TRABALHO DE GRADUAÇÃO

MODELAGEM CINEMÁTICA, SIMULAÇÃO E CONTROLE DE TRAJETÓRIAS DE UM ROBÔ ARTICULADO DE 7 GRAUS DE LIBERDADE UTILIZANDO CONTROLE EMBARCADO EM FPGA

Diogo Camargos Gomes

Brasília, julho de 2014

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

MODELAGEM CINEMÁTICA, SIMULAÇÃO E CONTROLE DE TRAJETÓRIAS DE UM ROBÔ ARTICULADO DE 7 GRAUS DE LIBERDADE UTILIZANDO CONTROLE EMBARCADO EM FPGA

Diogo Camargos Gomes

Relatório submetido ao Departamento de Engenharia Mecatrônica como requisito parcial para obtenção do grau de Engenheiro Mecatrônico

Banca Examinadora

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

Prof. Carlos Humberto Llanos Quintero, ENM/UnB _____
Co-orientador

Prof. Eugênio Libório Feitosa Fortaleza,ENM/UnB _____

Dedicatória

Dedico este trabalho antes de tudo a Deus, pois sem Seu amor eu não seria nada. Também dedico aos meus pais, que foram os principais responsáveis por não somente meu ingresso na Universidade, mas também a continuidade de meus estudos até este momento, sempre me apoiando e incentivando. Aos meus tios Elias e Juliana que foram tão importante para o começo da faculdade. Por fim, também dedico ao meu irmão e a Juliana, minha namorada, pessoas que amo e estiveram sempre ao meu lado.

Diogo Camargos Gomes

Agradecimentos

A Deus, que se mostra presente em todas as coisas da minha vida e me traz paz. A minha família,a quem amo muito e que sempre me deu suporte de forma que eu pudesse me dedicar ao estudo e fizeram todo o esforço necessário para que eu tivesse um estudo de qualidade. A Juliana Medeiros, minha namorada, que foi compreensiva com os finais de semana de estudo e que nunca deixou de me motivar a estudar. Ao professor Guilherme Caribé, que teve toda paciência comigo, sempre esteve disposto a me ajudar e que também me ensinou muito não só neste trabalho como eu todas as outras disciplinas em que tive oportunidade de estudar com ele. A todos professores com quem estudei durante o curso, que me instruíram muito e são responsáveis pelo profissional que serei a partir de agora. Aos meus colegas de cursos, que me apoiaram e ajudaram de forma mútua. Principalmente ao meu amigo Ivan de Souza, que foi meu parceiro do começo ao fim da faculdade.

Diogo Camargos Gomes

RESUMO

Robôs manipuladores são uma ferramenta usada hoje em dia em diversos segmentos de automação, seja para manipulação de objetos, processos de soldagem, ou qualquer outro tipo de trabalho que seja insalubre para um operador humano ou que exija uma precisão maior. O controle de trajetória de seu end-effector exige métodos de cálculo de sua estrutura cinemática. Quando um manipulador é redundante, ou seja, que possui um número de eixos maior que o número de variáveis que definem a posição de seu end-effector, estes cálculos se tornam mais complexos e então surgem diversos métodos para sua realização. Este trabalho realiza a modelagem de um manipulador de 7 graus de liberdade, além de apresentar dois métodos de controle cinemático, realizar suas simulações e implementar o controle em um dispositivo *FPGA*.

ABSTRACT

Manipulators are a tool that has been used in several Automation segments, like objects manipulation, welding, or others kinds of unhealthy works for the man or that requires greater precision. The end-effector path control requires calculation methods of the kinematics structure. When a manipulator is redudant, in others words, that has more axes than the number of variables that define the end-effector position, the calculation becomes more complex and than new control methods arise. This work creates the kinematics modeling of a 7-DOF manipulator, shows two kinematics control method, simulate both methods and implements it in a FPGA to control the manipulator.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

p	Variável de posição cartesiana
T_i^j	Matriz de transformação de i a j
q	Variável de junta
$\dot{p}$	Velocidade linear
J	Matriz jacobiana
J^+	Pseudo-inversa da matriz jacobiana
J^T	Matriz jacobiana transposta
H	Função de otimização de redundância
c_n	Cosseno da junta n
s_n	Seno da junta n
R	Matriz rotacional

Símbolos Gregos

(ϕ, θ, ψ)	Ângulos de Euler
μ	Índice de manipulabilidade
ω	Velocidade angular
θ_n	Ângulo de junta n

Siglas

ISO	*International Organization for Standardization*
FPGA	*Field-Programable Gate Array*
RPY	*Roll, Pitch and Yaw*
CLIK	*Closed-loop Inverse Kinematics*
RISC	*Reduced Instruction Set Computer*
IDE	*Integrated Development Environment*
LED	*Light Emitting Diode*
JTAG	*Joint Test Action Group*
UART	*Universal Asynchronous Receiver/Transmitter*
SDRAM	*Synchronous dynamic random access memory*
GUI	*Graphical User Interface*

Capítulo 1

Introdução

1.1 Contextualização

A automação aplica técnicas computadorizadas ou mecânicas para diminuir o uso de mão-de-obra. Cada vez mais, a automação de atividades tem exercido um papel fundamental na produção industrial e na prestação de serviços. Geralmente busca-se utilizar a automação de atividades que são repetitivas, que são realizadas em situação insalubre, ou que exigem um alto grau de precisão ou velocidade.

Uma ferramenta que tem um papel muito importante na automação industrial é o robô. De acordo com a norma *ISO* (*International Organization for Standardization*) 10218 o 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".

1.2 Definição do problema

Um robô industrial é formado pela integração de atuadores, sensores, estrutura mecânica, unidade de controle, unidade de potência e ferramenta[3]. A unidade de controle, que é o foco deste projeto, é responsável pelo gerenciamento e monitoramento dos parâmetros operacionais requeridos para realizar as tarefas do robô. Com base nas dimensões do robô e na leitura de seus sensores, os controladores devem planejar movimentos que seguem uma determinada trajetória.

1.3 Objetivos do projeto

O cerne deste projeto é a implementação do controle de movimento de um robô redundante de 7 graus de liberdade através de um sistema embarcado *FPGA* (*Field-programable gate array*). O *FPGA* é um dispositivo semicondutor no qual a sua programação é em nível de *hardware*, podendo haver um microprocessador integrado.

