UNIVERSIDADE FEDERAL DE MINAS GERAIS
Curso de Graduação em Engenharia de Controle e Automação

Trabalho de Graduação

"Desenvolvimento de uma interface unificada de programação de robôs utilizando código G"

Autor: Francisco Rodrigues Alves de Oliveira
Orientador: Eduardo Jose Lima II
Supervisor: Frederico Allevato Ramalho Filho

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Curso de Graduação em Engenharia de Controle e Automação

Francisco Rodrigues Alves de Oliveira

DESENVOLVIMENTO DE UMA INTERFACE UNIFICADA DE PROGRAMAÇÃO DE ROBÔS UTILIZANDO CÓDIGO G

Belo Horizonte
Escola de Engenharia da UFMG
2008

AGRADECIMENTOS

Agradeço a todos que contribuíram para a elaboração deste trabalho com palpites e sugestões.

A todos que contribuíram indiretamente dando o suporte necessário para que esse trabalho fosse concluído.

Ao professor Eduardo a ao Fred que deram apoio e me orientaram até quando eu sumia e não dava sinal de vida.

Sumário

Lista de Figuras:

Lista de Tabelas:

Lista de Siglas:

CNC.....(Comando Numérico Computadorizado);

CN.....(Controle Numérico)

LRSS.....(Laboratório de Robótica, Soldagem e Simulação)

DEMEC.....(Departamento de Engenharia Mecânica)

UFMG.....(Universidade Federal de Minas Gerais)

POO.....(Programação Orientada a Objetos)

CAD/CAM.....(Computer-Aided Manufacturing)

PID.....(Proporcional, Integral e Derivativo)

FPGA.....(Field-Programmable Gate Array)

LED.....(Light-Emitting Diodes)

Resumo

O projeto desenvolvido consiste em uma interface de programação de robôs utilizando Código G. O Código G foi escolhido por ser muito utilizado na programação de máquinas CNC (Comando Numérico Computadorizado) e por já haver uma vasta documentação abordando este tema. Outra vantagem do Código G é que ele é o padrão dos programas CAD/CAM (Computer-Aided Manufacturing). Contudo, robôs industriais possuem linguagens proprietárias de cada fabricante, dificultando a geração automática de programas por sistemas CAM (Computer-Aided Manufacturing).

O programa primeiramente analisa o código à procura de erros. Assim que todos os erros são corrigidos, o código é interpretado, sendo traduzindo em uma seqüência de pontos de uma trajetória no espaço cartesiano. A trajetória gerada é então combinada com o modelo do robô, gerando um novo código G no espaço de juntas. Esta trajetória no espaço de juntas pode ser enviada também para um controlador PID (Proporcional, Integral e Derivativo) responsável pelo controle de posição, velocidade e aceleração do robô. O controlador é responsável por converter esses pontos em pulsos elétricos que comandarão os motores do robô.

Ao fim do trabalho foi desenvolvido um programa capaz de traduzir uma trajetória no espaço cartesiano para o espaço de juntas.

Foram realizados testes simulados utilizando os ângulos calculados pelo programa e o software de desenho AutoCad, e em todos os casos simulados o robô seguiu a trajetória desejada.

Abstract

The developed project is a robots programming interface using G Code. The G Code was chosen because it is frequently used in the CNC machines programming and there is a lot of documentation about this subject. Another G Code advantage is that it is the CAD/CAM programs standard. However, each manufacturer is the owner of the industrial robots language. It makes difficult the automatic generation of CAM systems programs.

At first, the program analyzes the code searching errors. The code is interpreted after all corrected errors to get the trajectory points sequence. The generated trajectory is combined with the robot model to generate a new G code in the joints space. The joints space trajectory can also be sent to the PID controller responsible for the robot position, speed and acceleration control. The controller is responsible for converting these points into electric pulses that will command the robot engines.

In the end of this project a program able to translate a trajectory in the cartesian space for the joints space was developed.

Simulated tests had been carried by using the program calculated angles in the Auto Cad Software environment. The robot has followed the desired trajectory in all of the simulations.

1. Introdução

Este projeto final de curso será desenvolvido no Laboratório de Robótica, Soldagem e Simulação (LRSS) do Departamento de Engenharia Mecânica (DEMEC) da Universidade Federal de Minas Gerais (UFMG). Neste laboratório se encontra uma sala de aula para o ensino de matérias relacionadas à robótica, também se encontram diversas salas de professores e laboratórios para construção e testes de robôs.

A aplicação há ser desenvolvida será capaz de interpretar um programa que descreve uma trajetória, escrito em código G, traduzindo este código em pontos de uma trajetória a ser seguida por um robô. Esta trajetória será convertida em pontos no espaço de juntas do robô e, a partir destes novos pontos, será escrito um novo código G no espaço de juntas. Este novo código será utilizado como entrada para programas de controle de máquinas de controle numérico (CN) disponíveis gratuitamente. Esta trajetória no espaço de juntas pode ser enviada diretamente, ponto a ponto, também para um controlador PID (proporcional, integral e derivativo) responsável pelo controle de posição, velocidade e aceleração do robô. O controlador é responsável por converter esses pontos em pulsos elétricos que comandarão os motores do robô.

Para o desenvolvimento desta aplicação, serão empregados os conhecimentos da cinemática direta e inversa. Na cinemática direta o problema é determinar a posição e a orientação da ferramenta em relação ao sistema de coordenadas base, conhecendo-se os ângulos de cada junta do robô. Para isso, é definido um sistema de coordenadas para cada elo e obtidas as relações entre esses sistemas de coordenadas. Na cinemática inversa, o problema é converter

um ponto no espaço cartesiano para o espaço de juntas do robô, ou seja, calcular a posição de cada atuador do robô para que o manipulador atinja a posição definida.

Também serão empregados conhecimentos relacionados à geometria analítica para solucionar os problemas de interpolação entre os pontos.

O aplicativo será desenvolvido aplicando a metodologia de modelagem orientada a objetos, pois esta metodologia agiliza o processo de desenvolvimento do programa e faz com que ele seja de fácil manutenção.

Na elaboração deste projeto será utilizado o ambiente de programação Borland Delphi 6.0, pois este é compatível com a modelagem orientada a objetos e facilita muito a criação de interfaces gráficas por já possuir muitas ferramentas pré-desenvolvidas.

Esta monografia será dividida em capítulos intitulados da seguinte forma: Introdução: Descreve sucintamente o assunto que será abordado, o local aonde será desenvolvido o trabalho, as ferramentas e a metodologia que serão utilizadas para no desenvolvimento deste projeto; Revisão Bibliográfica: Revisa conceitos que serão utilizados; Materiais e Métodos: Descreve as ferramentas e materiais que serão utilizados; Desenvolvimento: Mostra como o projeto será desenvolvido, explicando a metodologia que será empregada e as funcionalidades do aplicativo; Resultados e Testes: Apresentará os resultados obtidos e os testes realizados; Conclusões: Descreverá as conclusões aprendidas no decorrer deste trabalho e as dificuldades encontradas durante o desenvolvimento; Bibliografia: Apresentará as obras consultadas no desenvolvimento deste trabalho.

O LRSS possui e está desenvolvendo robôs para desempenhar diversas funções industriais e de campo. O objetivo final deste trabalho será desenvolver uma interface que facilite e unifique a tarefa de programação das trajetórias destes robôs.

2. Revisão bibliográfica

2.1. Cinemática

Pela definição da (ISO, 1992) um robô industrial é “uma máquina manipuladora com vários graus de liberdade controlada automaticamente, reprogramável, multifuncional, que pode ter base fixa ou móvel para utilização em aplicações de automação industrial". Os campos de aplicação mais comuns da robótica industrial são: fundição, pintura, soldagem, montagem, movimentação de cargas e inspeção de produtos.

Um robô industrial é composto basicamente por um manipulador e um controlador. O manipulador é a parte mecânica que realiza o posicionamento da ferramenta através da movimentação de seus elos, que são conectados entre si através de juntas (que podem ser rotativas ou prismáticas). As juntas são acionadas por meio de atuadores, que convertem a energia elétrica em movimento e possuem sensores que fornecem ao controlador informações sobre posição e velocidade (Lima II, 2005).

O problema fundamental no controle de robôs industriais consiste na geração de trajetória no espaço de juntas: Dada uma trajetória definida no espaço cartesiano, obter a trajetória no espaço de juntas do robô, ou seja, os movimentos necessários a cada atuador para gerar uma trajetória da ferramenta. Esse problema é conhecido como o problema da cinemática inversa (Szkodny, 1995).

A cinemática trata do estudo dos movimentos sem considerar as causas que lhes dão origem (Craig, 1989). Na cinemática direta, o problema é determinar a posição e a orientação da ferramenta em relação ao sistema de coordenadas base, sabendo os ângulos de cada junta do robô. Para isso, é definido um sistema de coordenadas para cada elo e obtidas as relações entre esses sistemas de coordenadas.

O posicionamento do robô pode ser descrito de três maneiras diferentes:

- A primeira forma é a descrição no espaço cartesiano que utiliza um sistema de coordenadas ($P(t) = \{ X, Y, Z, A, B, C \}$), em que X, Y e Z determinam a posição no espaço e A, B e C determinam a orientação,em que A, B, e C representam os ângulos de rotação em torno de X, Y e Z respectivamente.

- A segunda maneira é no espaço de juntas, em que a posição é descrita através de um conjunto de variáveis ($\theta(t) = \{ \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6 \}$) de deslocamento ou ângulos das articulações do robô.

- A terceira maneira de descrever a posição do robô é no espaço dos atuadores, em que a posição é descrita pelas seguintes variáveis ($A(t) = \{ A_1, A_2, A_3, A_4, A_5, A_6 \}$) que representam a posição de cada atuador do robô.

Define-se então, como cinemática direta, a relação entre o espaço das juntas e o espaço cartesiano: $\theta (t) \rightarrow P(t)$. Já a cinemática inversa determina a relação entre o espaço cartesiano e o espaço das juntas: $P (t) \rightarrow \theta (t)$. A cinemática dos atuadores, por sua vez, determina a

relação entre o espaço das juntas e dos atuadores e vice-versa (Lima II, 2005). A Figura 2.1 mostra essas relações.

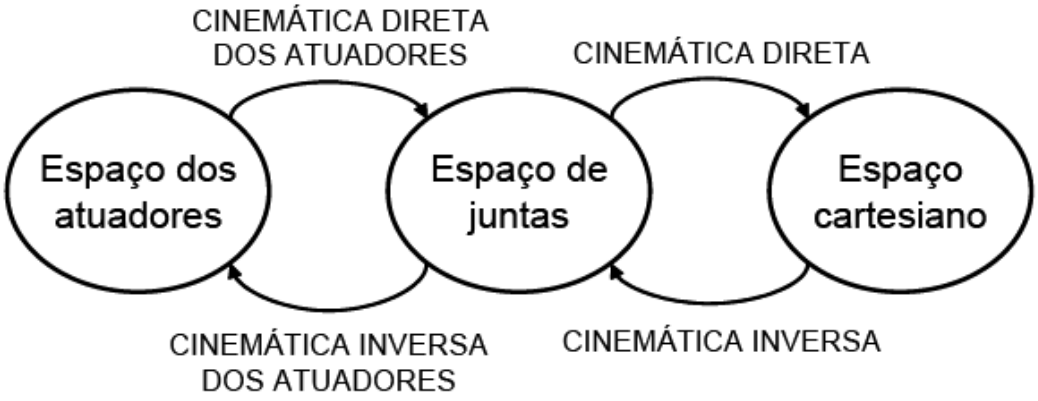

Figura 2-1 - Espaços utilizados para definir o posicionamento de um robô adaptado de (Craig, 1989).

2.1.1. Descrição espacial e transformada espaciais

Desde que um sistema de coordenadas cartesianas {A} esteja definido no espaço, é possível descrever a posição de um ponto por um vetor 3 x 1.

$$ {}^{A}P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \qquad 2.1 $$

Para descrever completamente um objeto no espaço, torna-se necessário também definir sua orientação. Para isso, define-se um sistema de coordenadas {B} preso ao objeto. Assim, uma descrição da orientação do sistema de coordenadas {B} em relação ao {A} define também a orientação do objeto em relação a {A}. Uma maneira de se obter essa orientação é definir os

vetores unitários dos três eixos principais de {B} ($^{\wedge}X_B$, $^{\wedge}Y_B$ e $^{\wedge}Z_B$) em relação a {A}, obtendo-se $^{A\wedge}X_B$, $^{A\wedge}Y_B$ e $^{A\wedge}Z_B$. Esses três vetores definem a matriz rotação 3 x 3 (Craig, 1989).

$$ {}^{A}_{B}R = \begin{bmatrix} {}^{A}\hat{X}_B & {}^{A}\hat{Y}_B & {}^{A}\hat{Z}_B \end{bmatrix} \qquad 2.2 $$

Para completar a descrição de um sistema de coordenadas {B} em relação a um sistema de coordenadas {A}, é necessário também conhecer a posição do ponto de origem do sistema {B} em relação ao sistema {A}. A esse deslocamento damos o nome de $^{A}P_{BORG}$.

