facilitar o entendimento, a seguir é apresentada a Tabela de Correspondência Lógica dos parâmetros que compõem o exemplo de simulação de uma célula fabril.

Variável	Correspondência Lógica
S1	0: Não há peça no campo de visão do sensor. 1: Há peça no campo de visão do sensor.
S2	0: Pistão não chegou ao fim de curso. 1: Pistão chegou ao fim de curso.
S3	0: Não há peça no campo de visão do sensor. 1: Há peça no campo de visão do sensor.
S4	0: A gaiola está aberta. 1: A gaiola está fechada.
M1	0: Motor da esteira desligado. 1: Motor da esteira ligado.
A1	0: Pistão 1 em repouso. 1: Pistão 1 atuando.
A2	0: Pistão 2 em repouso. 1: Pistão 2 atuando.
R1	0: Robô em repouso. 1: Robô atuando.

5.2.2 Diagrama trajeto-passo.

A formulação verbal do problema (Seção 5.2.1) pode ser representado graficamente pelo diagrama trajeto-passo, o qual serve como apoio para a programação em SFC-R e é de fácil compreensão. A Figura 5.9 apresenta o diagrama trajeto-passo desenvolvido.

LAMEC - UFRGS

Componentes				Tempo (s) 0 1 2 3 4 5 6 7
nº	Designação	Notação	Estado	Passo 1 2 3 4 5 6 7 8 9
1	Motor DC	M1	ligado	
2				
3			desligado	
4	Cilindro pneumático	A1	avançado	
5				
6			recuado	
7	Led indicativo de robô operando	R1	ligado	
8				
9			desligado	
10	Cilíndro pneumático	A2	avançado	
11				
12			recuado	

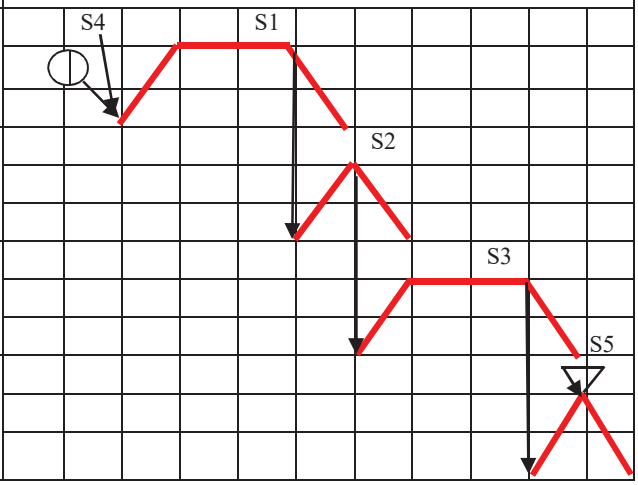

Figura 5.10 – Diagrama Trajeto-Passo.

5.2.3 Programação em SFC-R

Como pode ser visualizado na Figura 5.11, no passo inicial (Step_000) todas as variáveis referentes aos sensores, motores, atuadores e robô são reiniciadas (com valor zero). Em seguida, uma transição que depende do estado do sensor referente à grade (S4) está associada ao passo para ligar a esteira através da variável M1. A transição Trans_001 garante que somente quando sensor S1 é acionado, o programa processa a etapa que desliga o motor da esteira (M1=0) e aciona o pistão pneumático A1. A transição Trans_002, por sua vez, faz a verificação do sensor de final de curso do pistão (S2) e se a grade continua fechada (S4), para então processar a etapa que acende um led (R1) indicando que o robô está em operação, recuando o pistão (A1=0). Continuando o processo, uma transição faz a verificação do estado do robô, a qual está conectada à etapa seguinte que indica um novo movimento (new_cmd) para o Programa Tradutor e qual tipo de movimento (cmd_type=2). Dessa forma, são atribuídos os parâmetros das coordenadas desejadas de *pick* e *place* e é realizada a chamada para a sub-rotina de

movimentação que faz a execução do movimento. Uma transição Trans_005 verifica se o comando foi executado pelo RCP5 e, caso positivo, o programa avança para uma etapa onde o led associado à operação do robô (R1) é desligado e a variável “new_cmd” é reinicializada.