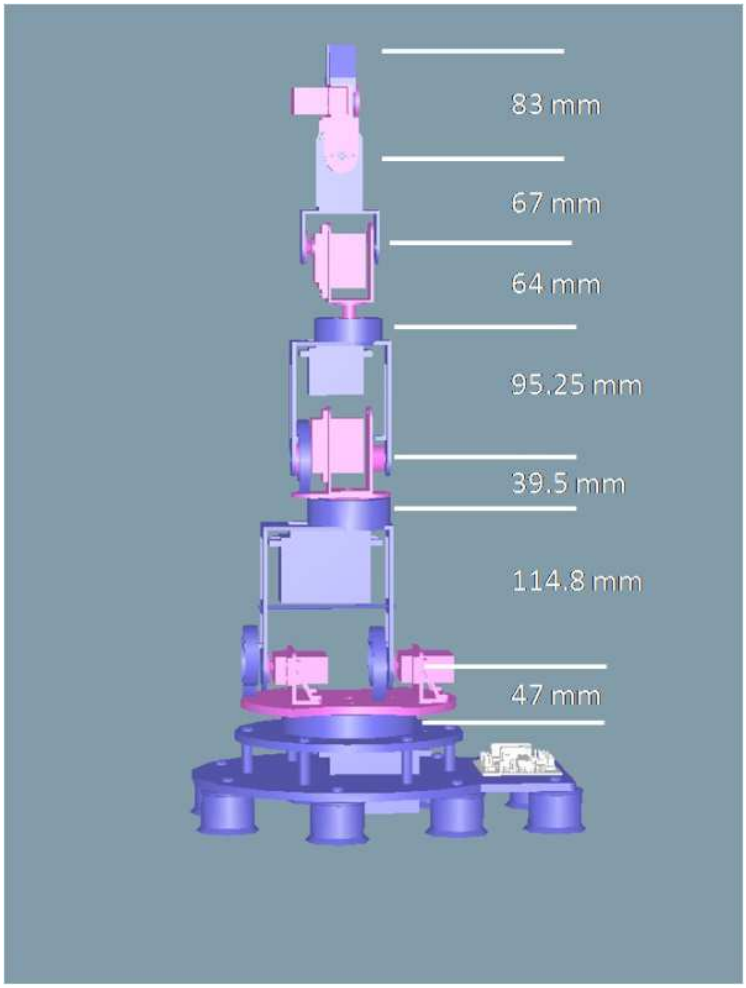


Figura 1.1: Manipulador Cyton I

O modelo do robô utilizado é o manipulador *Cyton I* da *Energid* cuja estrutura é apresentada na figura 1.1. Ele é acionado através de servomotores e o controle de posição dos motores é realizado a partir do controlador *SSC-32* da *Lynx Motion*.

O controle de movimento do manipulador será implementado em um kit de desenvolvimento para *FPGAs* da *Terasic Inc.* Esse kit utiliza um *FPGA Cyclone IV E* da *Altera Corp.*.

1.4 Organização do trabalho

Este trabalho é dividido em 5 capítulos, sendo que o primeiro aborda a introdução ao tema do trabalho, além de seu objetivo. O segundo capítulo concentra o conteúdo teórico necessário para a realização do trabalho baseando em literaturas de diversos autores. Já o terceiro capítulo apresenta todo o desenvolvimento do trabalho realizado com base nas referências bibliográficas. Enquanto que o quarto capítulo discute os resultados obtidos do capítulo anterior. Por fim, no quarto capítulo, tem-se uma conclusão e recomendações para futuros trabalhos.

Capítulo 2

Revisão Bibliográfica

2.1 Robôs manipuladores

Manipuladores são constituídos por atuadores, sensores, unidade de controle, unidade de potência, ferramenta e estrutura mecânica. Em relação à sua estrutura mecânica, ela consiste na combinação de elementos estruturais rígidos (corpos ou elos) conectados entre si através de articulações (juntas)[2].

As juntas são essencialmente de dois grandes tipos:

- As prismáticas (P), onde o movimento relativo dos elos é linear;
- As rotacionais (R), onde o movimento relativo dos elos é rotacional;
- Existe ainda um terceiro tipo de junta designada por esférica (S) que é uma combinação de três juntas rotacionais com o mesmo ponto de rotação.

2.2 Posição e orientação de um corpo rígido

Na análise do movimento de um manipulador, deve-se levar em conta alguns atributos espaciais do sistema do manipulador. São estes atributos a posição e a orientação.

Uma vez estabelecido um sistema de coordenadas, pode-se localizar qualquer ponto no universo com um vetor de posição $3x1$ (p_x, p_y, p_z), onde cada coordenada representa a projeção do ponto em cada um dos eixos da coordenada de referência. Sendo assim, a trajetória de uma partícula no espaço pode ser representado pela curva $\mathbf{p}(t) = (p_x(t), p_y(t), p_z(t))$.

Para um corpo no espaço, além da posição também deve-se definir sua orientação em relação ao sistemas de coordenadas de referência. A orientação de um corpo pode ser definida através de uma matriz de rotação $3x3$, onde cada coluna representa a projeção de um eixo do novo sistema de coordenadas em relação aos eixos do sistema base [2]. Logo, adota-se uma matriz $4x4$ para representação de posição e orientação em relação ao um sistema de coordenadas, a matriz de

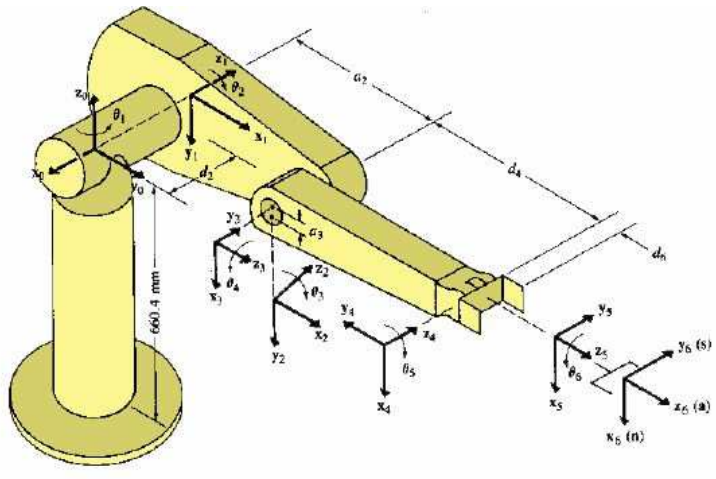


Figura 2.1: Exemplo de sistemas de coordenadas em um manipulador [1]

transformação homogênea. Ela contém a matriz de rotação e o vetor de translação[2].

Em manipuladores, convenciona-se considerar a base de cada junta como um novo sistemas de coordenadas, sendo que o sistema de coordenadas de cada junta pode ser referenciado a partir do sistema de coordenadas anterior. Um exemplo é apresentado na figura 2.1.

No caso da orientação de um corpo rígido, existem outras representações de mais fácil compreensão e de realização cálculos. Uma delas é a representação pelos ângulos RPY (*Roll, Pitch e Yaw*) [4]. Os ângulos RPY são capazes de descrever a orientação de um corpo através de três variáveis, que são os ângulos de giro independentes realizados em relação ao sistema inercial: um giro de ângulo ϕ no eixo x, um giro com ângulo θ no eixo y e um giro de ângulo ψ no eixo z. Sendo assim, é possível representar os ângulos RPY a partir de matriz de rotação gerada das três rotações, como mostra a equação 2.1.

$$\boldsymbol{T_{euler}} = \boldsymbol{T_\phi T_\theta T_\psi} = \begin{pmatrix} cos(\theta) & 0 & -sen(\theta) \\ 0 & 1 & 0 \\ sen(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} cos(\psi) & sen(\psi) & 0 \\ -sen(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sen(\phi) \\ 0 & -sen(\phi) & cos(\phi) \end{pmatrix} \tag{2.1}$$

2.3 Notação de Denavit-Hartenberg

Como já foi dito anteriormente, um manipulador é composto por uma cadeia de corpos rígidos, que são conectados através das juntas. Esta cadeia de elos pode ser caracterizada pelo seu grau de mobilidade, que é equivalente à quantidade de elos da cadeia e cada grau de mobilidade está relacionado a uma variável de junta, que é o valor do ângulo da junta no caso rotacional ou valor do deslocamento da junta no caso prismático. Para se calcular a posição e orientação de cada elo em relação ao anterior (cinemática do manipulador), convenciona-se o uso da notação de Denavit-Hartenberg. Os parâmetros de Denavit-Hartenberg permitem obter o conjunto de equações que