Conhecendo $^{A}_{B}R$ e $^{A}P_{BORG}$ é possível descrever, em relação ao sistema {A}, a posição e a orientação de um objeto conhecido em {B}. Essa transformação pode ser feita definindo-se uma matriz de transformada homogênea 4 x 4 ($^{A}_{B}T$) como indicado na equação (2.3).

$$ {}^{A}_{B}T = \begin{bmatrix} {}^{A}_{B}R & {}^{A}P_{BORG} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \qquad 2.3 $$

A descrição ^{B}P de um ponto P em relação ao sistema {B} pode ser transformada em uma descrição ^{A}P em relação ao sistema {A}, como indicado na equação (2.4). Esta transformação está ilustrada na Figura (2.2).

$$ {}^{A}P = {}^{A}_{B}T \, . \, {}^{B}P \qquad 2.4 $$

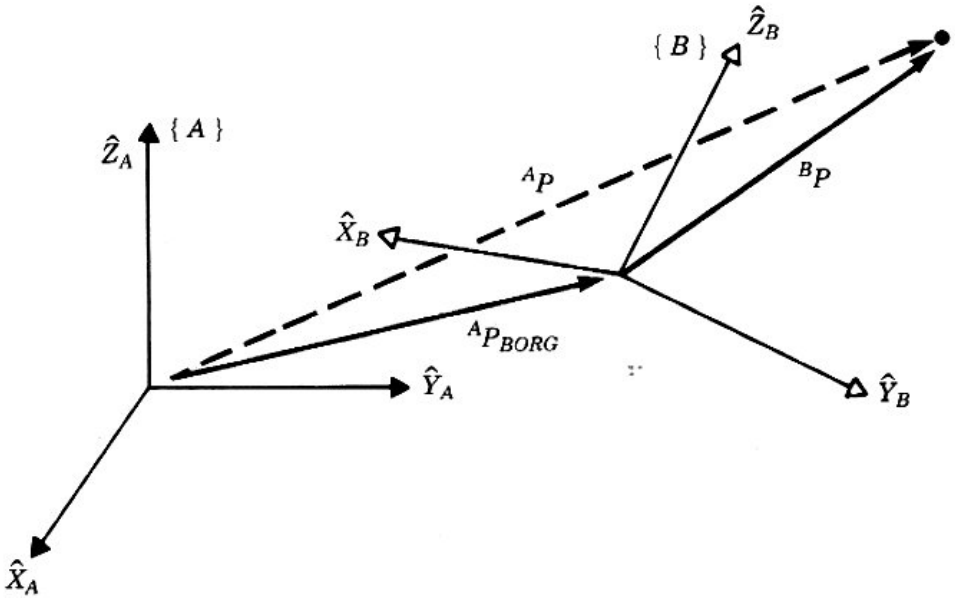

Figura 2-2 - Descrição do ponto ^BP em relação ao sistema {A} (Craig, 1989).

2.1.2. Modelagem cinemática

Para a modelagem cinemática dos manipuladores robóticos, é necessário anexar um sistema de coordenadas em cada uma das juntas do robô. O modelo é então obtido definindo-se a matriz de transformada homogênea de cada junta em relação à junta anterior.

Seguindo esse procedimento, define-se um sistema de coordenadas {0} como sendo o sistema de coordenada base (fixo). Definem-se também os sistemas {i} para cada articulação do manipulador, onde i é o número da junta. Finalmente, o sistema {T} é anexado à ponta da ferramenta.

Os sistemas de coordenadas serão anexados a cada uma das juntas seguindo a notação de Denavit-Hatenberg (D-H) (Craig, 1989). Nessa convenção, a transformação entre os sistemas de coordenadas de dois elos adjacentes (sistemas i e i - 1) é definida por apenas 4 parâmetros :

- a_{i-1} = distância de $^{\wedge}Z_{i-1}$ a $^{\wedge}Z_i$ medida ao longo de $^{\wedge}X_{i-1}$;
- α_{i-1} = ângulo entre $^{\wedge}Z_{i-1}$ e $^{\wedge}Z_i$ rotacionado em torno de $^{\wedge}X_{i-1}$;
- d_i = distância entre $^{\wedge}X_{i-1}$ e $^{\wedge}X_i$ medida ao longo de $^{\wedge}Z_i$;
- θ_i = ângulo entre $^{\wedge}X_{i-1}$ e $^{\wedge}X_i$ rotacionado em torno de $^{\wedge}Z_i$.

Definindo-se os sistemas de coordenadas desta maneira, os cálculos serão simplificados.

A Figura 2.3 ilustra os 4 parâmetros de Denavit-Hartenberg.

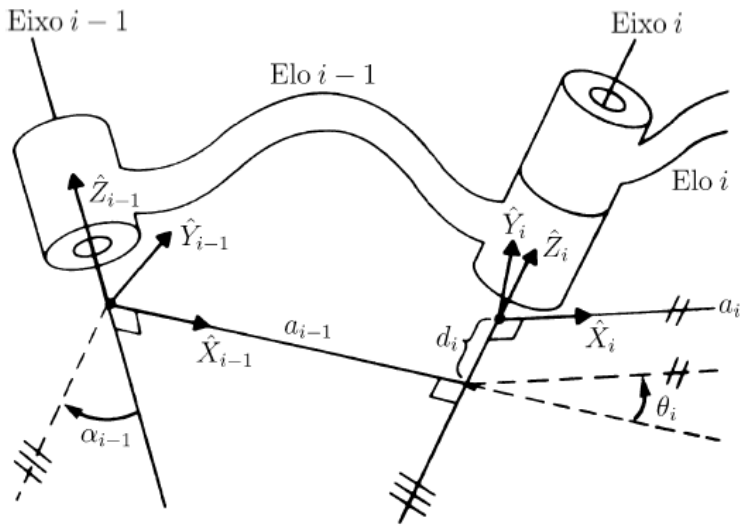

Figura 2-3 - Exemplo de uma junta rotativa e os sistemas de coordenadas anexados aos elos (Craig, 1989).

A matriz de transformada homogênea que descreve o sistema {i} em relação ao sistema {i-1} está descrita na equação (2.5).

$$ {}^{i-1}_{i}T = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i} & 0 & a_{i-1} \\ S_{\theta_i}C_{\alpha_{i-1}} & C_{\theta_i}C_{\alpha_{i-1}} & -S_{\alpha_{i-1}} & -S_{\alpha_{i-1}}d_i \\ S_{\theta_i}S_{\alpha_{i-1}} & C_{\theta_i}S_{\alpha_{i-1}} & C_{\alpha_{i-1}} & C_{\alpha_{i-1}}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 2.5 $$

Onde $C\theta_i = \cos(\theta_i)$ e $S\theta_i = \text{sen}(\theta_i)$ e assim por diante.

Para um robô de N juntas, a transformação entre o sistema de coordenadas da base e o sistema de coordenadas da junta N é dada por:

$$ {}^{0}_{N}T = {}^{0}_{1}T \,.\, {}^{1}_{2}T \,...\, {}^{N-1}_{N}T \qquad 2.6 $$

Normalmente essa matriz ${}^{0}_{N}T$ é uma constante, deste que as características geométricas do robô e da ferramenta permaneçam constantes.

2.2. O código G

O código G é uma maneira simples para descrever trajetórias utilizando-se linhas de código texto (CNC Information, 2008). Ele foi idealizado inicialmente para a programação de máquinas. Porém o código G não é uma linguagem totalmente padronizada, pois cada fabricante de maquinas CN segue a sua própria padronização.

Em linhas gerais, existe uma certa normatização para a sintaxe dos principais comandos, que é seguida pela maioria dos fabricantes, aplicando-se um mesmo padrão para programação das maquinas ferramentas. Entretanto, as rotinas e os ciclos automáticos podem variar de

fabricante para fabricante e com o modelo da máquina. Esses ciclos (conhecidos como "canned cycles") são utilizados principalmente na programação manual, para diminuir o trabalho do programador. No caso da programação auxiliada por computador, a tendência é de que o computador gere um código que descreva todos os passos, sem incorporar nenhuma etapa dentro de uma função especial (Ramalho, 2003).

A Tabela 2.1 apresenta as instruções principais do código G utilizado na programação de máquinas CN MACH 9 (Romi, S.A., 1996), onde X representa o afastamento radial em relação ao zero da peça, Y representa o afastamento longitudinal, F o avanço (Velocidade), os parâmetros I e J representam as coordenadas centro do arco.

Tabela 2-1 - Descrição dos principais comandos do código G em tornos CNC (Romi, S.A., 1996)

Função	Variáveis Dependentes	Descrição
G00	X e Y	Movimentação com maior avanço possível para coordenada (X,Y).
G01	X, Y, F	Interpolação linear para movimento de corte.
G02	X, Y, I, K	Interpolação circular no sentido horário.
G03	X, Y, I,	Interpolação circular no sentido anti-horário.
T0101	Número	Descreve a ferramenta a ser utilizada.
G99	-	Indica início do programa.
M30	-	Indica final do programa.
G33	Y e K	Abertura de rosca paralela a Y com o passo K em um único passe.
G37	X, Y, K, D	Ciclo automático para rosqueamento, para rosca em qualquer posição, com múltiplos passes.

G41	-	Ativa a compensação do raio da ferramenta (esquerda).
G42	-	Ativa a compensação do raio da ferramenta (direita).
G66	X, Y, I, K, W, P	Ciclo automático para desbaste paralelo ao eixo Y, com sobremetal de dimensões I e K e segundo o perfil contido no subprograma P.
G67	X, Y, I, K, W, P	Ciclo automático para desbaste paralelo ao eixo X, com sobremetal de dimensões I e K e segundo o perfil contido no subprograma P.
G68	X, Y, I, K, W, E, P	Ciclo automático para desbaste paralelo ao perfil final, com sobremetal de dimensões I e K e segundo o perfil contido no subprograma P.

Este trabalho utilizará as variáveis as X,Y e Z para definir um ponto no espaço e as variáveis A, B e C definem a orientação espacial da ferramenta.

2.3. Programação orientada a objetos

Programação Orientada a Objetos (POO) é uma metodologia de análise, projeto e programação de sistemas de software baseada na composição e interação entre diversas unidades de software chamadas de objetos (H. M. Deitel, 2004).

Essa metodologia tem como objetivo descrever e desenvolver um sistema de software a partir de um conjunto de objetos e suas interações. O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos.

Na programação orientada a objetos, define-se um conjunto de classes que descrevem os objetos presentes no sistema de software. A classe determina os comportamentos e estados possíveis de seus objetos, assim como o relacionamento com outros objetos.

Essa metodologia facilita muito o desenvolvimento de softwares, pois os objetos são facilmente reutilizáveis. É também uma arquitetura que economiza linhas de código, facilitando a manutenção dos programas.

Os principais conceitos utilizados pela modelagem orientada a objetos serão descritos a seguir (H. M. Deitel, 2004):

- **Classe:** Define o comportamento dos objetos através de métodos e, através de atributos, quais são os estados que eles são capazes de manter.

- **Objetos:** É uma instância de uma classe. Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.

- **Atributo**: Correspondem às características de um objeto. Basicamente, a estrutura de dados que vai representar a classe.

- **Método**: Descreve o comportamento dos objetos em relação a um determinado evento. A ação só ocorre quando o método é invocado através da instância de uma classe.

- **Sobrecarga:** é a utilização do mesmo nome para símbolos ou métodos com operações ou funcionalidades distintas.

- **Herança**: É o mecanismo pelo qual uma "classe-filha" herda os métodos e atributos de uma "classe-pai".

- **Associação:** é o mecanismo pelo qual um objeto utiliza os recursos de outro. Pode tratar-se de uma associação simples "usa um" ou de um acoplamento "parte de".

  Encapsulamento: Define a separação de aspectos internos e externos de um objeto. Este mecanismo é utilizado amplamente para impedir o acesso direto a um atributo do objeto, disponibilizando métodos públicos que alterem ou retornem os valores destes atributos.

- **Abstração**: É a habilidade de modelar um objeto concentrando-se nos aspectos essenciais para uma dada aplicação.

- **Polimorfismo:** É o princípio pelo qual duas ou mais "classes-filhas" de uma mesma "classe-pai" podem invocar métodos que têm o mesmo nome, lista de parâmetros e retorno, mas comportamentos diferentes, especializados para cada "classe-filha".

O projeto desenvolvido utiliza a metodologia de modelagem orientada a objeto. A razão para se utilizar esta metodologia foram as vantagens citadas anteriormente.

3. Materiais e métodos

3.1. Ambiente de programação Borland Delphi

O Object Pascal é uma linguagem de programação orientada a objetos de alto nível. Desde a década de noventa, é uma das linguagens de programação mais populares, sendo muito usada no meio acadêmico e por empresas desenvolvedoras de programas. A sintaxe utilizada é muito parecida com a linguagem Pascal, por se tratar de uma evolução desta, o que facilita sua aprendizagem. É uma linguagem muito utilizada e com uma vasta literatura acessível, que disponibiliza uma poderosa gama de ferramentas matemáticas necessárias para o desenvolvimento deste projeto.