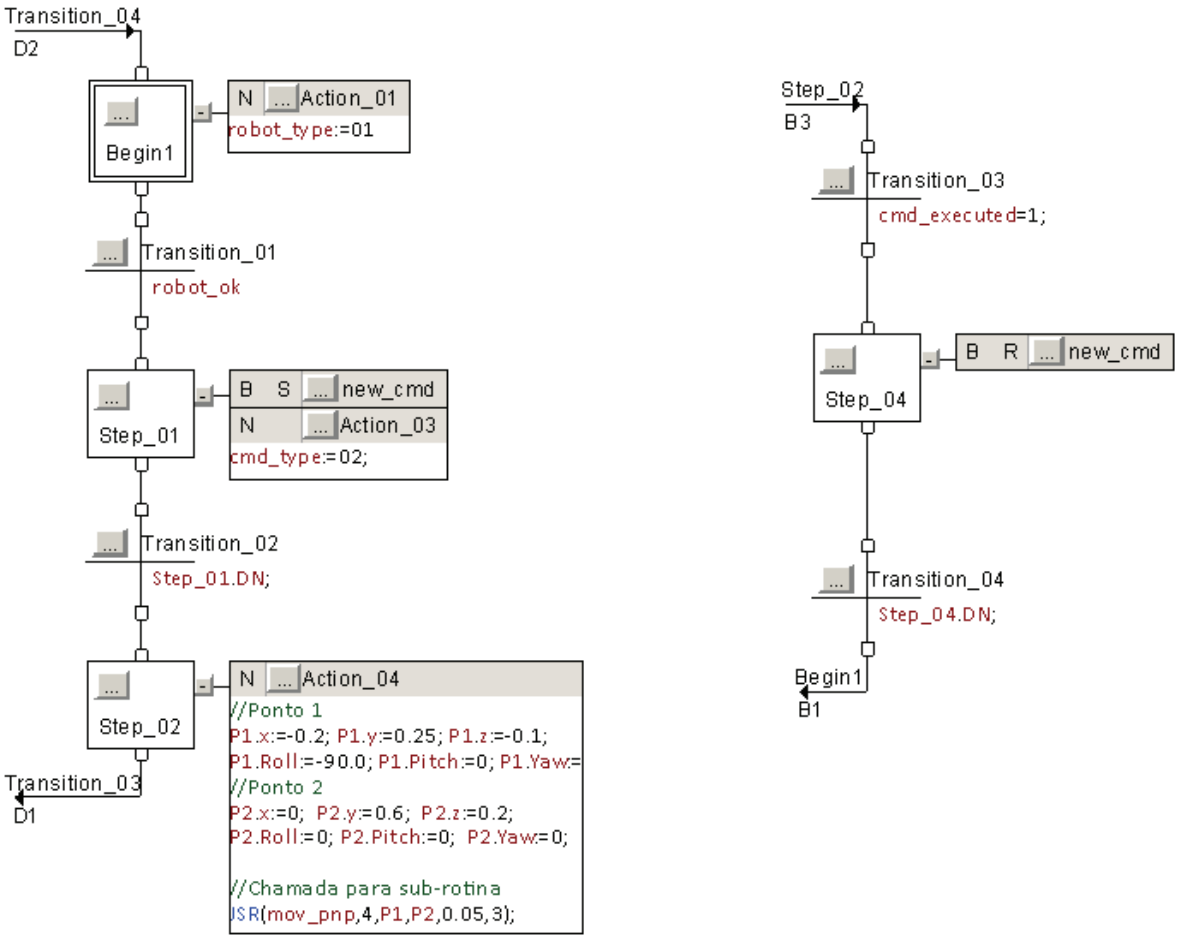

Figura 5.11 – Programação do sistema automático fabril (parcial 1).

A continuação do programa SFC-R é apresentada na Figura 5.12 e conta com uma transição que depende do sensor S3, o qual indica que a peça foi movimentada até o ponto C da Figura 5.9, a qual está associada uma etapa (Step_010) responsável por realizar o avanço do segundo pistão (A2), quando o mesmo atinge o final de curso, e somente quando isto ocorre, o que é garantido pela transição Trans_003. O último passo do programa é responsável por recuar o pistão A2 e retornar ao início do programa SFC-R.

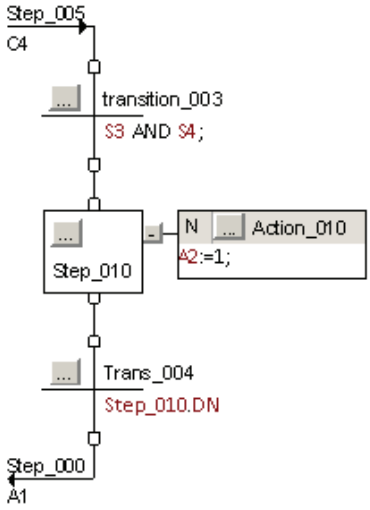

Figura 5.12 – Programação do sistema automático fabril (parcial 2).