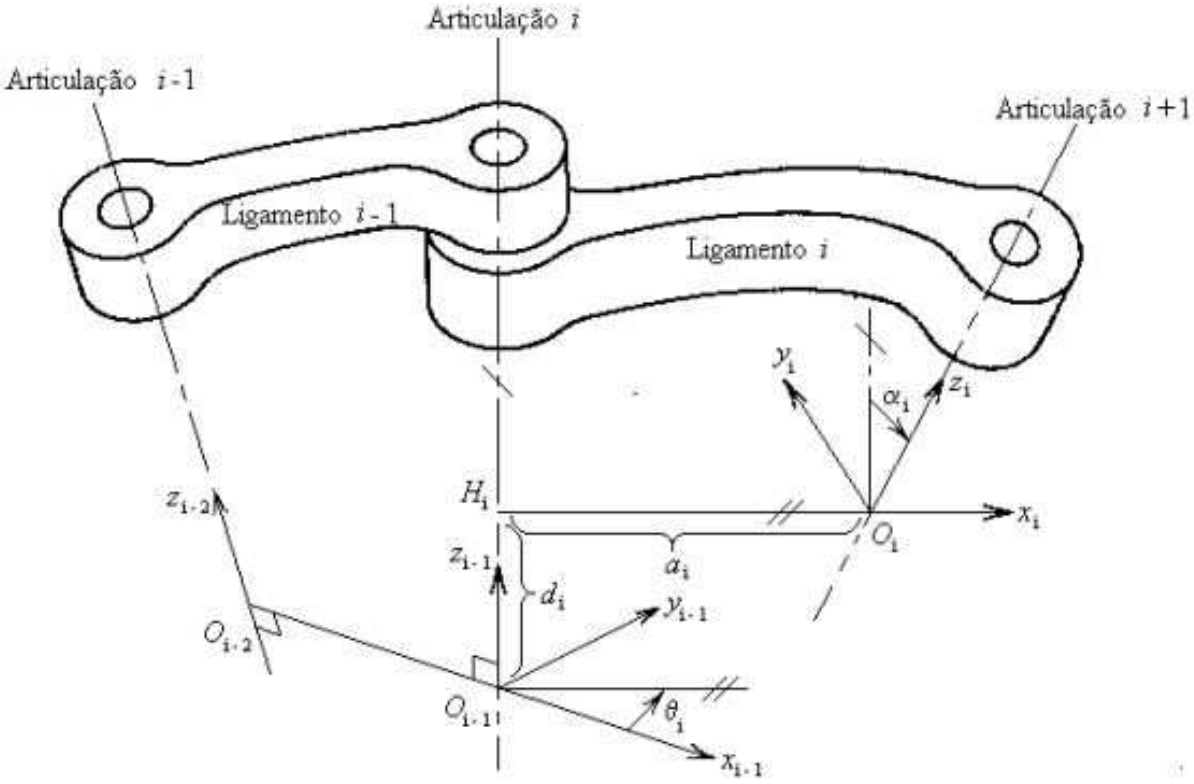


Figura 2.2: Parâmetros de Denavit-Hartenberg [2]

descreve a cinemática de uma junta com relação à junta seguinte e vice-versa. O primeiro parâmetro é a distância medida ao longo da normal comum entre as duas retas e o segundo é o ângulo de rotação em torno da normal comum,que uma das retas deve girar, de forma que fique paralela à outra. Observa-se que a normal comum entre duas retas no espaço é definida por uma terceira reta que intercepta as duas primeiras retas, com ângulos de 90°. Além disso, a distância medida entre as duas retas, ao longo da normal comum, é a menor distância entre as mesmas. Sendo assim, a notação segue a seguinte regra apresentada abaixo[1]. A figura 2.2 exibe um exemplo de elo e seus parâmetros.

- a_i (ou l_i) é o módulo da distância entre z_{i-1} e z_i ao longo do eixo x_i. Na figura 2.2 é a distância H_iO_i;
- α_i é o ângulo entre os eixos z_{i-1} e z_i, em torno de x_i.
- d_i é distância entre os eixos x_{i-1} e x_i, ao longo do eixo z_{i-1}. O sinal de d_i depende do sentido de z_{i-};
- θ_i é o ângulo entre os eixos x_{i-1} e x_i, em torno de z_{i-1}. O seu sinal depende do sentido de rotação

e a matriz de transformação que representa o elo:

$$\boldsymbol{T_i} = \begin{pmatrix} c\theta_i & -s\theta_1 c\alpha_i & s\theta_1 s\alpha_i & l_i c\theta_i \\ s\theta_i & c\theta_1 c\alpha_i & -c\theta_1 s\alpha_i & l_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

2.4 Estudo cinemático de manipuladores

A cinemática é a ciência do movimento sem a análise das forças que o causam. Através dela é possível estudar posição, velocidade e aceleração de corpos rígidos. Em uma cadeia cinemática aberta (que é o caso de um manipulador), cada junta conecta dois elos, então se considera a relação cinemática entre eles, e de maneira recursiva, até atingir a descrição espacial do end-effector em relação ao sistema de coordenadas de referência. Logo, obtem-se a descrição espacial do end-effector em relação ao sistema de referência a partir da multiplicação das matrizes de transformação homogênea entre elos consecutivos [1].

$$T_0^n(q) = T_0^1(q_1)T_1^2(q_2)T_2^3(q_3)...T_{n-1}^n(q_n) \tag{2.3}$$

onde $T_{x-1}^x(q_x)$ é a matriz de transformação homogênea do elo x em relação ao elo $x-1$ e q_x é a variável da respectiva junta.

O problema fundamental no estudo da cinemática de manipuladores é a cinemática direta. É o método para se encontrar a posição e orientação do end-effector a partir das variáveis de juntas do manipulador. Podemos considerar esta situação como uma função onde se transforma a representação da posição do robô no espaço das juntas para o espaço cartesiano, como mostra a figura 2.3. A transformação contrária é chamada cinemática inversa, e exige cálculos mais complexos. A cinemática direta pode ser representada pela equação 2.4

$$\mathbf{p} = f(\mathbf{q}) \tag{2.4}$$

onde $\mathbf{p}$ é o vetor posição e orientação da garra do robô, enquanto que q representa o vetor contendo as variáveis de junta. De forma análoga, a cinemática indireta é representada pela equação 2.5

$$\mathbf{q} = f^{-1}(\mathbf{p}) \tag{2.5}$$

O cálculo da cinemática direta não é uma tarefa complexa, uma vez que suas equações são encontradas na própria matriz de transformação da garra em relação ao sistema euclidiano de referência. Já o problema da cinemática inversa encontra uma maior complexidade devido a alguns motivos:

- as equações não são lineares, e por isso nem sempre é possível encontrar uma solução na forma fechada;

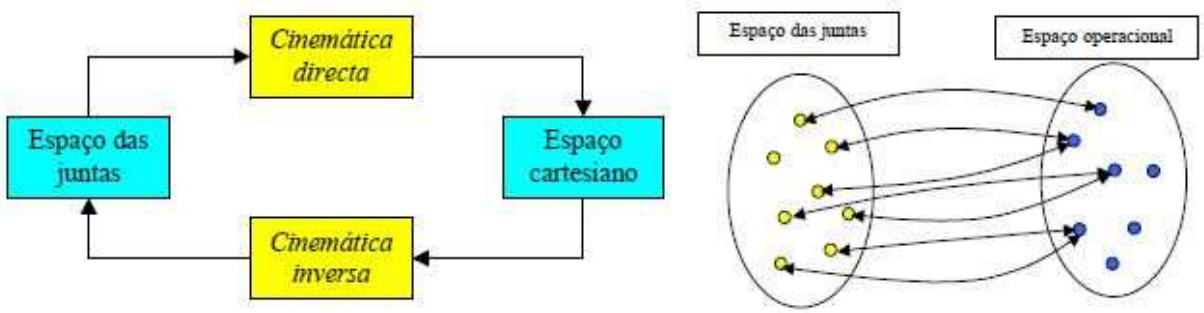


Figura 2.3: Cinemática direta e inversa[1]

- podem existir múltiplas ou infinitas soluções (como no caso de robôs redundantes);
- podem existir soluções que não sejam admissíveis de acordo com a estrutura cinemática do robô.

Existem dois tipos de soluções tradicionais para o cálculo da cinemática inversa. Primeiramente, as soluções analítica, que se baseia em identidades geométricas e algébricas, sendo dessa forma ideal para robôs com cadeia cinemática mais simples, uma vez que quanto maior a quantidade de eixos, maior a complexidade do cálculo. Por outro lado, também existe as soluções numéricas, que se baseiam no uso de métodos iterativos para resolver a equação.