A escolha do ambiente Borland Delphi se justifica por ser um ambiente simples que oferece uma grande variedade de funcionalidades pré-desenvolvidas, facilitando e agilizando o desenvolvimento de interfaces gráficas amigáveis. Outra vantagem do Delphi é que ele pode gerar aplicativos com baixa exigência técnica, sendo apropriado para computadores de pequeno porte.

Os requisitos necessários para a criação de um programa no Borland Delphi são:

- Microsoft Windows 98 ou superior;
- 32 megabytes de memória RAM;
- 1Gb de memória livre no disco rígido;

- Placa de vídeo VGA ou superior;

- Microprocessador da família 80386.

3.2. O código G

Robôs antropomórficos normalmente não utilizam código G para programação de seus movimentos. Esta linguagem é mais usada para a programação de máquinas CN, possibilitando descrever uma trajetória de forma simples. Ela foi escolhida para esta aplicação por ser uma linguagem muito utilizada pela indústria metal-mecânica e por ser o padrão dos programas CAD/CAM (computer-aided manufacturing). Além disso, trata-se de uma linguagem com um grande número de usuários e com uma vasta literatura disponível.

O código G a ser interpretado pelo programa a ser desenvolvido, é uma simplificação das instruções contidas na Tabela 2-1, pois alguns comandos descritos na Tabela 2-1 como os comandos G66, G67 e G68 são específicos para uso em tornos. Outros, como os comandos G41 e G42 que fazem menção à ferramenta utilizada, também não são necessários à aplicação a ser desenvolvida.

O interpretador estará apto a reconhecer funções que descrevem uma trajetória como G00, G01, G02 e G03.

3.3. O controlador (PID) de juntas

O controlador de juntas recebe os set-points de posição e comanda os motores de cada junta para que o robô realize a trajetória.

Outra função é controlar a velocidade de movimentação do robô, condicionando a curva de velocidade. Por exemplo, uma trajetória que descreva o deslocamento do robô da posição P1 para P2 e em seguida para P3 (Figura 3-1A), necessitaria idealmente de uma curva de velocidade como ilustrado pela Figura 3-1B.

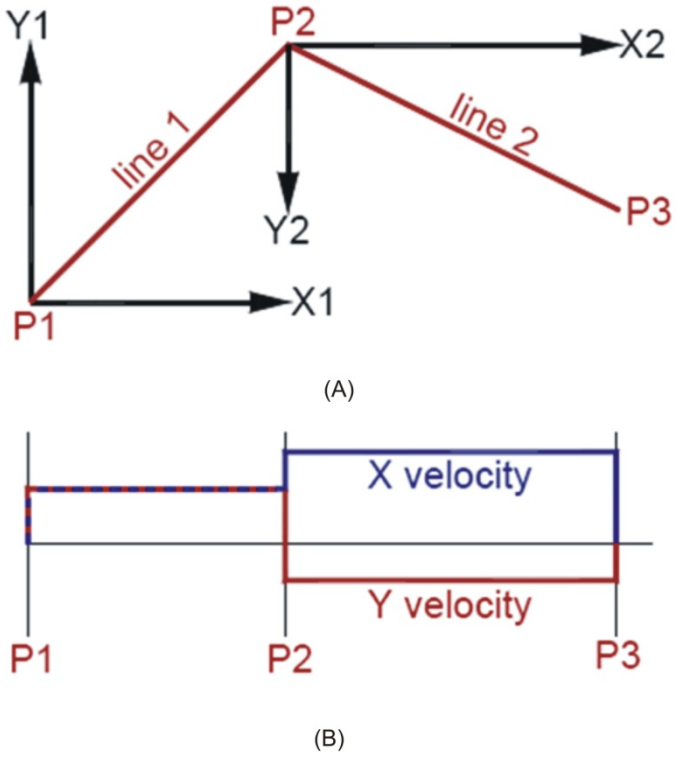

Figura 3-1 - Curva de posição e velocidade em degrau (GeckoMotion, 2005).

O controlador suaviza a curva de velocidade, como mostrado na Figura 3-2. Pois, mudanças bruscas de velocidade podem danificar o manipulador, além dos overshoots gerarem grandes desvios nas trajetórias.

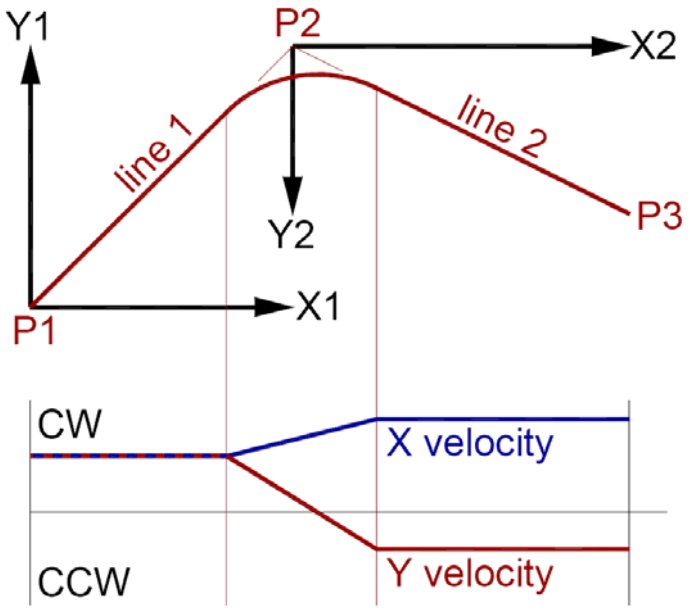

Figura 3-2 - Curva de posição e velocidade suavizada (GeckoMotion, 2005).

Um exemplo de controlador de juntas é o modelo G-REX G100, que é fabricado pela G-REX e é um controlador de movimento de seis eixos de rotação, que se conecta via USB ou Ethernet ao PC servidor. A comunicação é feita através de mensagens de comandos e de resposta. O G100 tem vinte e duas entradas de uso geral, dezesseis saídas de uso geral, quatro entradas analógicas e quatro saídas analógicas.

O G100 usa um FPGA (field-programmable gate array) TSK80 fabricado pela Altium, para gerar o sincronismo dos pulsos enviados para cada eixo de rotação e um processador de oito bits para controlar o movimento de cada eixo. Cada eixo pode funcionar independentemente ou em coordenação com outro eixo.

O FPGA produz freqüências de pulso proporcionais a uma palavra de 16 bits. A freqüência somente pode ser alterada em valores múltiplos de 1.024. As entradas, incluindo as analógicas, são amostradas nessa taxa e todas as saídas, incluindo as analógicas, são atualizadas nessa taxa.

Características de hardware do controlador G-REX G100:

- Seis entradas para encoders. Essas entradas são filtradas e têm LEDs (Light-Emitting Diodes) indicando cada entrada;

- Dezesseis saídas de uso geral. Cada saída tem um LED indicador que se acende quando uma saída é ligada;

- Vinte e duas entradas de uso geral, que são filtradas e protegidas. Cada entrada tem um LED indicador que acende quando uma entrada é aterrada;

- Quatro saídas analógicas, as quais podem variar de 0V a 5V;

- Quatro entradas analógicas de 0V a 5V, cada uma possuindo uma impedância de entrada de 2.2K Ω;

- O G100 requer uma alimentação de 12 ou 24VCC com corrente de 1.5A. Internamente fornece 5VCC para alimentar os terminais de saída;

- O G100 possui uma porta USB e uma Ethernet para comunicação com PC.

4. Desenvolvimento

4.1. Modelagem do Sistema

O diagrama de caso de uso do sistema está descrito na Figura 4-1.

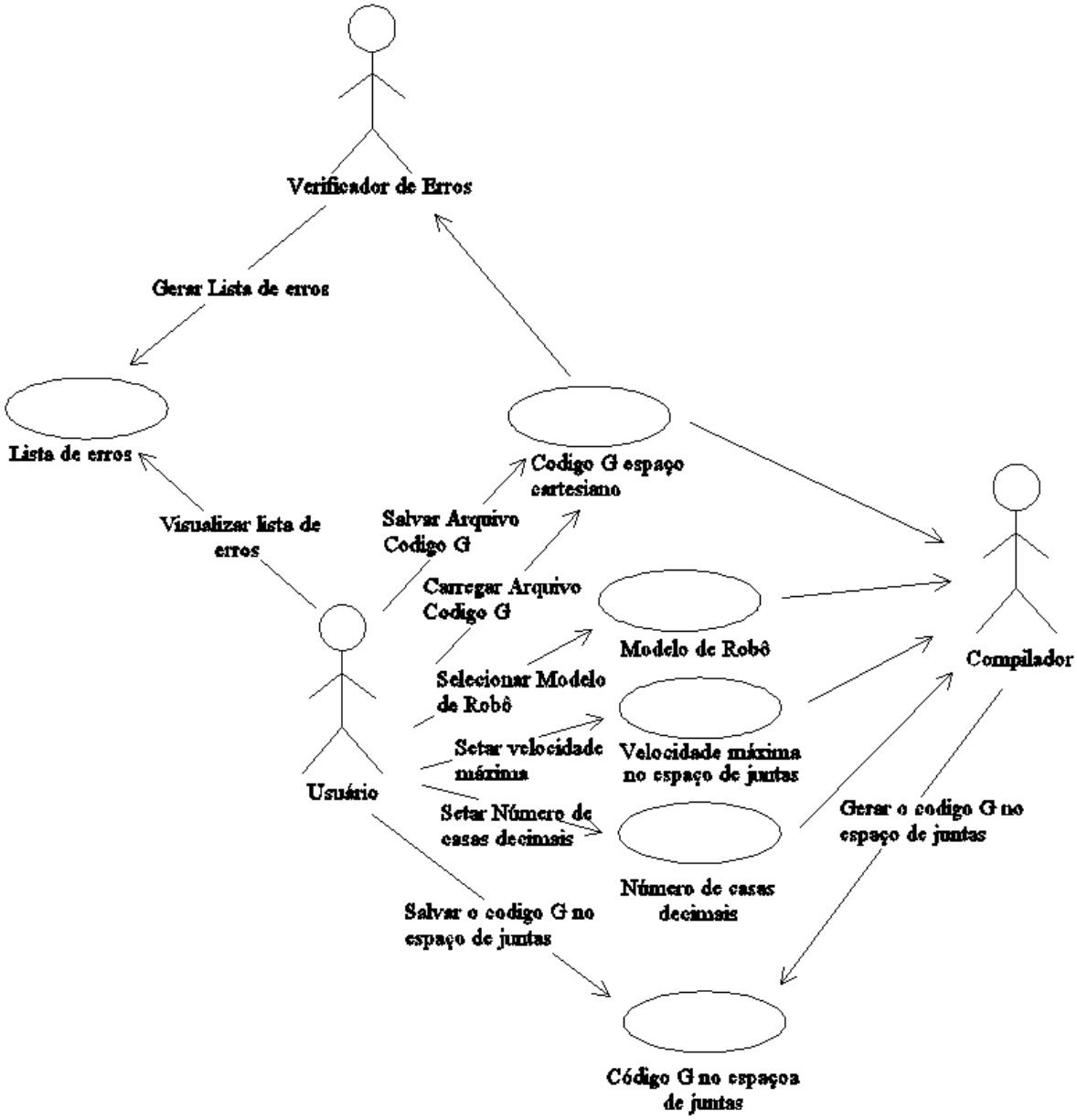

Figura 4-1 - Diagrama de caso de uso

O diagrama descrito na figura 4-1 descreve as possíveis funções a serem desempenhadas pelo usuário, mostrando também as funções que os outros atores do sistema estão aptos a desempenhar.

4.2. Desenvolvimento

O programa foi desenvolvido utilizando uma modelagem orientada a objetos, que permite que cada objeto seja desenvolvido e atualizado separadamente, facilitando a tarefa de atualização e manutenção do aplicativo.

A aplicação foi dividida em quatro partes:

- A primeira é a interface com o usuário, que tem como objetivo a "comunicação" com o mesmo. Por exemplo, o usuário pode carregar ou salvar um arquivo que contém o código G que descreve uma trajetória, a interface pode indicar erros no código e o usuário pode assim escolher o modelo de robô a ser utilizado, etc.

- A segunda parte é o verificador de erros, que é responsável por "varrer" o código G a procura de erros de digitação como, por exemplo, um comando G inexistente, um parâmetro sem valor determinado, um parâmetro com dois valores determinados no mesmo comando, etc.

- A terceira parte é responsável por interpretar a geometria dos comandos, calculando, para um determinado instante de tempo, o ponto em que o robô deve estar. Nesta

parte, também são verificados os “erros geométricos” como um comando G02 ou G03 que não descreva uma circunferência corretamente.

- A quarta parte é a responsável por descrever a cinemática inversa do robô. Descreve também seus limites físicos, definindo assim um espaço de atuação. É responsável também por verificar se algum comando pede que o robô extrapole o espaço de trabalho do robô. Por fim, é responsável por combinar o modelo cinemático do robô escolhido com os pontos no espaço cartesiano calculados na parte dois, obtendo uma lista de pontos no espaço de juntas do robô.