5.2.4 Resultados

A Figura 5.13 apresenta a trajetória gerada para cada grau de liberdade do robô.

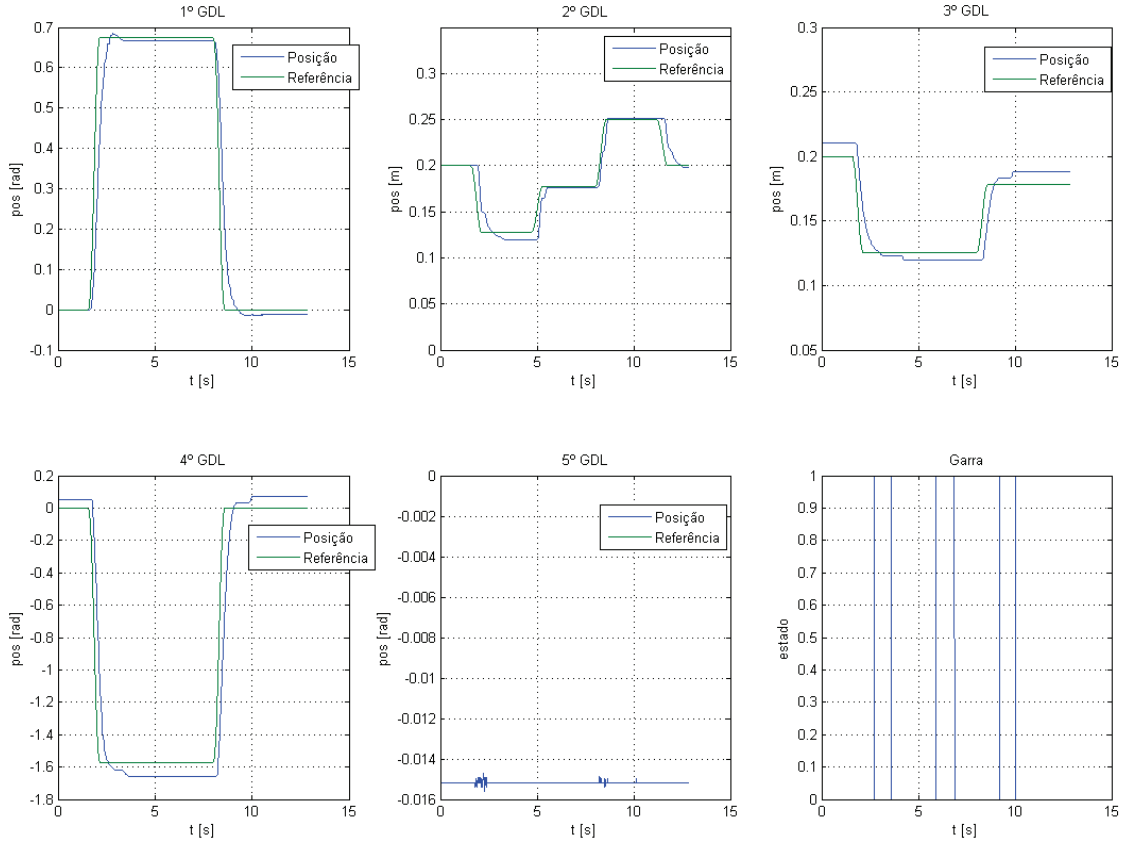

Figura 5.13 – Gráfico de posição com relação à referência de trajetória *pick and place* gerada para o simulação de um sistema automático fabril.

Através das equações 4.7 a 4.9 calculou-se as coordenadas de junta para os pontos P1 e P2 que resultaram em: P1[gdl1 gdl2 gdl3 gdl4 gdl5]=[0.6747rad 0.1273m 0.1257m -1.57rad 0rad] e P2[gd1 gdl2 gdl3 gdl4 gdl5]=[0rad 0.25m 0.1782m 0rad 0rad] e verificou-se através dos gráficos da Figura 5.13 que o Programa Tradutor obteve sucesso em gerar e transmitir a trajetória de *pick and place* para o robô.