2.5 Jacobiano

As cinemáticas direta e inversa abordam as relações de posição, entretanto elas não envolvem movimentações temporais. Para a resolução deste problema, trabalha-se com uma matriz de transformação diferencial chamada jacobiano[2].

A matriz jacobiana nada mais é que a relação entre as derivadas das variáveis cartesianas e as derivadas das variáveis de junta. Ela estabelece a relação entre velocidade no espaço cartesiano e velocidades no espaço das juntas. O jacobiano é um das principais ferramentas usadas na implementação do controle de um manipulador. Através da sua matriz inversa é possível determinar o movimento diferencial das variáveis de junta tal que execute um determinado movimento cartesiano. Também é possível identificar posições singulares, analisar redundância e mapear forças e torques realizadas nas juntas.

$$\begin{pmatrix} \dot{\boldsymbol{p}} \\ \boldsymbol{\omega} \end{pmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.6}$$

O jacobiano é uma matriz formada pelas derivadas parciais de primeira ordem de uma função vetorial. Nada mais é que uma forma multidimensional de derivada. A sua determinação exige um esforço computacional maior. Pode ser obtido analiticamente por diferenciação da cinemática direta e também a partir de cálculo vetorial.

2.5.1 Cálculo da matriz jacobiana

O jacobiano é um operador linear, sendo assim possível se utilizar do princípio da superposição para a obtenção da velocidade da garra a partir da velocidade das garras. Baseando neste princípio, pode-se encontrar a velocidade angular a partir da equação 2.7 [2]

$$\omega_n^0 = \omega_1^0 + Rot_1^0 * \omega_2^1 + Rot_2^0 * \omega_3^2 + ... + Rot_{n-1}^0 * \omega_n^{n-1} \tag{2.7}$$

sendo $\omega_i^{i-1} = \dot{q}_i$.

Para a velocidade linear, tem-se [2]

$$V_i^{i-1} = V_{i-1}^0 + \omega_i^{i-1} \times P_i^{i-1} \tag{2.8}$$

Multiplicando ambos lados da equação 2.8 por Rot_{i-1}^0, obtem-se

$$V_i^0 = R_i^{i-1}(V_{i-1}^0 + \omega_i^{i-1} \times P_i^{i-1}) \tag{2.9}$$

Uma ferramenta importante no cálculo de trajetórias do robô é a matriz inversa do jacobiano. Contudo, nem sempre o jacobiano é uma matriz quadrada, um requesito para o cálculo de sua inversa. Com isso, desenvolve-se um método de cálculo da pseudo-inversa de uma matriz, chamada como matriz pseudo-inversa de *Moore-Penrose* [2].

$$J^+ = (J^T J)^{-1} J^T \tag{2.10}$$

2.5.2 Análise da matriz jacobiana

Se para uma determinada configuração do manipulador a matriz J não é singular, então J^+ existe e é única. Já que a matriz J depende do vetor q, é possível que, em determinadas configurações, a matriz jacobiana seja singular. Nesses casos, a pseudo-inversa do jacobiano não existe. As configurações onde isso ocorre são chamadas de *configurações singulares*. Nesta configuração, os vetores coluna do jacobiano são linearmente dependentes, não sendo possível operar sobre todo o vetor $\dot{r}$, ou seja, existem direções onde o end-effector não é capaz de se movimentar. Pontos singulares são de grande interesse na movimentação do manipulador, pois nestes pontos a mobilidade é reduzida, ou há geração de grandes velocidades de juntas para pequenas velocidades cartesianas ou então há múltiplas soluções para o problema da cinemática inversa.

2.6 Robôs redundantes

A capacidade de posicionamento geral no espaço requer somente 6 graus de mobilidade, mas existem vantagens em ter mais juntas controláveis. Um robô redundante é um manipulador capaz

de apresentar mais de uma configuração para uma determinada posição de seu end-effector. Para que isto seja possível, é necessário que o manipulador possua mais graus de mobilidade que o número de variáveis que definem uma posição. Como a posição do end-effector é definida a partir de 6 variáveis(posição e orientação nos eixos x, y e z), um robô com 7 ou mais graus de mobilidade é considerado redundante, no caso de posicionamento em um espaço tridimensional.[5]

A redundância de manipuladores possui um papel importante no desenvolvimento de sua flexibilidade e versatilidade. Por exemplo, tal característica pode ser usada para evitar pontos de singularidades, para diminuir o troque em juntas e para desviar de osbtáculos. Contudo, para tirar total proveito da sua redundância, outras análises devem ser realizadas e algoritmos de controle efetivos devem ser desenvolvidos aumentando consideravelmente a complexidade dos cálculos. Mesmo apresentando vantagens em relação a robôs não-redundantes no que se refere às configurações singulares, estes robôs apresentam um número maior de casos em que a estrutura possa apresentar singularidades. Nestes casos, observa-se que o determinante do produto $J * J^T$ é nulo.

Analisando o determinante mencionado acima, observa-se que quanto menor é o seu valor, mais próximo o manipulador está de uma configuração singular. Com base nesta observação, *Yoshikawa* propôs o índice de manipulabilidade do robô [6]:

$$\mu(q) = \sqrt{det(J * J^T)} \tag{2.11}$$

2.6.1 Espaço nulo de robôs redundantes

Se $\boldsymbol{A}$ é uma matriz e $\boldsymbol{A} * \boldsymbol{x} = 0$, tem-se que o espaço nulo da matriz $\boldsymbol{A}$ consiste no espaço gerado a partir dos vetores soluções da equação. Para o jacobiano, o espaço nulo refere-se a movimentos gerados no espaço das juntas que não causam alteração no plano cartesiano. Somente robôs redundantes, onde o jacobiano possui um número de colunas maior que de linhas, apresentam tal propriedade.

Com base nisso, é possível gerar movimentos nas juntas para atender a determinada restrição no movimento sem que seja alterada a trajetória realizada. [7] apresenta o método de projeção do gradiente que se baseia na projeção no espaço nulo da matriz jacobiana.

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}(\boldsymbol{q})\dot{\boldsymbol{x}} + [\boldsymbol{I} - \boldsymbol{J}^{+}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})]\dot{\boldsymbol{q}}_{\boldsymbol{0}} \tag{2.12}$$

onde $[\boldsymbol{I} - \boldsymbol{J}^{+}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})]$ é o operador de projeção no espaço nulo de $\boldsymbol{J}$ e $\dot{\boldsymbol{q}}_{\boldsymbol{0}}$ é um vetor de velocidade de juntas abritário. A equação 2.12 é uma adaptação da equação 2.6 pela adição do termo homogêneo criado pela projeção de $\dot{\boldsymbol{q}}_{\boldsymbol{0}}$ no espaço nulo do jacobiano, então o vetor $\dot{\boldsymbol{q}}_{\boldsymbol{0}}$ produz movimentos no espaço de juntas e não no espaço cartesiano. No método de projeção do gradiente, escolhe-se uma função custo $h(q)$, no qual

$$\dot{\boldsymbol{q}}_{\boldsymbol{0}} = (\frac{\delta h}{\delta \boldsymbol{q}})^T \tag{2.13}$$

Algumas funções custos serão discutidas no próximo capítulo.

Capítulo 3

Desenvolvimento

3.1 Introdução

Este capítulo é dividido em etapas: modelagem cinemática, implementação e simulação de métodos de geração de trajetórias e implementação em sistema embarcado do método desenvolvido em simulação.

3.2 Matrizes de transformação

Para a obtenção das matrizes de transformação de cada elo e, consequentemente, da ferramenta em relação à base do manipulador, tem-se como base a notação de Denavit-Hartenberg [2].