- A quinta parte é responsável por escrever um novo código G no espaço de juntas, a partir da lista de pontos gerada anteriormente.

4.3. Interface

Todos os elementos da interface estão ilustrados na Figura 4-2.

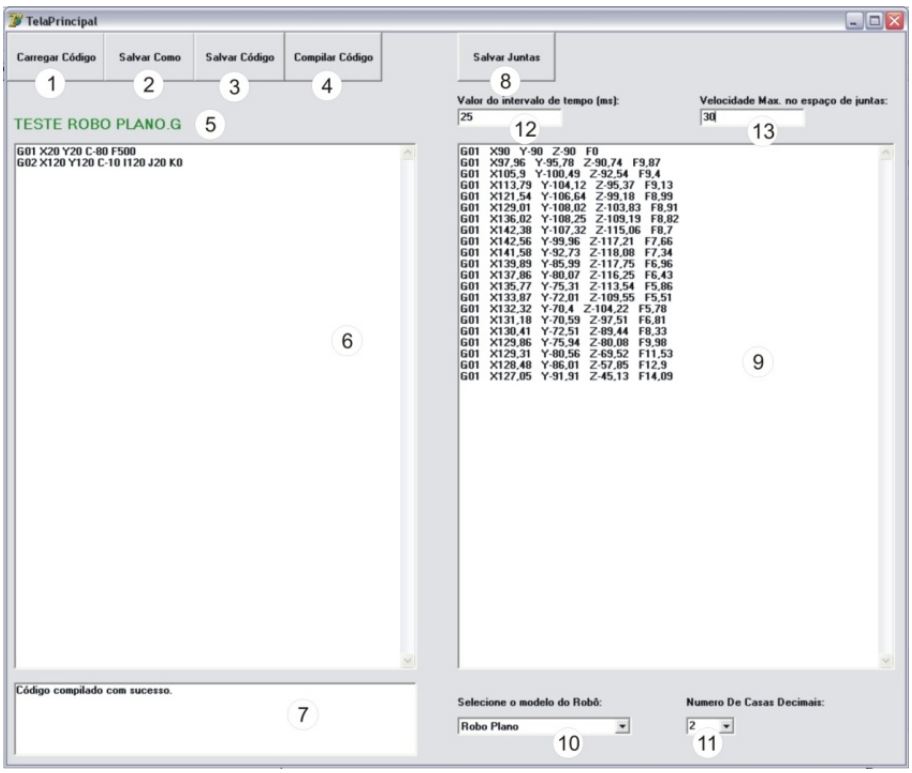

Figura 4-2 - Interface do aplicativo.

1. Botão Carregar Código: através deste botão o usuário pode carregar um arquivo “.g” contendo um código previamente salvo.

2. Botão Salvar Como: ao clicar neste botão o usuário pode salvar o código com um novo nome em um arquivo com extensão “.g”.

3. Botão Salvar Código: ao clicar neste botão o usuário salva o código com o nome mostrado em 5 em um arquivo com extensão “.g”.

4. Botão Compilar Código: esse botão é responsável por disparar as rotinas que vão verificar erros no código escrito em 6. Se não houver erros, esse código será combinado com o modelo do robô para gerar uma lista das posições das juntas do robô.

5. Label indicando o nome do arquivo carregado. Se estiver verde, indica que o código escrito em 6 é igual ao código salvo em arquivo; se estiver vermelho, indica que há alguma modificação no código.

6. Caixa de texto onde é escrito o código G a ser salvo ou compilado.

7. Mostra a lista de erros de compilação encontrados no código depois de compilado. Para cada erro encontrado, é adicionada uma linha que indica a posição na caixa de texto 6 em que o erro ocorreu e uma descrição do erro. Se o usuário clicar sobre a

descrição do erro, o cursor indicará em 6 a linha em que o erro ocorreu. Se não houve erros aparecerá a mensagem “Código foi compilado com sucesso”.

8. Botão Salvar Juntas: ao clicar neste botão, o usuário salva o código escrito em 9 com o nome mostrado em 5 adicionado de “- juntas”, em um arquivo com extensão “.g”.

9. Caixa de texto que mostra o novo código G nos espaços de juntas, gerado a partir do código escrito em 6, combinado com a cinemática inversa do robô selecionado na ComboBox 10.

10. ComboBox que lista os modelos de robôs que podem ser selecionados.

11. ComboBox para selecionar o número de casas decimais dos valores gerados no código escrito em 9.

12. EditBox indicando o intervalo de tempo, em milissegundos, desejado pelo usuário.

13. EditBox indicando a velocidade máxima no espaço de juntas.

4.4. Verificador de erros de sintaxe

Essa parte do programa é responsável por verificar a sintaxe do código G escrito pelo usuário. Esse é o primeiro procedimento que é disparado após o usuário clicar o botão “Compilar Código”, pois se forem encontrados erros de sintaxe no código, o programa não executará as tarefas subsequentes a essa.

Para verificar os erros de sintaxe foi desenvolvida uma classe de objeto TErroSintaxe, que possui uma interface capaz de receber uma linha de comando de código G e retornar um objeto do tipo TVetorErro.

O objeto TVetorErro é um objeto que contém um campo indicando o número da linha em que ocorreu o erro e um campo com a descrição do erro. Caso não haja erro, o campo que indica a linha em que ocorreu o erro é preenchida com o valor (-1).

A Tabela 4-1 mostra e descreva todos o métodos implementados na classe TErroSintaxe.

Tabela 4-1- Descrição dos métodos implementados na classe TErro.

procedure VerificaTamanhoComando(PComando:array of string; PIndice: integer);	Recebe um comando e verifica se o tamanho do comando é maior que um.
procedure VerificaIniciaisComando(PComando:array of string; PIndice: integer);	Recebe um comando e verifica se as iniciais de cada parâmetro são compatíveis com as iniciais possíveis (G,X,Y,Z,A,B,C,F,I,J,K).
procedure CalculaTamanhoComando(PComando:array of string);	Recebe um comando e calcula o tamanho deste, contando o numero de parâmetros.
procedure VerificaCompatibilidadeTipoComandoParametros(PComando:array of string;PIndice: integer);	Recebe um comando e verifica se os parâmetros são compatíveis com o tipo de comando.
procedure VerificaValorParametros(PComando:array of string;PIndice: integer);	Verifica se os valores dos parâmetros são números reais validos.
procedure VerificaParametrosRepetidos(PComando:array of string;PIndice: integer);	Verifica se não há parâmetros repetidos em um mesmo comando.
function RecebeVetorComando(PComando: array of string; PIndice: integer):TVetorErro;	Recebe o comando e retorna um erro se este ocorrer.

A Figura 4-3 mostra a ordem que os métodos serão executados.

```
function TErroSintaxe.RecebeVetorComando(PComando: array of string; PIndice: integer):TVetorErro;
begin
  VetorErro.Descricao := '';
  VetorErro.Indice := -1;
  CalculaTamanhoComando(PComando);
  VerificaTamanhoComando(PComando,PIndice);
  VerificaIniciaisComando(Pcomando,PIndice);
  VerificaCompatibilidadeTipoComandoParametros(Pcomando,PIndice);
  VerificaValorParametros(Pcomando,PIndice);
  VerificaParametrosRepetidos(Pcomando,PIndice);;
  RecebeVetorComando:= VetorErro;
end; {funcition}
```

Figura 4-3 - Ordem que os métodos de verificação de erros de sintaxe são executados.

4.5. Geometria dos comandos

Essa parte do programa é responsável por interpretar a geometria de cada comando, estando preparada para reconhecer os comandos G00, G01, G02 e G03. Também é capaz de reconhecer alterações no parâmetro “F”, o qual descreve a velocidade de movimento do robô em (mm/s).

O ponto inicial da trajetória é definido no modelo cinemático do robô. Todos os comandos possuem os parâmetros X, Y, Z, A, B e C que descrevem o ponto de destino e a orientação do comando. Qualquer um destes parâmetros pode ser omitido, mas não todos. O valor do parâmetro omitido permanecerá inalterado, mantendo o valor do ultimo comando que este parâmetro apareceu.

Para analisar a geometria destes comandos, foi desenvolvida uma classe TGeometria, a qual possui uma interface capaz de receber um objeto do tipo TComando, que é o objeto que descreve o comando G. Por fim, retorna um objeto do tipo TTrajetoria , que é uma lista de

pontos que descreve a trajetória. Essa classe também é responsável por retornar um objeto TVetorErro, indicando se algum erro foi encontrado na geometria do comando.

4.5.1. Comando G00

O comando G00 é o comando que descreve um movimento linear no espaço de juntas. Ele é utilizado quando se deseja posicionar o robô em um ponto, sem se importar com o trajeto percorrido pela ferramenta para chegar a esse ponto.

Para executar este comando o programa calcula a cinemática inversa do ponto de destino. O resultado disto são as posições que cada junta do robô deve assumir.

Por exemplo, o modelo do robô "Robô Plano" tem seu ponto inicial em (X = 100, Y = 0) e a orientação inicial da ferramenta é (C = - 90°). No espaço de juntas, este estado inicial do robô indica que as juntas estão posicionadas em (Junta1 = 90°, Junta2 = -90° e Junta3 = -90°) como ilustra a Figura 4-11. Deseja-se reposicioná-lo através do comando "G00 X50 Y50 A-60" no ponto (X = 50, Y = 50) e a com a orientação da ferramenta em (C = - 60°). Isto indica que o estado final no espaço de juntas deve ser (Junta1 = 136,92°, Junta2 = -93,84° e Junta3 = -103,07°) esses estados foram calculados usando o modelo cinemático do robô, que está descrito na seção 4.4. Para isso, o ângulo de cada junta é alterado linearmente até o ponto desejado, como mostrado na Figura 4-4.

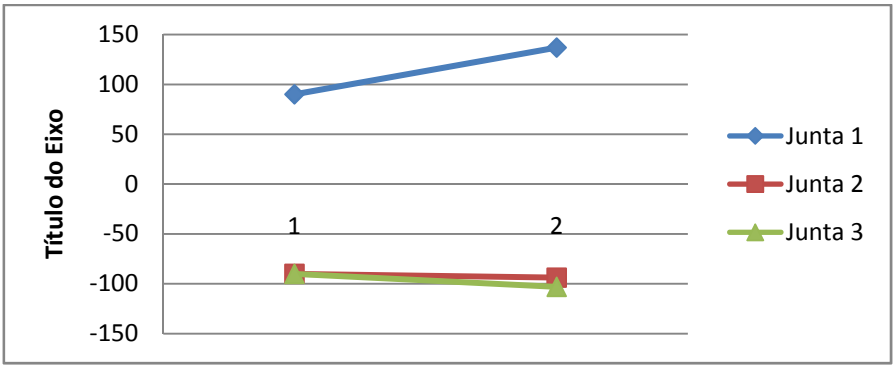

Figura 4-4 - Descrição do comando G00 executado pelo robô "Robô Plano" no espaço de juntas.

Este movimento linear no espaço de juntas provoca, neste caso, um movimento praticamente circular no espaço cartesiano como mostra a Figura 4-5.

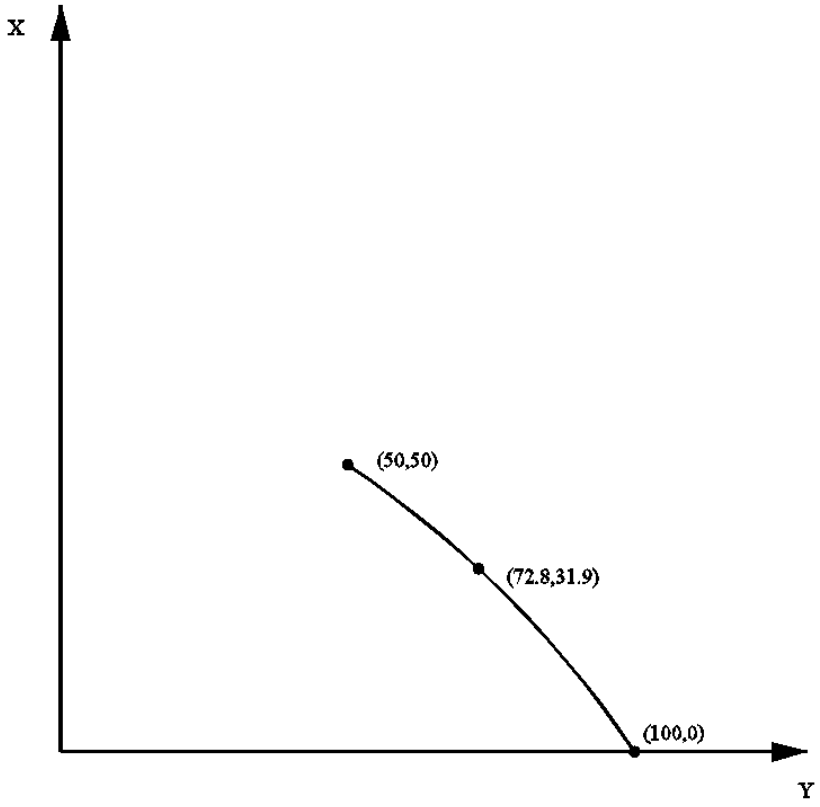

Figura 4-5 - Descrição do comando G00 executado pelo robô "Robô Plano" no espaço de cartesiano

Portanto, para o comando G00, o aplicativo não interpola os pontos entre o ponto inicial e o ponto final. Assim, o objeto do tipo TGeometria retorna um objeto do tipo TTrajetoria com apenas o ponto final convertido para o espaço de juntas.