A Figura 5.14 apresenta resultados das trajetórias de juntas do PCR5 obtidos para o caso de movimentação integrada com dispositivos em uma planta fabril automática. A operação do servidor OPC é baseada em eventos associados a alterações dos estados de variáveis. O vetor de tempo foi adquirido no formato global (dd/mm/aaaa : hh:mm:ss). Comparando o gráfico da Figura 5.14 com o diagrama trajeto-passo da Figura 5.10 e com a programação SFC-R das figuras 5.11 e 5.12 se pode observar que o processamento do programa seguiu de maneira correta com o procedimento previsto, de forma que cada passo executado habilita o próximo passo (e assim por diante), ou seja, os intertravamentos garantiram a correta operação do sistema. O gráfico dos parâmetros de controle apresentado na Figura 5.14 inicia com a mudança de estado das variáveis “robot_ok” e “S4” para 1, o que indica que o robô está apto para operar e a grade está fechada. O programa aciona esteira, e a variável M1 assume o valor 1. Em seguida, quando a variável S1 muda seu estado para o valor 1, a esteira é desligada (M=0) e o pistão A1 avança (A1=1) até que o sensor S2 mude seu estado para o valor 1 (é possível notar que a variável A1 permanece no nível lógico 1 até que a variável S2 mude o seu estado). Neste instante, a variável new_cmd indica o processamento de um novo comando para o robô. A variável “cmd_executed” muda de estado por meio de ação do Programa Tradutor indicando que o comando foi executado pelo robô. Assim, pode-se observar que após a ação para um novo comando assumir o nível 1, a variável “cmd_executed” é alterada 4 vezes, indicando as movimentações para os 4 pontos do *pick and place*. Quando o sensor S3 muda seu estado para o valor 1, indicando que a peça chegou ao destino final, o pistão A2 é avançado (A2=1) finalizando o processamento do programa.

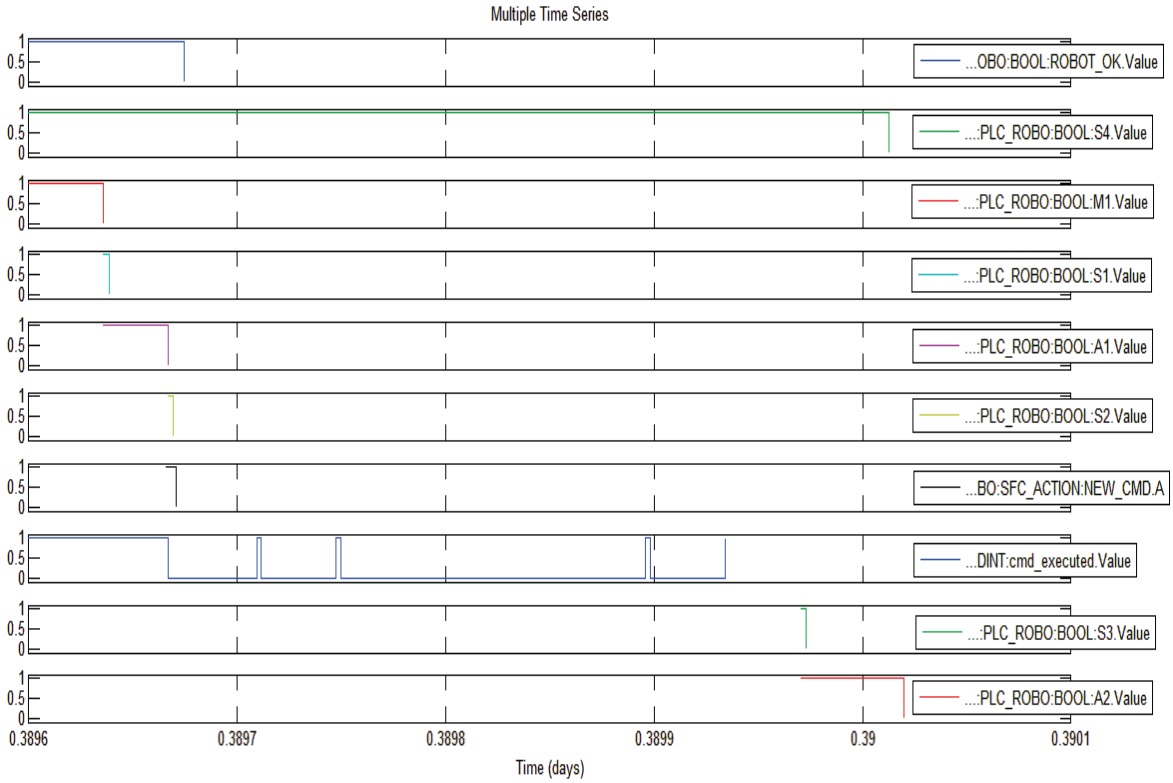

Figura 5.14 – Gráfico dos parâmetros de controle.