Na figura 1.1, pode-se observar que todas as juntas são do tipo rotacional, havendo uma diferença de 30° entre eixos z de cada junta. No caso do terceiro elo, por exemplo, que possui comprimento de 39.5 mm, a sua rotação ocorre no seu próprio eixo. Sendo assim, é possível que o elo 2 seja considerado como um elo de comprimento nulo, enquanto que o elo 3 deve possuir um comprimento igual à soma dos elos 2 e 3. A mesma observação ocorre entre os elos 4 e 5. Esta abordagem pode facilitar a modelagem.

A partir do método descrito, tem-se a tabela 3.1 com os parâmetros de Denavit-Hartenberg do *Cyton I*.

A partir dos parâmetros da tabela 3.1, tem-se as matrizes de transformação de todos elos e do manipulador.

$$ {}^{0}T_1 = \begin{pmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 47 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1} $$

Elo	θ	α	**d**	**l**
1	θ_1	$90°$	47	0
2	θ_2	$-90°$	0	0
3	θ_3	$90°$	154.3	0
4	θ_4	$-90°$	0	0
5	θ_5	$90°$	159.25	0
6	$\theta_6 + 90°$	$-90°$	0	67
7	θ_7	$90°$	0	83

Tabela 3.1: Parâmetros de Denavit-Hartenberg para o manipulador

$$
{}^1T_2 = \begin{pmatrix} c_2 & 0 & -s_2 & 0 \\ s_2 & 0 & c_2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}
$$

$$
{}^2T_3 = \begin{pmatrix} c_3 & 0 & s_3 & 0 \\ s_3 & 0 & -c_3 & 0 \\ 0 & 1 & 0 & 154.3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}
$$

$$
{}^3T_4 = \begin{pmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}
$$

$$
{}^4T_5 = \begin{pmatrix} c_5 & 0 & s_5 & 0 \\ s_5 & 0 & -c_5 & 0 \\ 0 & 1 & 0 & 159.25 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5}
$$

$$
{}^5T_6 = \begin{pmatrix} -s_6 & 0 & -c_6 & -67\,s_6 \\ c_6 & 0 & -s_6 & 67\,c_6 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6}
$$

$$
{}^6T_7 = \begin{pmatrix} c_7 & 0 & s_7 & 83\,c_7 \\ s_7 & 0 & -c_7 & 83\,s_7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.7}
$$

$$
\begin{gathered}
[^0T_7]_{11} = c_7(c_6(s_4(s_1s_3 - c_1c_2c_3) - c_1c_4s_2) + s_6(c_5(c_4(s_1s_3 - c_1c_2c_3) + c_1s_2s_4) \\
+s_5(c_3s_1 + c_1c_2s_3))) + s_7(s_5(c_4(s_1s_3 - c_1c_2c_3) + c_1s_2s_4) - c_5(c_3s_1 + c_1c_2s_3)) \\
\\
[^0T_7]_{12} = -s_6(s_4(s_1s_3 - c_1c_2c_3) - c_1c_4s_2) + c_6(c_5(c_4(s_1s_3 - c_1c_2c_3) + c_1s_2s_4) \\
+s_5(c_3s_1 + c_1c_2s_3)) \\
\\
[^0T_7]_{13} = s_7(c_6(s_4(s_1s_3 - c_1c_2c_3) - c_1c_4s_2)s_6(c_5(c_4(s_1s_3 - c_1c_2c_3) + c_1s_2s_4) \\
+s_5(c_3s_1 + c_1c_2s_3))) - c_7(s_5(c_4(s_1s_3 - c_1c_2c_3) + c_1s_2s_4) - c_5(c_3s_1 + c_1c_2s_3)) \\
\\
[^0T_7]_{14} = 67c_6(s_4(s_1s_3 - c_1c_2c_3) - c_1c_4s_2) - (154.3c_1s_2) + 83c_7(c_6(s_4(s_1s_3 - c_1c_2c_3) - c_1c_4s_2) \\
+s_6(c_5(c_4(s_1s_3 - c_1c_2c_3) + c_1s_2s_4) + s_5(c_3s_1 + c_1c_2s_3))) \\
+67s_6(c_5(c_4(s_1s_3 - c_1c_2c_3) + c_1s_2s_4) + s_5(c_3s_1 + c_1c_2s_3)) + 83s_7(s_5(c_4(s_1s_3 - c_1c_2c_3) \\
+c_1s_2s_4) - c_5(c_3s_1 + c_1c_2s_3)) + (159.25s_4(s_1s_3 - c_1c_2c_3)) - (159.25c_1c_4s_2) \\
\\
[^0T_7]_{21} = -s_7(s_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) - c_5(c_1c_3 - c_2s_1s_3)) \\
-c_7(c_6(s_4(c_1s_3 + c_2c_3s_1) + c_4s_1s_2)s_6(c_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) + s_5(c_1c_3 - c_2s_1s_3))) \\
\\
[^0T_7]_{22} = s_6(s_4(c_1s_3 + c_2c_3s_1) + c_4s_1s_2) - c_6(c_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) \\
+s_5(c_1c_3 - c_2s_1s_3)) \\
\\
[^0T_7]_{23} = c_7(s_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) - c_5(c_1c_3 - c_2s_1s_3)) \\
-s_7(c_6(s_4(c_1s_3 + c_2c_3s_1) + c_4s_1s_2)(c_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) + s_5(c_1c_3 - c_2s_1s_3))) \\
\\
[^0T_7]_{24} = -67s_6(c_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) + s_5(c_1c_3 - c_2s_1s_3)) \\
-67c_6(s_4(c_1s_3 + c_2c_3s_1) + c_4s_1s_2) - (154.3s_1s_2) - 83s_7(s_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) \\
-c_5(c_1c_3 - c_2s_1s_3)) - (159.25s_4(c_1s_3 + c_2c_3s_1)) - 83c_7(c_6(s_4(c_1s_3 + c_2c_3s_1) + c_4s_1s_2) \\
+s_6(c_5(c_4(c_1s_3 + c_2c_3s_1) - s_1s_2s_4) + s_5(c_1c_3 - c_2s_1s_3))) - (159.25c_4s_1s_2) \\
\\
[^0T_7]_{31} = c_7(-s_6(c_5(c_2s_4 + c_3c_4s_2) - s_2s_3s_5) + c_6(c_2c_4 - c_3s_2s_4)) \\
-s_7(s_5(c_2s_4 + c_3c_4s_2) + c_5s_2s_3) \\
\\
[^0T_7]_{32} = -s_6(c_2c_4 - c_3s_2s_4) - c_6(c_5(c_2s_4 + c_3c_4s_2) - s_2s_3s_5) \\
\\
[^0T_7]_{33} = c_7(s_5(c_2s_4 + c_3c_4s_2) + c_5s_2s_3) + s_7(-s_6(c_5(c_2s_4 + c_3c_4s_2) - s_2s_3s_5) \\
+c_6(c_2c_4 - c_3s_2s_4)) \\
\\
[^0T_7]_{34} = (154.3c_2) + (159.25c_2c_4) - 67s_6(c_5(c_2s_4 + c_3c_4s_2) - s_2s_3s_5) \\
-83s_7(s_5(c_2s_4 + c_3c_4s_2) + c_5s_2s_3) + 83c_7(-s_6(c_5(c_2s_4 + c_3c_4s_2) - s_2s_3s_5) \\
+c_6(c_2c_4 - c_3s_2s_4)) + 67c_6(c_2c_4 - c_3s_2s_4) - (159.25c_3s_2s_4) + 47 \\
\\
[^0T_7]_{41} = 0 \\
[^0T_7]_{42} = 0 \\
[^0T_7]_{43} = 0 \\
[^0T_7]_{44} = 1
\end{gathered}
\tag{3.8}
$$