4.5.2. Comando G01

O comando G01 é o comando que descreve um movimento linear no espaço cartesiano. É utilizado para que a ferramenta do robô vá de um ponto a outro, seguindo uma trajetória reta.

Para que o robô se mova em linha reta, o aplicativo interpola pontos intermediários desta trajetória no espaço cartesiano. O número de pontos entre o ponto inicial e final é determinado pelo parâmetro velocidade (F): quanto maior a velocidade, menor é o número de pontos interpolados. Sendo assim, o movimento é subdividido em pequenos segmentos de reta para cada intervalo de tempo, pois sem esta subdivisão ficaria muito difícil sincronizar os atuadores de cada junta do robô. A Figura 4.6 mostra os pontos no espaço cartesiano, os quais o robô "Robô Plano" deve percorrer, partindo de seu ponto inicial (X = 100, Y = 0, C = -90°) para o ponto final (X = 50, Y = 50, C = -60°), obedecendo ao comando "G01 X50 Y50 C-60 F100".

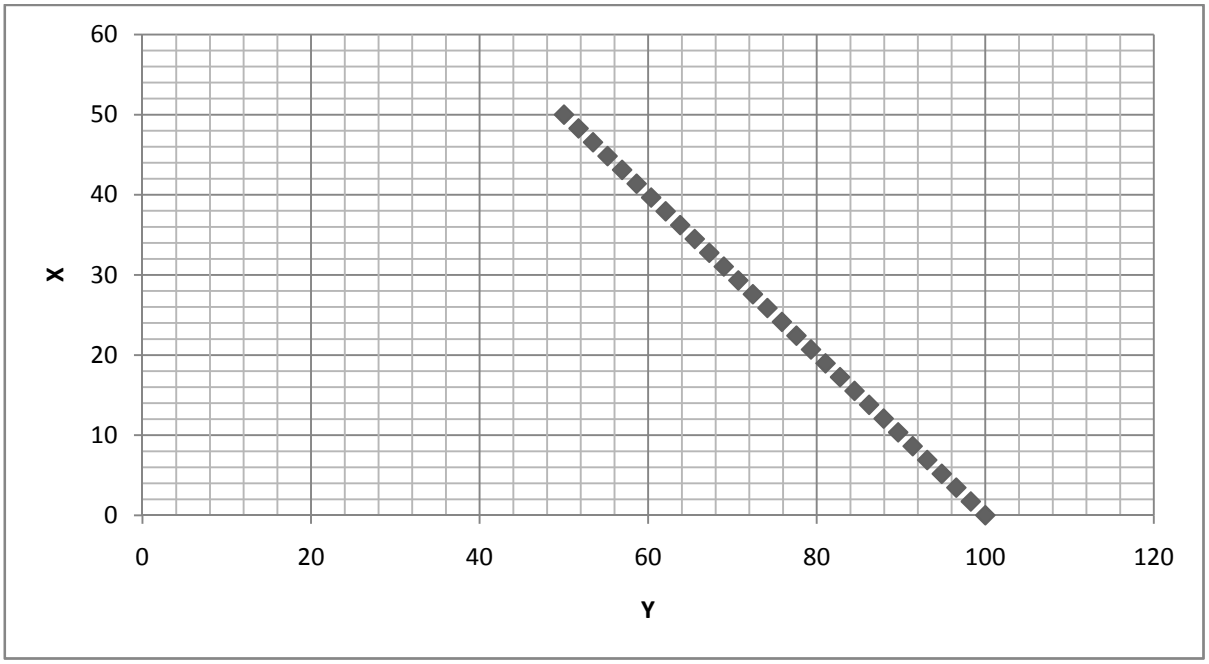

Figura 4-6 - Descrição do comando G01 executado pelo robô "Robô Plano" no espaço de cartesiano.

Para esta trajetória obtém-se, calculando a cinemática inversa em cada ponto da trajetória, o comportamento das juntas 1, 2 e 3, conforme demonstrado na Figura 4-7 .

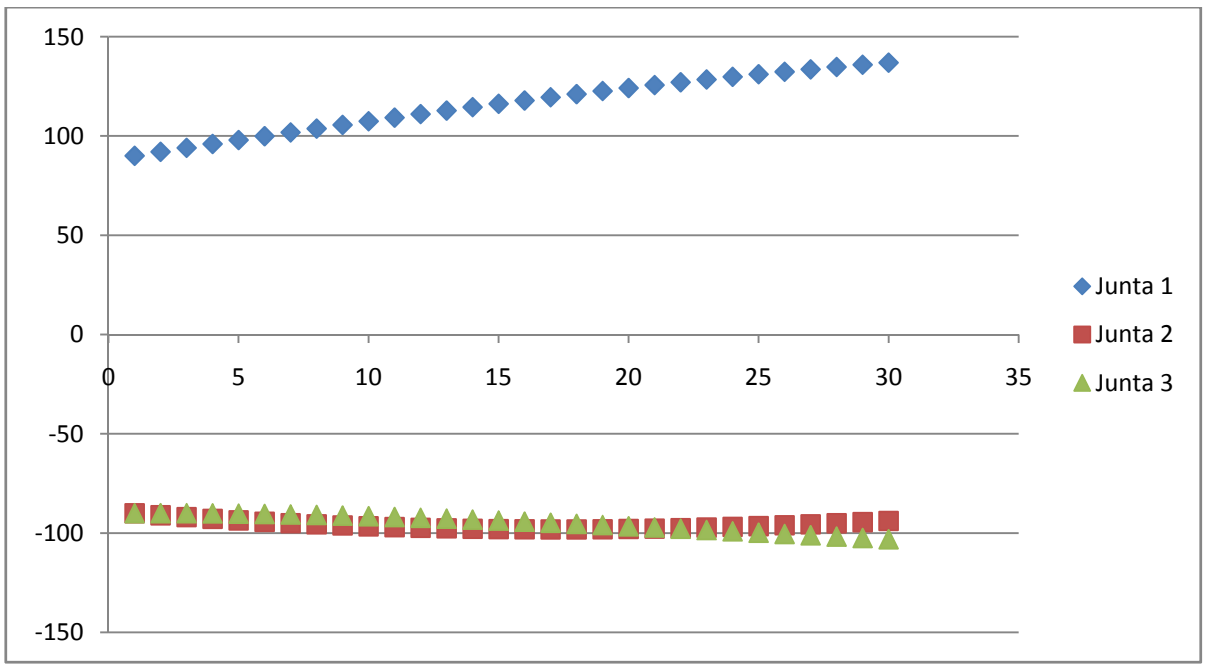

Figura 4-7 - Descrição do comando G01 executado pelo robô "Robô Plano" no espaço de juntas.

Através da Figura 4-7, pode-se perceber que nenhuma das juntas segue uma trajetória linear no espaço de juntas.

Para o cálculo destes pontos intermediários, deve-se primeiramente calcular a distância entre o ponto inicial (X_0, Y_0, Z_0) e o ponto final (X_N, Y_N, Z_N), como descrito na equação (4.1).

$$D = \sqrt{(X_N - X_0)^2 + (Y_N - Y_0)^2 + (Z_N - Z_0)^2} \qquad 4.1$$

Sabendo-se a distância (D) entre os pontos, a velocidade (F) em mm/s e o tempo (t) que o robô gasta para mover-se de um ponto ao outro, pode-se calcular o número de pontos (N) a ser interpolado juntamente com o ponto final, como mostra a equação (4.2).

$$N = trunc\left(\frac{D}{F*t}\right) + 1 \qquad 4.2$$

Sendo assim, no exemplo citado anteriormente, N = 29. Com o número de pontos (N) calculado, sabendo-se o ponto inicial e sua orientação (X_0, Y_0, Z_0, A_0, B_0, C_0) bem como as mesmas informações para o ponto final (X_N, Y_N, Z_N, A_N, B_N, C_N), é possível calcular os incrementos (I_X, I_Y, I_Z, I_A, I_B, I_C), como demonstrado pelas equações (4.3).

$$I_X = \frac{X_N - X_0}{N} \quad I_Y = \frac{Y_N - Y_0}{N} \quad I_Z = \frac{Z_N - Z_0}{N} \qquad 4.3$$

$$I_A = \frac{A_N - A_0}{N} \quad I_B = \frac{B_N - B_0}{N} \quad I_C = \frac{C_N - C_0}{N}$$

Assim sendo, os incrementos do exemplo citado são (I_X = -1,72, I_Y = 1,72, I_Z = 0, I_A = 0, I_B = 0, I_C = 1,03). Esses incrementos devem ser aplicados a cada coordenada do ponto inicial, para

que ele se “mova” linearmente até o ponto final. Assim, para calcular o ponto P_A (X_A, Y_A, Z_A), entre o ponto inicial e o ponto final, deve-se proceder como descrevem as equações (4.4).

$$0 \leq A \leq N$$

$$X_A = X_0 + (I_X * A)\ ,\quad Y_A = Y_0 + (I_Y * A)\ ,\quad Z_A = Z_0 + (I_Z * A) \qquad 4.4$$

Assim, o objeto do tipo TGeometria monta a “trajetória parcial”, no espaço cartesiano, do comando G01, representada por um objeto do tipo TTrajetoria. Um objeto do tipo TCompilador junta esta “trajetória parcial” ao seu próprio objeto TTrajetoria, que representa a trajetória somada de todos os comandos do código G.

4.5.3. Comando G02 e G03

Os comandos G02 e G03 são os comandos que descrevem um movimento em arco no espaço cartesiano. A diferença entre eles é que, no comando G02, a trajetória em arco é no sentido horário e, no G03, é no sentido anti-horário. Nestes comandos os parâmetros I, J e K são obrigatórios e determinam as coordenadas X, Y e Z, respectivamente, do centro do arco. A Figura 4-8 ilustra a diferença entre os comandos G02 e G03 para um ponto inicial (X = 0, Y= 0, Z = 0, A = 0, B = 0, C = 0), ponto final (X = 50, Y= 50, Z = 0, A = 0, B = 0, C = 0) e coordenadas de centro (I = 50, J = 0, K = 0).

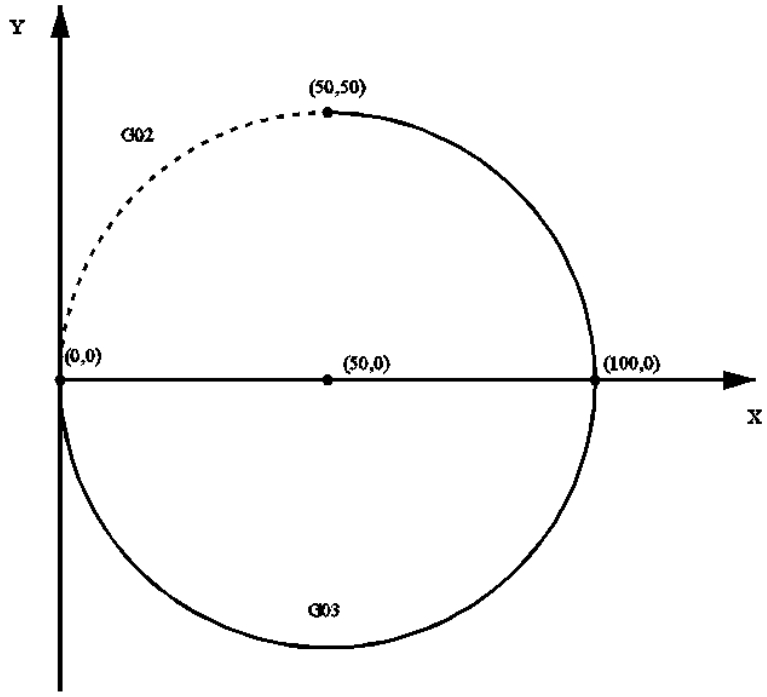

Figura 4-8 - Comparação da trajetória do comando G02 e G03.