6 CONCLUSÕES

Tendo como base os objetivos para o trabalho traçados no Capítulo 1 e os resultados do trabalho desenvolvido, pode-se concluir que a metodologia proposta permite integrar a programação de robôs industriais à linguagem baseada em funções SFC, permitindo que robôs industriais sejam programados conjuntamente com os demais equipamentos do sistema automático. Além disso, o estudo de caso realizado com o RCP5 permite afirmar que a estratégia adotada resultou em um sistema eficaz, que permitiu programar o robô e integrá-lo com outros elementos de uma célula automática. Com isso, é possível concluir que a linguagem de programação SFC-R proposta no presente trabalho tem viabilidade de ser adotada por fabricante de robôs para aplicação comercial.

Com relação às implementações práticas, é também possível concluir, baseado nos resultados, que as sub-rotinas para os três casos de movimentações: movimento linear, de *pick and place* e paletização, foram implantadas com sucesso. Além disso, verificou-se que, tanto as sub-rotinas quanto o arquivo de *tags* podem ser importados pelo usuário e utilizados em qualquer das 5 linguagens descritas na norma IEC61131-3.

Quanto ao estudo de caso utilizado para validar a metodologia proposta, através de dois exemplos práticos, é possível concluir que o programa interpreta corretamente as sub-rotinas em SFC-R, gera novas trajetórias e as atualiza no sistema de controle. Além disso, através da análise dos dados, foi possível verificar que a sincronia entre o programa em SFC-R e o Programa Tradutor garante que os inter-travamentos típicos de um sistema automatizado sejam observados.

Como trabalhos futuros, pode-se mencionar a integração com o sistema de visão desenvolvido por Oliveira, 2015 e a possibilidade de adição de novos robôs ao Programa Tradutor e respectivas sub-rotinas. Além disso, outra possibilidade de trabalho futuro consiste na criação de um software de programação SFC próprio, para eliminar a dependência do RSLogix 5000, e de um supervisório para aquisição de coordenadas, seleção e visualização de parâmetros, bem como geração de relatórios de variáveis e de gráficos, tornando a programação mais amigável para o usuário. Outro trabalho importante é o relacionado ao estudo do desenvolvimento de ações envolvendo a programação de robôs industriais através de uma proposta de extensão da norma IEC 61131-3.

REFERÊNCIAS BIBLIOGRÁFICAS

Bollmann, A. **Fundamentos da Automação Industrial Pneutrônica- Projetos de comandos binários eletropneumáticos**.ABHP,1996.

Bonfe, M. Vignali M. and Fiorini, M. **"PLC-based control of a robot manipulator with closed kinematic chain,"** Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, Kobe, 2009,

Chaves J. H., **Linguagem de programação de robôs**, Mecatrônica Atual. Disponível em: < http://www.mecatronicaatual.com.br/secoes/leitura/418>. Acesso em 2016.

Chung, C. A. **"A cost-effective approach for the development of an integrated PC-PLC-robot system for industrial engineering education,"** in IEEE Transactions on Education, vol. 41, no. 4, pp. 306-310, Nov 1998.

Dreamstime, **Cardboard boxes on pallet** Disponível em: <https://www.dreamstime.com/stock-image-cardboard-boxes-pallet-wooden-isolated-white-background-warehouse-shipping-cargo-delivery-concept-image38802701>. Acesso em 2016.

El Din, A. S. **"High performance PLC controlled stepper motor in robot manipulator,"** Industrial Electronics, 1996. ISIE '96., Proceedings of the IEEE International Symposium on, Warsaw, 1996, pp. 974-978 vol.2.

Fu, K. S.; Gonzalez, R. C;.; Lee, C. S. G. **"Robotics: Control, Sensing, Vision and Intelligence"**. McGraw-Hill, New York, 1987.

Gong, W. **"Automatic robot path generation for manufazturing on sculptured surfaces".** Master Thesis, Windsor University, 1998.