Considerando uma matriz de transformação constante para a ferramenta que será usada no manipulador.

$$
{}^{7}T_{tool} = \begin{pmatrix} n_1 & p_1 & r_1 & k_1 \\ n_2 & p_2 & r_2 & k_2 \\ n_3 & p_3 & r_3 & k_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.9}
$$

Por fim, encontra-se a seguinte matriz do manipulador com a ferramenta:

$$
{}^{0}T_{tool} = \begin{pmatrix} \sum_{j=1}^{3} [{}^{0}T_7]_{1j}\, n_j & \sum_{j=1}^{3} [{}^{0}T_7]_{1j}\, p_j & \sum_{j=1}^{3} [{}^{0}T_7]_{1j}\, r_j & \sum_{j=1}^{3} [{}^{0}T_7]_{1j}\, k_j + [{}^{0}T_7]_{14} \\ \sum_{j=1}^{3} [{}^{0}T_7]_{2j}\, n_j & \sum_{j=1}^{3} [{}^{0}T_7]_{2j}\, p_j & \sum_{j=1}^{3} [{}^{0}T_7]_{2j}\, r_j & \sum_{j=1}^{3} [{}^{0}T_7]_{2j}\, k_j + [{}^{0}T_7]_{24} \\ \sum_{j=1}^{3} [{}^{0}T_7]_{3j}\, n_j & \sum_{j=1}^{3} [{}^{0}T_7]_{3j}\, p_j & \sum_{j=1}^{3} [{}^{0}T_7]_{3j}\, r_j & \sum_{j=1}^{3} [{}^{0}T_7]_{3j}\, k_j + [{}^{0}T_7]_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.10}
$$

Com a matriz ${}^{0}T_{tool}$ em mãos é possível encontrar a cinemática direta, inversa e a matriz jacobiana do manipulador.

3.3 Cinemática direta

Para apresentação da orientação da ferramenta do robô, vamos usar o método conhecido como *RPY (Roll Pitch Yaw)*. A matriz equivalente na convenção *RPY* é apresentado na equação 3.11 [2].

$$
{}^{0}T_{tool} = \begin{pmatrix} c_\phi\, c_\theta & -s_\phi\, c_\psi + c_\phi\, s_\theta\, s_\psi & s_\phi\, s_\psi + c_\phi\, s_\theta\, c_\psi & p_x \\ s_\phi\, c_\theta & c_\phi\, c_\psi + s_\phi\, s_\theta\, s_\psi & -c_\phi\, s_\psi + s_\phi\, s_\theta\, c_\psi & p_y \\ -s_\theta & c_\theta\, s_\psi & c_\theta\, c_\psi & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.11}
$$

Comparando ambas matrizes, podemos obter os valores da posição da ponta da ferramenta e também sua orientação *RPY*.

$$
p_x = \sum_{j=1}^{3} [{}^{0}T_7]_{1j}\, k_j + [{}^{0}T_7]_{14} \tag{3.12}
$$

$$
p_y = \sum_{j=1}^{3} [{}^{0}T_7]_{2j}\, k_j + [{}^{0}T_7]_{24} \tag{3.13}
$$

$$
p_z = \sum_{j=1}^{3} [{}^{0}T_7]_{3j}\, k_j + [{}^{0}T_7]_{34} \tag{3.14}
$$

No caso da orientação, poderíamos fazer os cálculos a partir de arco-seno ou arco-cosseno. Porém, nestes casos poderia haver imprecisão do cálculo para ângulos próximos de 0° ou 90°. Uma forma de resolver este problema, é realizar o cálculo do arco tangente considerando o ângulo do resultado. A função que realiza este cálculo é representada por *atan2(y,x)*[8].

$$atan2(y,x) = \begin{cases} \alpha\, sgn(y) & x > 0 \\ \frac{\pi}{2}\, sgn(y) & x = 0 \\ (\pi - \alpha)\, sgn(y) & x < 0 \end{cases} \tag{3.15}$$

Desta forma, podemos obter a orientação da ferramenta de uma forma eficaz.

$$\phi = atan2\left(\sum_{j=1}^{3}[^0T_7]_{2j}\, n_j, \sum_{j=1}^{3}[^0T_7]_{1j}\, n_j\right) \tag{3.16}$$

$$\theta = atan2\left(-\sum_{j=1}^{3}[^0T_7]_{3j}\, n_j, \sqrt{(\sum_{j=1}^{3}[^0T_7]_{1j}\, n_j)^2 + (\sum_{j=1}^{3}[^0T_7]_{2j}\, n_j)^2}\right) \tag{3.17}$$

$$\psi = atan2\left(\sum_{j=1}^{3}[^0T_7]_{3j}\, p_j, \sum_{j=1}^{3}[^0T_7]_{3j}\, r_j\right) \tag{3.18}$$

3.4 Matriz jacobiana

Para este trabalho, dentre diversos métodos de cálculo da matriz jacobiana, adotou-se o método de Whitney [9]. Como o jacobiano é um operador linear, pode-se utilizar superposição para obter a velocidade da garra em função das velocidades das juntas. Assim, a parcela da velocidade cartesiana da garra em relação ao sistema de coordenadas da base, representada no sistema de coordenadas da base, devido à velocidade da junta i é dada por

$$^0V_n^i = \begin{cases} ^0R_{i-1}(^{i-1}Z_{i-1} \times^{i-1} P_n)\dot{q}_i & \text{para junta rotacional} \\ ^0R_{i-1}^{i-1}Z_{i-1}\dot{q}_i & \text{para junta prismática} \end{cases} \tag{3.19}$$

onde $^0R_{i-1}$ é a matriz rotacional da junta $i-1$ em relação à base, ^iZ_i é o vetor unitário apontando na direção z do sistema de coordenadas i e ^iP_n é a posição da junta i.
Analogamente, a parcela de velocidade angular da garra em relação ao sistema de coordenadas da base, representada no sistema de coordenadas da base, devido à velocidade da junta i é dada por

$$
{}^0\omega_n^i = \begin{cases} {}^0R_{i-1}^{i-1} Z_{i-1} \dot{q}_i & \text{para junta rotacional} \\ 0 & \text{para junta prismática} \end{cases} \tag{3.20}
$$

A *i-ésima* coluna do jacobiano representado no sistema de coordenadas de base será dada por

$$
J(q) = \begin{pmatrix} {}^0V_n^1 & {}^0V_n^2 \ldots {}^0V_n^n \\ {}^0\omega_n^1 & {}^0\omega_n^2 \ldots {}^0\omega_n^n \end{pmatrix} \tag{3.21}
$$

3.5 Jacobiano inverso

A matriz jacobiana tem dimensões 6x7, logo não é inversível. Devido a este fato, deve-ser calculado a matriz pseudo-inversa de *Moore-Penrose* do jacobiano, que é dado pela seguinte equação.

$$
J^+ = (J^T J)^{-1} J^T \tag{3.22}
$$

3.6 Cinemática inversa

Métodos analíticos do problema de cinemática inversa encontram soluções exatas através da inversão das equações de cinemática inversa, entretanto são aplicáveis somente a problemas mais simples. No problema presente, não é possível a adoção de métodos analíticos, sendo então necessário adotar soluções numéricas, que utilizam aproximações e diversas iterações para tentar convergir para a solução. Apesar de serem mais genéricas, as soluções numéricas são computacionalmente mais custosas.

3.6.1 Cinemática inversa por malha fechada

Devido à complexidade do cálculo, torna-se uma opção atraente o uso do algoritmo de cinemática inversa por malha fechada (*CLIK - Closed-loop inverse kinematics*) [10]. O algoritmo *CLIK* se baseia no erro entre a posição desejada e atual da garra ou então o erro da velocidade. Como o robô do projeto não possui controle a nível de aceleração, este trabalho se atentará somente ao erro de posição.