A primeira tentativa de desenvolver a função que interpreta o comando G02 ou G03 foi a seguinte: primeiramente calcula-se o raio do arco (R_a), que pode ser obtido pela equação (4.1) que mede a distância entre pontos de centro P_c (X_c, Y_c, Z_c) e o ponto inicial P_0 (X_0, Y_0, Z_0). Depois são calculados dois vetores, um que vai do ponto de centro para o ponto inicial V_{ci} e outro V_{cf} que vai do ponto de centro ao ponto final P_n (X_n, Y_n, Z_n), esses vetores podem ser calculados como mostra a equação (4.5).

$$V_{ci} = (X_0 - X_c, Y_0 - Y_c, Z_0 - Z_c) \qquad 4.5$$

$$V_{cf} = (X_n - X_c, Y_n - Y_c, Z_n - Z_c)$$

Com os valores obtidos para os vetores, calcula-se o ângulo (θ) entre eles, como mostra a equação (4.6), em que acos representa a função arco cosseno.

$$\theta = \operatorname{acos}\left(\frac{V_{ci}.V_{cf}}{\|V_{ci}\| * \|V_{cf}\|}\right) \qquad 4.6$$

De posse do valor do raio e do ângulo é possível calcular o perímetro (D) do movimento, como descreve a equação (4.7).

$$D = \frac{2\pi * \theta * R}{2\pi} \qquad 4.7$$

Com o valor do perímetro calculado, o número de pontos entre o ponto inicial e final é determinado pela equação (4.2). O próximo passo é determinar o plano do arco; para isso é necessário primeiro calcular o produto vetorial entre V_{ci} e V_{cf}. O resultado deste produto será um vetor normal V_n ao plano, como descreve a equação (4.8).

$$V_{ci} = (X_{ci}, Y_{ci}, Z_{ci}) \ e \ \ V_{cf} = (X_{cf}, Y_{cf}, Z_{cf})$$

$$V_n = \left(\begin{vmatrix} Y_{ci} & Z_{ci} \\ Y_{cf} & Z_{cf} \end{vmatrix}, \quad \begin{vmatrix} X_{ci} & Z_{ci} \\ X_{cf} & Z_{cf} \end{vmatrix}, \quad \begin{vmatrix} X_{ci} & Y_{ci} \\ X_{cf} & Y_{cf} \end{vmatrix}\right) \qquad 4.8$$

Para a completa descrição do plano devem-se saber as coordenadas de um ponto que pertença ao plano e um vetor normal ao plano. O ponto de centro pertence ao plano e V_n é um vetor normal ao plano. Então a equação (4.9) determina se um ponto P_a (X_a, Y_a, Z_a) pertence ao plano do arco:

$$V_n = (X_n, Y_n, Z_n)$$

$$X_n * X_a + Y_n * Y_a + Z_n * Z_a - (X_n * X_c + Y_n * Y_c + Z_n * Z_c) = 0 \qquad 4.9$$

Agora define-se uma esfera de centro no ponto de centro do arco e raio igual ao arco, isto é, o ponto P_a (X_a, Y_a, Z_a) pertence a esta esfera se a equação (4.10) for verdadeira:

$$X_a^2 + Y_a^2 + Z_a^2 - 2(X_a * X_c + Y_a * Y_c + Z_a * Z_c) + X_c^2 + Y_c^2 + Z_c^2 - R_a^2 = 0 \qquad 4.10$$

Define-se finalmente, então, uma nova esfera com centro no ponto inicial da trajetória e raio (R_p) igual ao perímetro do movimento dividido pelo número de pontos calculados pela equação (4.2). A equação desta nova esfera é ilustrada na equação (4.11).

$$X_a^2 + Y_a^2 + Z_a^2 - 2(X_a * X_0 + Y_a * Y_0 + Z_a * Z_0) + X_0^2 + Y_0^2 + Z_0^2 - R_p^2 = 0 \qquad 4.11$$

Assim o primeiro ponto P_1 da interpolação entre o ponto inicial e final é calculado pelo sistema descrito pela equação (4.12).

$$\begin{cases} X_n * X_a + Y_n * Y_a + Z_n * Z_a - (X_n * X_c + Y_n * Y_c + Z_n * Z_c) = 0 \\ X_a^2 + Y_a^2 + Z_a^2 - 2(X_a * X_c + Y_a * Y_c + Z_a * Z_c) + X_c^2 + Y_c^2 + Z_c^2 - R_a^2 = 0 \\ X_a^2 + Y_a^2 + Z_a^2 - 2(X_a * X_0 + Y_a * Y_0 + Z_a * Z_0) + X_0^2 + Y_0^2 + Z_0^2 - R_p^2 = 0 \end{cases} \qquad 4.12$$

O sistema representado pela equação (4.12) fornece duas respostas para cada coordenada; determina-se a resposta desejada sabendo-se o sentido do movimento (comando G02, G03). Assim os pontos (P_2, P_3, ... P_n) são calculados utilizando a equação (4.11), somente alterando as coordenadas do ponto de centro da esfera para as coordenadas do ponto anteriormente calculado. A Figura 4-9 ilustra esta metodologia.

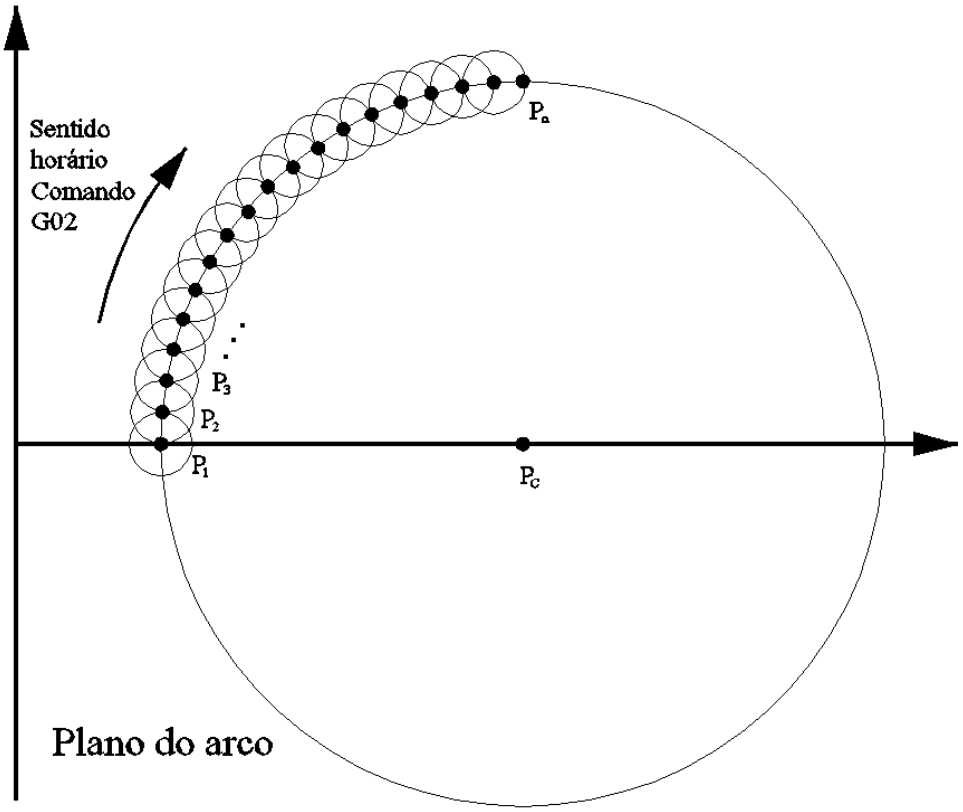

Figura 4-9 - Primeira metodologia para desenvolvimento dos comandos G02 e G03.

O problema desta metodologia é que o sistema de equações descrito pela equação (4.12) é de resolução muito complicada.

Optou-se, então, por desenvolver outra metodologia para o cálculo da trajetória, discutida a seguir. Primeiramente, o sistema de coordenadas fixo inicial é definido como {0}, sendo sua origem localizada na base do robô; depois o raio (R_a) do arco, o perímetro do movimento e o número de pontos (N) são calculados como na metodologia anterior. Agora cria-se um novo sistema de coordenadas {E}. Para determinar este novo sistema de coordenadas, define-se um vetor $\hat{}R_{Ex}$, que é o vetor V_{ci} transformado em unitário do vetor $\hat{}X_E$, como descreve a equação (4.13):

$$\hat{}R_{Ex} = \left(\frac{X_{c0}}{\|V_{ci}\|}, \frac{Y_{c0}}{\|V_{ci}\|}, \frac{Z_{c0}}{\|V_{ci}\|}\right) \qquad 4.13$$

Assim, o eixo X do novo sistema {E} aponta do centro do arco para o ponto inicial. O próximo passo é determinar o vetor unitário $^{\wedge}R_{Ez}$, que representa o eixo Z do novo sistema de coordenadas. Para isso, determina-se o vetor V_n normal ao plano do arco, que é calculado como descreve a equação (4.8). O próximo passo é transformar em unitário o vetor normal. Como $^{\wedge}R_{Ez}$ é igual $^{\wedge}V_n$ então a equação (4.14) mostra como calcular o vetor $^{\wedge}R_{Ez}$.

$$^{\wedge}R_{Ez} = \left(\frac{X_n}{\|V_n\|}, \frac{Y_n}{\|V_n\|}, \frac{Z_n}{\|V_n\|}\right) \qquad 4.13$$

Para finalizar a descrição do sistema de coordenadas {E} deve-se calcular $^{\wedge}R_{Ey}$. Para isso calcula-se o produto vetorial de $^{\wedge}R_{Ez}$ por $^{\wedge}R_{Ex}$, como descreve a equação (4.14).

$$^{\wedge}R_{Ez} = (X_{Ez}, Y_{Ez}, Z_{Ez}) \;\; e \;\; ^{\wedge}R_{Ex} = (X_{Ex}, Y_{Ex}, Z_{Ex})$$

$$^{\wedge}R_{Ey} = \left(\begin{vmatrix} Y_{Ez} & Z_{Ez} \\ Y_{Ex} & Z_{Ex} \end{vmatrix}, \quad \begin{vmatrix} X_{Ez} & Z_{Ez} \\ X_{Ex} & Z_{Ex} \end{vmatrix}, \quad \begin{vmatrix} X_{Ez} & Y_{Ez} \\ X_{Ex} & Y_{Ex} \end{vmatrix}\right) \qquad 4.14$$

Para descrever o sistema de coordenadas {E} em relação ao sistema {0}, deve-se determinar o ponto de origem do sistema {E}. Define-se a origem no centro do arco, sendo então $^{0}P_{Eorg}$ igual a P_c. Assim, a matriz transformada homogênea que descreve {E} em relação a {0} é mostrada pela equação (4.15).

$$^{0}_{E}T = \begin{bmatrix} ^{\wedge}R_{Ex} & ^{\wedge}R_{Ey} & ^{\wedge}R_{Ez} & ^{0}P_{Eorg} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 4.15$$

Como $^{\wedge}R_{Ez}$ é perpendicular ao plano do arco, então todo ponto do arco tem a coordenada ^{E}Z igual a zero.

Define-se agora uma circunferência no plano XY do sistema de coordenada {E}. Este círculo tem raio R_a. Calculando o ângulo (θ) entre V_{ci} e $V_{cf,}$ como demonstrado na equação (4.6), e de posse do número de pontos (N), é possível determinar um incremento angular (I_θ) como descreve a equação (4.16).

$$I_\theta = \frac{\theta}{N} \qquad 4.16$$

Assim, para calcular o ponto ${}^{E}P_K$, deve-se proceder como descrevem as equações (4.17).

$${}^{E}P_K = ({}^{E}X_K, {}^{E}Y_K, {}^{E}Z_K) \quad , \quad 0 \leq K \leq N \quad e \quad {}^{E}Z_K = 0 \; \forall \; K$$

$${}^{E}X_K = R_A * cos(I_\theta * K) \quad , \quad {}^{E}Y_K = R_A * sen(I_\theta * K) \quad e \quad {}^{E}Z_K = 0 \qquad 4.17$$

A Figura 4-10 ilustra este procedimento.

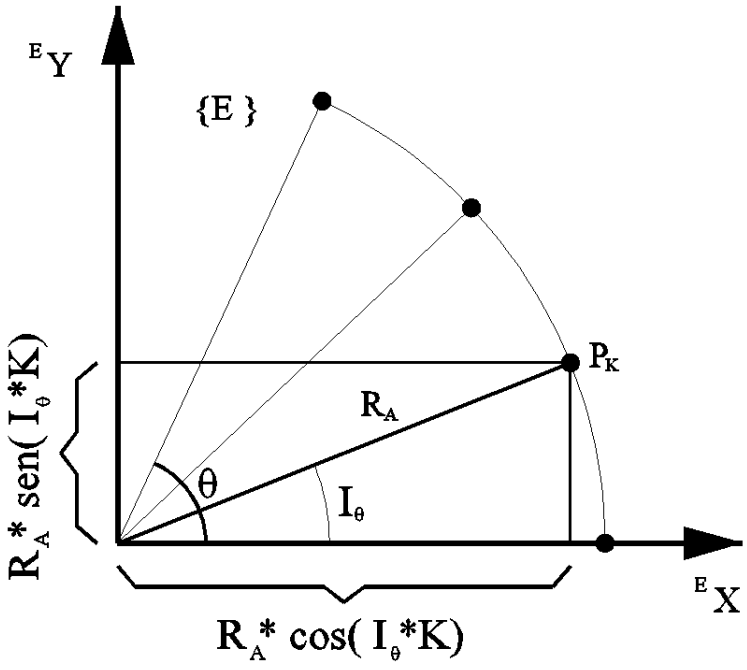

Figura 4-10 – Metodologia utilizada no programa para desenvolvimento dos comandos G02 e G03.