Henriques, R. V. B. **Programação e Simulação de Robôs.** Em: Romano, V. F., Henriques, R. V. B., Dutra, M. S., Rosario, J. M., Pereira, C. E., Lages, W. F., Costa, A. H. R.. (org.). Robótica industrial: Aplicação na Indústria de Manufatura e de Processos.. 1ed.São Paulo: Edgard Blücher, 2002

Ivanescu, N. A. Borangiu, T. Brotac S. and Dogar, A. **"Implementation of sequential function charts with microcontrollers,"** Control & Automation, 2007. MED '07. Mediterranean Conference on, Athens, 2007, pp. 1-6.

Kumaar A. A. N. and TSB, S. **"Mobile Robot Programming by Demonstration,"** 2011 Fourth International Conference on Emerging Trends in Engineering & Technology, Port Louis, 2011, pp. 206-209

Matlab. **Matlab Product Description**. Dísponível em: <http://www.mathworks.com/help/matlab/learn_matlab/product-description.html>. Acesso em 2016.

Missiaggia, L. **Planejamento Otimizado de Trajetória para um Robô Cilíndrico Acionado Pneumaticamente**. Disseração de Mestrado, Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Engenharia Mecânica,2014.

Neto, P. Pires J. N. and Moreira, A. P. **"CAD-based off-line robot programming,"** 2010 IEEE Conference on Robotics, Automation and Mechatronics, Singapore, 2010, pp. 516-521.

Oliveira, B. V. **Sistema de Visão Computacional Aplicado a um Robô Cilíndrico Acionado Pneumaticamente**. Disseração de Mestrado, Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Engenharia Mecânica,2015.

OPC Foundation, **What is OPC**. Disponpivel em: <https://opcfoundation.org/about/what-is-opc/>. Acesso em 2016.

Pereira, A. Lima, C. and Martins, J. F. **"The use of IEC 61131-3 to enhance PLC control and Matlab/Simulink process simulations,"** 2011 IEEE International Symposium on Industrial Electronics, Gdansk, 2011, pp. 1243-1247.

PLCOpen. **TC1-Standards**. Disponível em:<http://www.plcopen.org/pages/tc1_standards/>. Acesso em 2016.

Rijo, M.G.Q**. Desenvolvimento da Base e Controle do Grau de Liberdade de um Robô Cilíndrico com Acionamento Pneumático**. Disseração de Mestrado, Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Engenharia Mecânica,2013.

Sarmanho Jr, C. A. **Desenvolvimento de um Robô Pneumático de 5 Graus de Liberdade com Controlador Não Linear com Compensação de Atrito**. Tese de Doutorado.niversidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Engenharia Mecânica,2014.

Secil, S. Turgut, K. Parlaktuna O. and Ozkan, M. **"Simple programming scheme for industrial robot manipulators: A case study for feasibility proof,"** Innovations in Intelligent SysTems and Applications (INISTA), 2015 International Symposium on, Madrid, 2015, pp. 1-7.

Silveira, P. R.; Santos, W. E. **Automação e Controle Discreto**. São Paulo: Érica, 1998.

Skulavik, T., Michal, K. and Alena, K. **"Fuzzy control of robotic arm implemented in PLC,"** Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on, Tihany, 2013, pp. 45-49.

Spong, M. W.; Vidyasagar, M. **Robot Dynamics and Control**. John Wiley & Sons, Inc., 1989.

Subbaraman, S. Patil M. M. and Nilkund, P. S. **"Novel integrated development environment for implementing PLC on FPGA by converting ladder diagram to synthesizable VHDL code,"** Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on, Singapore, 2010, pp. 1791-1795.

Thomas, U. Hirzinger, G. Rumpe, B. Schulze C. and Wortmann, A. **"A new skill based robot programming language using UML/P Statecharts,"** Robotics and Automation (ICRA), 2013 IEEE International Conference on, Karlsruhe, 2013, pp. 461-466.

Tomas, S. Michal K. and Alena, K. **"Fuzzy control of robotic arm implemented in PLC,"** Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on, Tihany, 2013, pp. 45-49.

Wu, H. Deng, H. Yang, C. Guan, Y. Zhang H. and Li, H. **"A robotic off-line programming system based on SolidWorks,"** 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), Zhuhai, 2015, pp. 1711-1716.