3.7 Método da projeção do gradiente

O *CLIK* calcula a cinemática inversa, sem tirar proveito da redundância do manipulador, gerando movimento. Entretanto, conforme apresentado no capítulo anterior, é possível gerar mo-

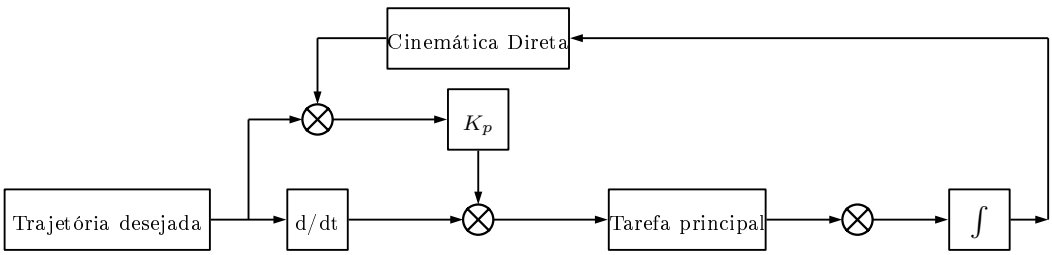


Figura 3.1: Algoritmo em malha fechada da cinemática inversa

vimentos no espaço nulo do jacobiano para que a trajetória atenda a determinadas restrições sem a sua alteração. A equação 3.23 expressa o método.

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}(\boldsymbol{q})\dot{\boldsymbol{x}} + [\boldsymbol{I} - \boldsymbol{J}^{+}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})]\dot{\boldsymbol{q}}_{\boldsymbol{0}} \tag{3.23}$$

onde $\dot{\boldsymbol{q}}_{\boldsymbol{0}}$ é o gradiente de uma função de restrição ($\nabla H(q)$). Durante a execução da trajetória $p(t)$, o robô busca incrementar o valor de $H(q)$. A definição da função $H(q)$ depende restrição aplicada ao movimento.

Umas das funções do método é evitar que o manipulador atinja ponto de singulares. Então, uma boa opção para a função $H(q)$ é o índice de manipulabilidade [6].

$$H_{manip}(q) = \sqrt{det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{q})} \tag{3.24}$$

Dessa forma, o movimento do manipulador tende a configurações que possuem os maiores valores do índice de manipulabilidade possível. Esta função seria a mais adequada para implementação neste trabalho, entretanto o cálculo da equação de manipulabilidade do manipulador de 7 graus de mobilidade exige um processamento desproporcional às ferramentas disponíveis para tal.

Sendo assim, outra opção de implementação é usar o alcance disponível de cada junta como critério de otimização da resolução de redundância. Pode-se considerar a função $H(q)$ como [6]

$$H_{junta}(q) = \frac{1}{2n}\sum_{i=1}^{n}(\frac{q_i - q_{ic}}{q_{iM} - q_{im}})^2 \tag{3.25}$$

Onde q_{ic}, q_{iM} e q_{im} são respectivamente a posição central, o alcance máximo e o alcance mínimo de uma junta i. No caso deste manipulador, tem-se $q_{ic} = 0°$, $q_{iM} = 90°$ e $q_{im} = -90°$. Logo tem-se como resultado

$$H_{junta}(q) = \frac{1}{14}\sum_{i=1}^{n}(q_i)^2 \tag{3.26}$$

O gradiente da função apresentado na equação 3.26 resulta no vetor

$$\dot{\boldsymbol{q}}_{\mathbf{0}} = \frac{1}{7} * \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \end{pmatrix} \tag{3.27}$$

Nas simulações, foi usado o vetor acima multiplicado por -1, para que o robô tenda a possuir o menor valor da soma dos ângulos possíveis. Os resultados serão discutidos no próximo capítulo.

3.8 Simulações

Para as simulações do algoritmo de controle cinemático do manipulador, foi adotado o seguinte método: um programa em C que realiza os cálculos do controle cinemático e armazena os dados em um arquivo e um programa em *Matlab* que lê os dados gerados pelo outro programa e simula a trajetória do manipulador em ambiente 3D, além de apresentar gráficos relativos à trajetória.

As simulações foram divididas em algumas etapas, mas todas elas seguem o mesmo algoritmo, baseado no método de integração de *Euler*. O método de Euler é um procedimento numérico para aproximar a solução de uma equação diferencial. Por definição, temos que a derivada $\dot{x}$ é dada por

$$\dot{x} = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t} \tag{3.28}$$

Se $x_n = x(t)$, $x_{n+1} = x(t + h)$ e admitindo que h é suficientemente pequeno, obtem-se a aproximação

$$\dot{x} = \frac{x_{n+1} - x_n}{\Delta t} \tag{3.29}$$

Esta aproximação é necessária no caso da simulação, uma vez que não é possível enviar parâmetros de velocidade ao programa de simulação no *Matlab*.

$$J^{-1} = \frac{\Delta q}{\Delta p} \tag{3.30}$$

$$q_{n+1} = q_n + J^{-1} \Delta p \tag{3.31}$$

Para o cálculos de matrizes, tais como multiplicação entre matrizes e pseudo-inversão de matrizes, adotou-se a biblioteca *GMatrix* versão 1.0 desenvolvida pelo professor Geovany A. Borges [11].

Devido ao fato da grande quantidade de variáveis a serem manipuladas e também a quantidade de funções, optou-se pelo uso de variáveis globais. São elas:

- **q**: matriz com as variáveis de junta atuais do robô;
- **dq**: matriz com a diferencial das variáveis de junta;
- **p**: matriz com as variáveis cartesianas da posição atual do robô;
- **dp**: matriz com a diferencial das variáveis cartesianas;
- **MT**: matriz de transformação do robô;
- **J**: matriz jacobiana;
- **Jinv**: pseudo-inversa da matriz jacobiana;
- **pInit**: matriz com a posição inicial do robô em variáveis cartesianas;
- **pTraj**: matriz com a próxima posição em variáveis cartesianas de acordo com a trajetória.
- **veloc-max**: constante usada para determinar velocidade linear máxima;
- **velocX, velocY, velocZ**: variáveis usadas para definir velocidade do robô nos três eixos no caso da movimentação linear;
- **p1, p2, p3...**: matrizes de posição em variáveis cartesianas para memória de posições.

Como dito anteriormente, o código em C gera um arquivo contendo uma lista com as posições na variável de junta do robô para uma determinada trajetória, além de também gravar a trajetória ideal. O programa segue o seguinte algoritmo:

- Inicializar o robô numa posição não-singular;
- Receber a posição alvo e definir o tipo de trajetória;
- Calcular cinemática direta para encontrar posição atual;
- Gerar trajetória a partir da posição atual e posição final (definir velocidades cartesianas); A partir deste ponto, o programa entra em repetição:
- Definir próximo ponto da trajetória de acordo com a velocidade cartesiana determinada e o tempo de execução de cada ciclo.
- Ler posição atual;
- Calcular da nova matriz de transformação;
- Calcular o dp de acordo com a posição atual;
- Calcular do jacobiano para a posição atual;
- Inverter do jacobiano;
- Calcular do dq a partir do dp e do jacobiano invertido;

- Atualizar da posição do robô;
- Verificar se alvo foi atingido;
- Sair do *loop* se alvo foi atingido;
- Ler tempo de execução do ciclo;
- Repetição do ciclo.
- Leitura do tempo do ciclo.

Já em ambiente *Matlab*, o código de simulação realiza a leitura do arquivo contendo os dados da trajetória e realiza uma animação do movimento do manipulador usando a ferramenta *Robotics Toolbox* [12]. Após a execução da simulação são impressos gráficos apresentando a trajetória desejada e realizada em cada uma das 6 variáveis cartesianas de posição e orientação. O código encontra-se nos anexos.