Assim, podem-se calcular as coordenadas dos pontos intermediários no sistema coordenado {E}. Agora, aplicando-se a equação (4.18), pode-se descrever o ponto P_K no sistema de coordenadas {0}, obtendo assim ${}^0P_K$.

$$ {}^0P_K = {}^0_E T \, . \, {}^E P_K \qquad 4.18 $$

Enfim, o objeto do tipo TGeometria monta a "trajetória parcial", no espaço cartesiano, do comando G02 ou G03, representada por um objeto do tipo TTrajetoria. O objeto TCompilador junta esta "trajetória parcial" ao seu próprio objeto TTrajetoria, que representa a trajetória somada de todos os comandos do código G.

4.6. Modelo cinemático do robô

Essa parte do programa é responsável por descrever o modelo cinemático do robô. Para isso foi desenvolvida uma classe TCinematicaRobo, que é capaz de receber um ponto no espaço cartesiano e verificar se este ponto está dentro do espaço de trabalho do robô. Se não estiver, retorna um objeto TVetorErro descrevendo este erro, e indicando a linha de código em que o mesmo ocorreu. Possui também uma interface capaz de retornar o ponto inicial do robô. Sendo assim, o objeto TCompilador envia para o objeto que descreve o modelo do robô, um a um todos os pontos calculados pelo objeto TGeometria e recebe todos os pontos agora representados no espaço de juntas.

A classe TCinemáticaRoboPlano é uma classe derivada de TCinematicaRobo. Somente a função "AplicaCinematicaInversaAoPontoCartesiano" é uma função especifica da classe TCinemáticaRoboPlano. Esta função descreve a cinemática inversa do modelo de robô "Robô

Plano". Outras classes podem ser, então, criadas de forma a implementar a cinemática de outros manipuladores.

Pelo desenho esquemático, Figura 4-11, pode-se ver que este modelo de robô, que é um robô planar, possui três graus de liberdade, ou seja, só é capaz de se mover no plano XY. Então, Z, A e B tem que ser sempre igual a zero, o que indica que ele não é capaz de variar sua coordenada Z e também não é capaz de rotacionar a sua ferramenta em torno dos eixos X e Y. Observa-se também que sua posição inicial é ($X = L_2$, $Y = 0$, $Z = 0$, $A = 0$, $B = 0$, $C = -90$). Os comprimentos L_1, L_2, L_e dos braços do manipulador são declarados como constantes no programa, sendo seus valores facilmente alterados.

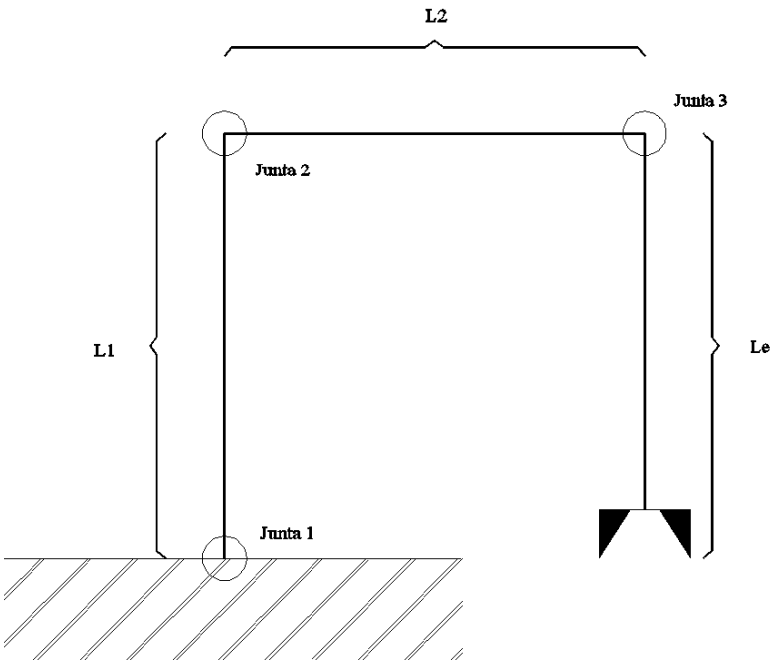

Figura 4-11 - Desenho esquemático do robô "Robô Plano em sua posição inicial."

Na Figura 4-12 estão descritas todas variáveis necessárias para descrever o modelo cinemático do robô.

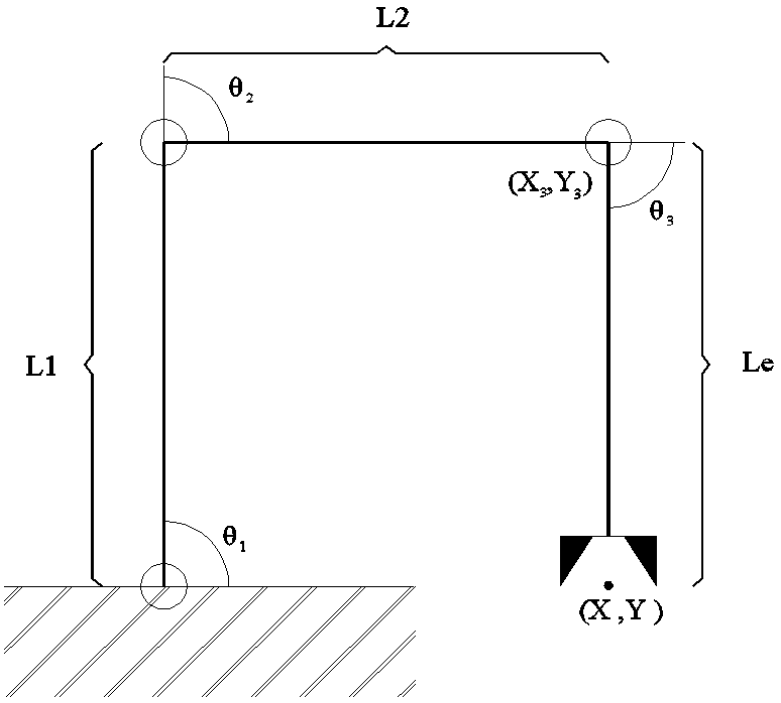

Figura 4-12 - Variáveis do modelo cinemático do robô "Robô Plano".

Assim conhecendo-se o ponto P da ponta da ferramenta no espaço cartesiano, pode-se convertê-lo a um ponto equivalente no espaço de juntas utilizando-se as equações em (4.18) (Craig, 1989).

$$Sendo\ \ P = (X, Y, Z, A, B, C) \quad e \quad Z = A = B = 0$$

$$X_3 = X - L_e * cos\emptyset \quad , \quad Y_3 = Y - L_e * sen\emptyset \quad e \quad \emptyset = C$$

$$\cos(\theta_2) = \frac{(X_3^2 + Y_3^2 - L_1^2 - L_2^2)}{2 * L_1 * L_2}$$

$$sen(\theta_2) = \sqrt{1 - cos^2(\theta_2)}\ , \qquad sendo \qquad sen(\theta_2) \leq 0\ \forall\ \theta_2 \qquad 4.18$$

$$\theta_2 = \text{atan2}\left(\frac{sen(\theta_2)}{cos(\theta_2)}\right)$$

$$Definindo \quad K_1 = L_1 + L_2 * \cos(\theta_2) \quad e \quad K_2 = L_2 * sen(\theta_2)$$

$$\theta_1 = \text{atan2}\left(\frac{Y_3}{X_3}\right) - \text{atan2}\left(\frac{K_2}{K_1}\right) \quad e \quad \theta_3 = \emptyset - \theta_1 - \theta_2$$

A partir das equações 4.18 converte-se o ponto P (X, Y, C) para o espaço de juntas do robô (θ_1, θ_2, θ_3).

O objeto TCinematicaRoboPlano vai retornar um objeto do tipo TPontoNoTempoJuntas que é um objeto que descreve o ponto no espaço de juntas e o instante de tempo que o robô deve estar nele. O objeto TCompilador junta este ponto ao seu próprio objeto TTrajetoriaJuntas, que representa a trajetória no espaço de juntas de todos os comandos do código G.

4.7. Gerar o código G no espaço de juntas

Esta é a parte mais simples do programa, sendo responsável somente por montar o código G no espaço de juntas. Para isso a lista de pontos no espaço de juntas é combinada com o número de casas decimais desejadas pelo usuário. Uma série de comandos G01 no espaço de juntas é então gerado. O parâmetro F é calculado segundo a equação (5.1).

$$F = \frac{\sqrt{\Delta X + \Delta Y + \Delta Z + \Delta A + \Delta B + \Delta C}}{\Delta t} \qquad 5.1$$

Por exemplo, para os comandos “G01 X20 Y20 C-80 F500”, “G02 X120 Y120 C-10 I120 J20 K0” é gerada o código G no espaço de juntas mostrado na Figura 4-13.

```
G01  X90  Y-90  Z-90  F0
G01  X97,96  Y-95,78  Z-90,74  F394,85
G01  X105,9  Y-100,49  Z-92,54  F376,22
G01  X113,79  Y-104,12  Z-95,37  F365,36
G01  X121,54  Y-106,64  Z-99,18  F359,72
G01  X129,01  Y-108,02  Z-103,83  F356,44
G01  X136,02  Y-108,25  Z-109,19  F353,04
G01  X142,38  Y-107,32  Z-115,06  F348,31
G01  X142,56  Y-99,96  Z-117,21  F306,67
G01  X141,58  Y-92,73  Z-118,08  F293,97
G01  X139,89  Y-85,99  Z-117,75  F278,44
G01  X137,86  Y-80,07  Z-116,25  F257,42
G01  X135,77  Y-75,31  Z-113,54  F234,42
G01  X133,87  Y-72,01  Z-109,55  F220,44
G01  X132,32  Y-70,4  Z-104,22  F231,2
G01  X131,18  Y-70,59  Z-97,51  F272,44
G01  X130,41  Y-72,51  Z-89,44  F333,32
G01  X129,86  Y-75,94  Z-80,08  F399,41
G01  X129,31  Y-80,56  Z-69,52  F461,52
G01  X128,48  Y-86,01  Z-57,85  F516,17
G01  X127,05  Y-91,91  Z-45,13  F563,69
```

Figura 4-13 - Exemplo de código G no espaço de juntas.

No código G no espaço de juntas, X, Y, Z, A, B e C representam as juntas 1, 2, 3, 4, 5 e 6 respectivamente. No exemplo da Figura 4-13 só foram representadas as juntas X, Y, Z porque o modelo de robô “Robô Plano” só possui três juntas.

5. Resultados e Testes

Este trabalho resulta no desenvolvimento de uma ferramenta capaz de interpretar uma trajetória no espaço cartesiano escrita em código G, subdividindo esta trajetória em uma lista de pontos interpolados e convertendo-a em uma lista de pontos no espaço de juntas.

Como saída, este aplicativo fornece um arquivo ".G" contendo o código G no espaço de juntas e dois arquivos ".csv", um contendo todos os pontos interpolados no espaço cartesiano e o outro representando estes mesmos pontos no espaço de estados.

Como exemplo, tem-se a seleção de um robô do modelo "Robô Plano" com os comprimentos de braço L_1, L_2 e L_e iguais a 100, sendo selecionado 2 casas decimais para o código G no espaço de juntas e intervalo de tempo de 2ms. O código G no espaço cartesiano que apresenta a trajetória a ser seguida está representado pela Figura 5-1.

```
G01 X20 Y20 C-80 F500
G02 X120 Y120 C-10 I120 J20 K0
```

Figura 5-1 - Código G descrevendo a trajetória a ser seguida pelo robô modelo "Robô Plano".

Esses comandos representam a trajetória mostrada na Figura 5-2 e a Figura 5-3 mostra todos os pontos interpolados no espaço cartesiano.

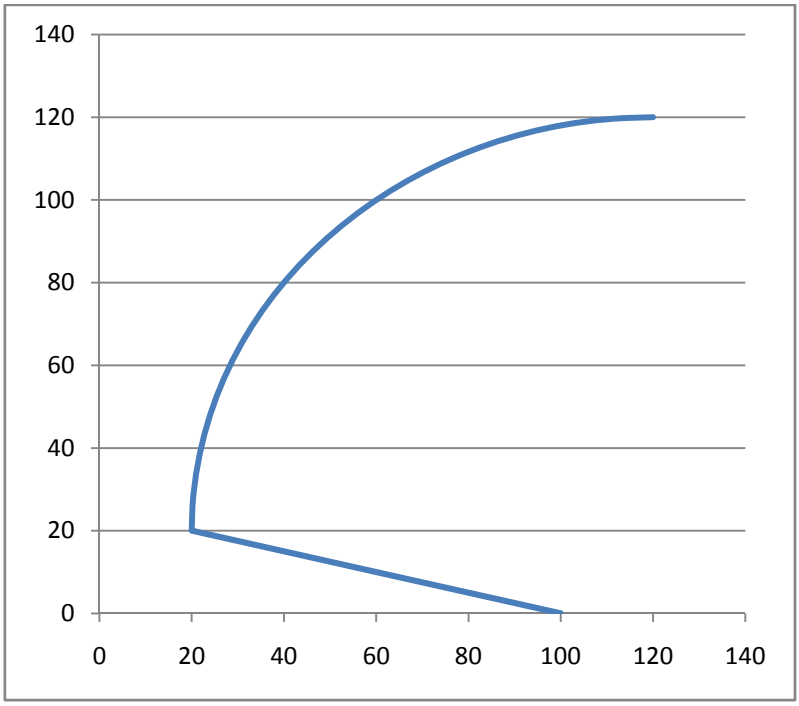

Figura 5-2 - Trajetória do movimento descrito pelo código da Figura 5-1.