APÊNDICE A – LINGUAGENS DA NORMA IEC 61131-3

A.1 Diagrama de Blocos Funcionais (FBD)

A linguagem de diagrama de blocos funcionais é uma linguagem gráfica que se assemelha aos diagramas utilizados para descrever circuitos e esquemas com portas lógicas na eletrônica. Este tipo de linguagem é recomendada para programadores principiantes, pois é uma linguagem simples que segue um caminho lógico visualmente simples de ser acompanhado. A Figura A.1 apresenta um exemplo de aplicação da linguagem.

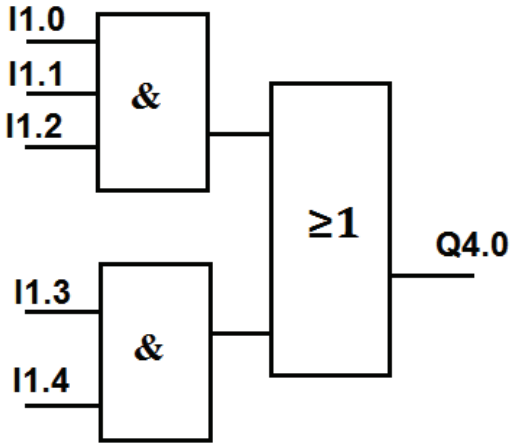

Figura A.1 – Exemplo de Fluxograma Lógico.

Fonte: Adaptado de Bollmann, 1997.

Uma das desvantagens deste tipo de programação é que os blocos ocupam bastante espaço de trabalho, principalmente quando o programa é complexo e utiliza diversas funcionalidades tais como memórias do tipo RS. Outra desvantagem é que o programador precisa ter um conhecimento do seu programa e como ele irá fluir antes mesmo de programar, pois pode se tornar difícil fazer correções posteriores.

A.2 Texto Estruturado (ST)

A linguagem em texto estruturado é a que mais se assemelha às linguagens convencionais de programação como o C e Pascal. É uma linguagem de alto nível e permite programação de funções e blocos que ficariam de difícil solução com outras linguagens. Ela diminui o tempo de execução e torna mais compacto os programas de maior complexidade, é

possível também se utilizar de funções de decisão, controle de loops e funções de trigonometria e aritmética. Um exemplo deste tipo de linguagem pode ser visualizado na Figura A.2.

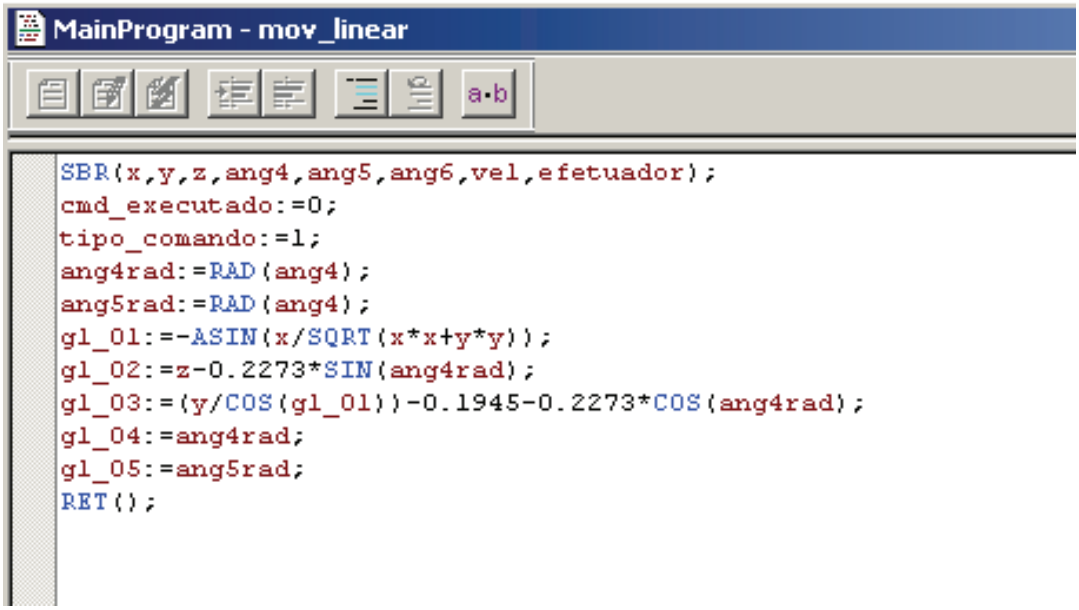

Figura A.2 – Programa em Texto Estruturado.