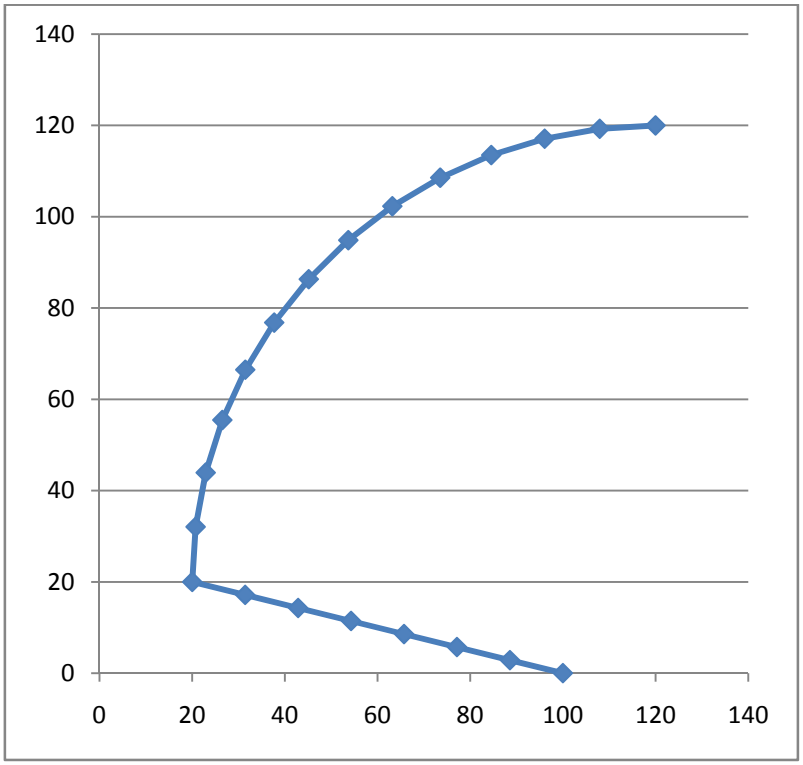

Figura 5-3 - Pontos interpolados levando em consideração o código descrito na Figura 5-1.

```
G01   X90   Y-90   Z-90   F0
G01   X97,96   Y-95,78   Z-90,74   F394,85
G01   X105,9   Y-100,49   Z-92,54   F376,22
G01   X113,79   Y-104,12   Z-95,37   F365,36
G01   X121,54   Y-106,64   Z-99,18   F359,72
G01   X129,01   Y-108,02   Z-103,83   F356,44
G01   X136,02   Y-108,25   Z-109,19   F353,04
G01   X142,38   Y-107,32   Z-115,06   F348,31
G01   X142,56   Y-99,96   Z-117,21   F306,67
G01   X141,58   Y-92,73   Z-118,08   F293,97
G01   X139,89   Y-85,99   Z-117,75   F278,44
G01   X137,86   Y-80,07   Z-116,25   F257,42
G01   X135,77   Y-75,31   Z-113,54   F234,42
G01   X133,87   Y-72,01   Z-109,55   F220,44
G01   X132,32   Y-70,4   Z-104,22   F231,2
G01   X131,18   Y-70,59   Z-97,51   F272,44
G01   X130,41   Y-72,51   Z-89,44   F333,32
G01   X129,86   Y-75,94   Z-80,08   F399,41
G01   X129,31   Y-80,56   Z-69,52   F461,52
G01   X128,48   Y-86,01   Z-57,85   F516,17
G01   X127,05   Y-91,91   Z-45,13   F563,69
```

Figura 5-4 - Código G no espaço de juntas, obtido a partir do código descrito na Figura 5-1.

A Figura 5-4 representa o código G no espaço de juntas, gerado levando em consideração todas as condições anteriores. A Figura 5-5 descreve o comportamento de todas as três juntas do robô, enquanto executa a trajetória ilustrada na Figura 5-2.

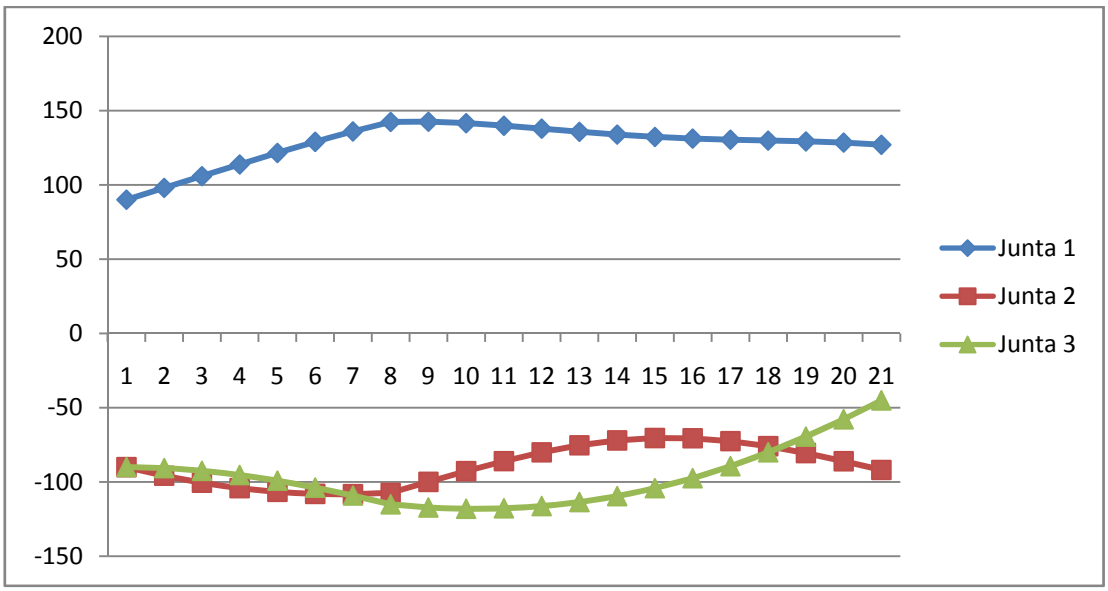

Figura 5-5 - Comportamento das juntas do robô, enquanto ele executa a trajetória descrita na Figura 1

A Figura 5-6 mostra estes mesmos resultados obtidos na interface do programa.

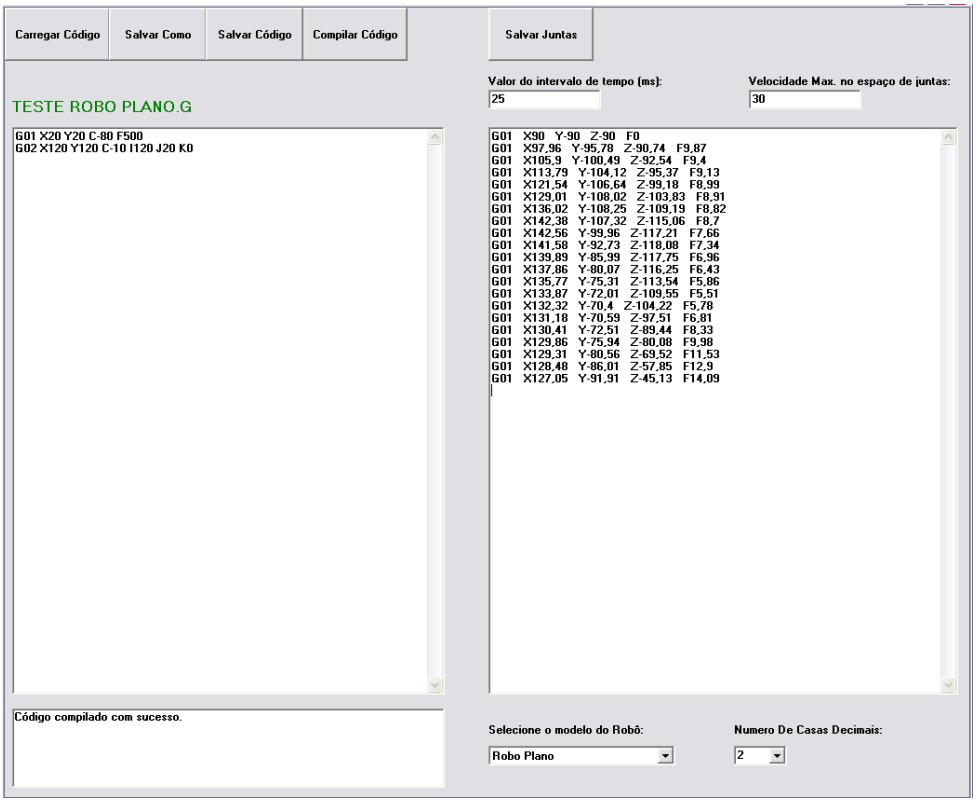

Figura 5-6- Resultados mostrados na interface do programa.

Se o comando inicial “G01” for substituído pelo comando “G00”, o código G no espaço de juntas fica muito diferente como mostra Figura 5-7. Percebe-se que todos os pontos da interpolação linear do comando “G01” foram apagados.

```
G01  X90  Y-90  Z-90  F0
G01  X142,38  Y-107,32  Z-115,06  F2424,13
G01  X142,56  Y-99,96  Z-117,21  F306,67
G01  X141,58  Y-92,73  Z-118,08  F293,97
G01  X139,89  Y-85,99  Z-117,75  F278,44
G01  X137,86  Y-80,07  Z-116,25  F257,42
G01  X135,77  Y-75,31  Z-113,54  F234,42
G01  X133,87  Y-72,01  Z-109,55  F220,44
G01  X132,32  Y-70,4  Z-104,22  F231,2
G01  X131,18  Y-70,59  Z-97,51  F272,44
G01  X130,41  Y-72,51  Z-89,44  F333,32
G01  X129,86  Y-75,94  Z-80,08  F399,41
G01  X129,31  Y-80,56  Z-69,52  F461,52
G01  X128,48  Y-86,01  Z-57,85  F516,17
G01  X127,05  Y-91,91  Z-45,13  F563,69
```

Figura 5-7 - Substituição do comando G01 por G00.

O teste das trajetórias no robô real foi inviabilizado, pois este ainda se encontra em fase de construção.

Foram então realizados testes simulados, a partir de diversas trajetórias descritas em códigos G. Para a validação destas respostas do aplicativo, foi utilizado um modelo do robô em AutoCad, em que os ângulos de cada junta do robô eram variados como os valores descritos pela saída do programa, e era verificado se estes valores geravam uma trajetória como a descrita pelo código G inicial.

6. Conclusões

Ao fim deste trabalho, que foi desenvolvido pelo aluno de graduação em engenharia de controle e automação, Francisco Rodrigues Alves de Oliveira, o aplicativo proposto foi desenvolvido e está apto a ser utilizado na programação de trajetórias dos robôs pré-configurados.

Sendo confiável devido aos variados testes e simulações feitos para a validação das funcionalidades do aplicativo.

A inclusão de novos modelos de robôs pode ser feita de forma simples, pois cada robô é um objeto diferente. Sendo assim, alterações e inclusões de novos robôs não interferem na configuração dos robôs previamente configurados.

A maior dificuldade encontrada foi determinar a maneira mais simples de interpolar os pontos dos comandos G02 e G03. A primeira metodologia resultou em sistema de equações de difícil solução. Depois de algumas tentativas e algum tempo, a segunda metodologia foi desenvolvida, resultando em uma metodologia confiável e relativamente simples.

7. Bibliografia

CNC Information. 2008. CNC Information. *CNC Information.* [Online] CNC Information, 05 de 04 de 2008. http://www.cncinformation.com.

Craig, J. J. 1989. *Introduction to Robotics Mechanics and Control.* 2nd edition. s.l. : Addison-Wesley Publishing Company, Inc, 1989.

GeckoMotion, Inc. 2005. *GREX G100 Manual.* 2005. p. 22.

H. M. Deitel, P. J. Deitel. 2004. *C++ Como Programar.* 3° edição. s.l. : Bookman, 2004. p. 1098. 85-7307-740-9.

ISO, 10218. 1992. *Manipulating industrial robots - Safety.* 1992.

Lima II, E.J. 2005. *Soldagem robotizada com eletrodo revestido.* 2005, p. 85.

Ramalho, F.A.F. 2003. *Desenvolvimento de uma interface operacional baseada em um IBM-PC para programação e controle de um torno CNC modernizado.* Fevereiro de 2003, p. 100.

Romi, S.A. 1996. *Manual de programação e operação CNC MACH 9.* 1996, p. 70.

Szkodny, T. 1995. Forward and inverse kinematics of irb-6 manipulator. *Mechanism and Machine Theory.* 1995, pp. 1039-1056.