3.9 Configuração do *Hardware* Reconfigurável - *FPGA*

O *FPGA* é um dispositivo semicondutor que pode ser programado conforme as aplicações do usuário. Partindo deste princípio, esta seção apresenta a configuração de um *FPGA Cyclone IV* do kit *Altera DE2-115*. Para o projeto, o processamento será realizado no *Nios II*, que é o processador integrado ao *FPGA*. Sendo assim, deve-ser realizar a configuração e conexão do processador aos dispositivos necessários, tais como memória e elementos de entrada e saída. O *Nios II* é um processador integrado ao *FPGA* com arquitetura RISC em pipeline e de 32 bits. Pode ser configurado através da ferramenta *Qsys* e programado na IDE *Eclipse Nios II*. Basicamente, seu sistema é representado pela figura 3.2.

O kit é apresentado na figura 3.3 com seus diversos recursos. Todo processamento será realizado no processador *Nios II*, entretanto deve-se configurar sua conexão com outros dispositivos da placa para que seja possível seu funcionamento.

Os dispositivos necessários para o uso do *FPGA* são: a comunicação USB com o computador para o envio do programa de controle do robô,o dispositivo de memória para armazenamento do programa e dos dados, a porta RS-232 para a comunicação com o controlador do robô, os dispositivos de entrada e saída de dados (switches, botões e LEDS) para interface com o usuário e por fim o clock do processador. Para configuração do processador, a *Altera* possui a ferramenta *Qsys* na qual podemos instanciar e projetar os dispositivos de *hardware* usados no projeto. A figura 3.4 apresenta o *hardware* instanciado para este projeto.

[13] O processador é conectado à memória e as inferfaces de entrada e saída através de uma rede de intercomunicação chamada *Avalon swicth fabric*. Esta rede é automaticamente gerada pela ferramenta *Qsys*. A princípio, a memória que seria usada no projeto é a memória *on-chip*. Entretanto seu espaço ($8kB$) não foi suficiente para o armazenamento do *software* de controle do robô, sendo assim optou-se pela memória *SDRAM* presente no kit. A instanciação desta memória

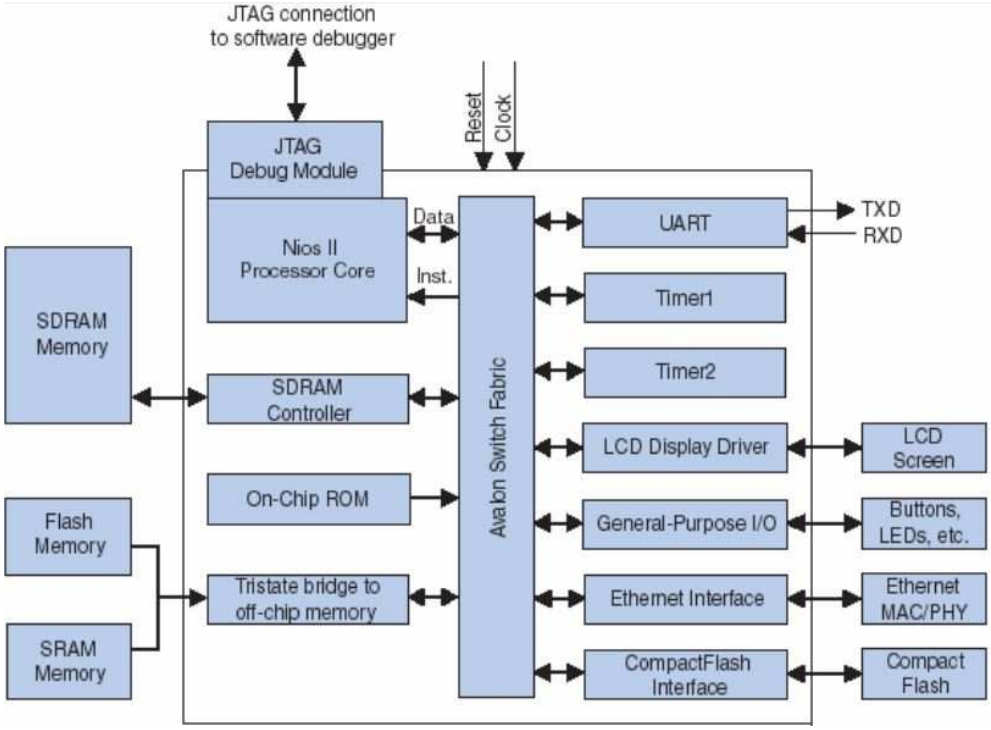


Figura 3.2: Sistema de processamento *Nios II*

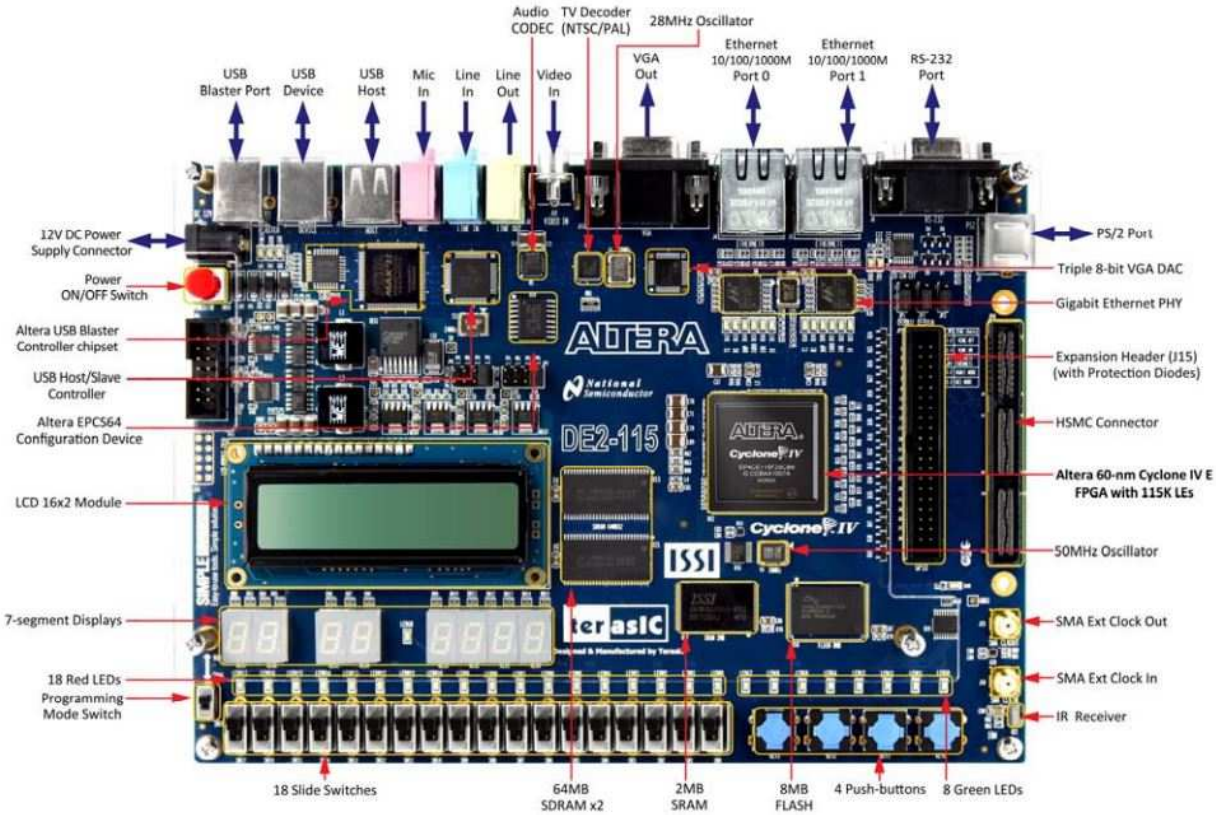
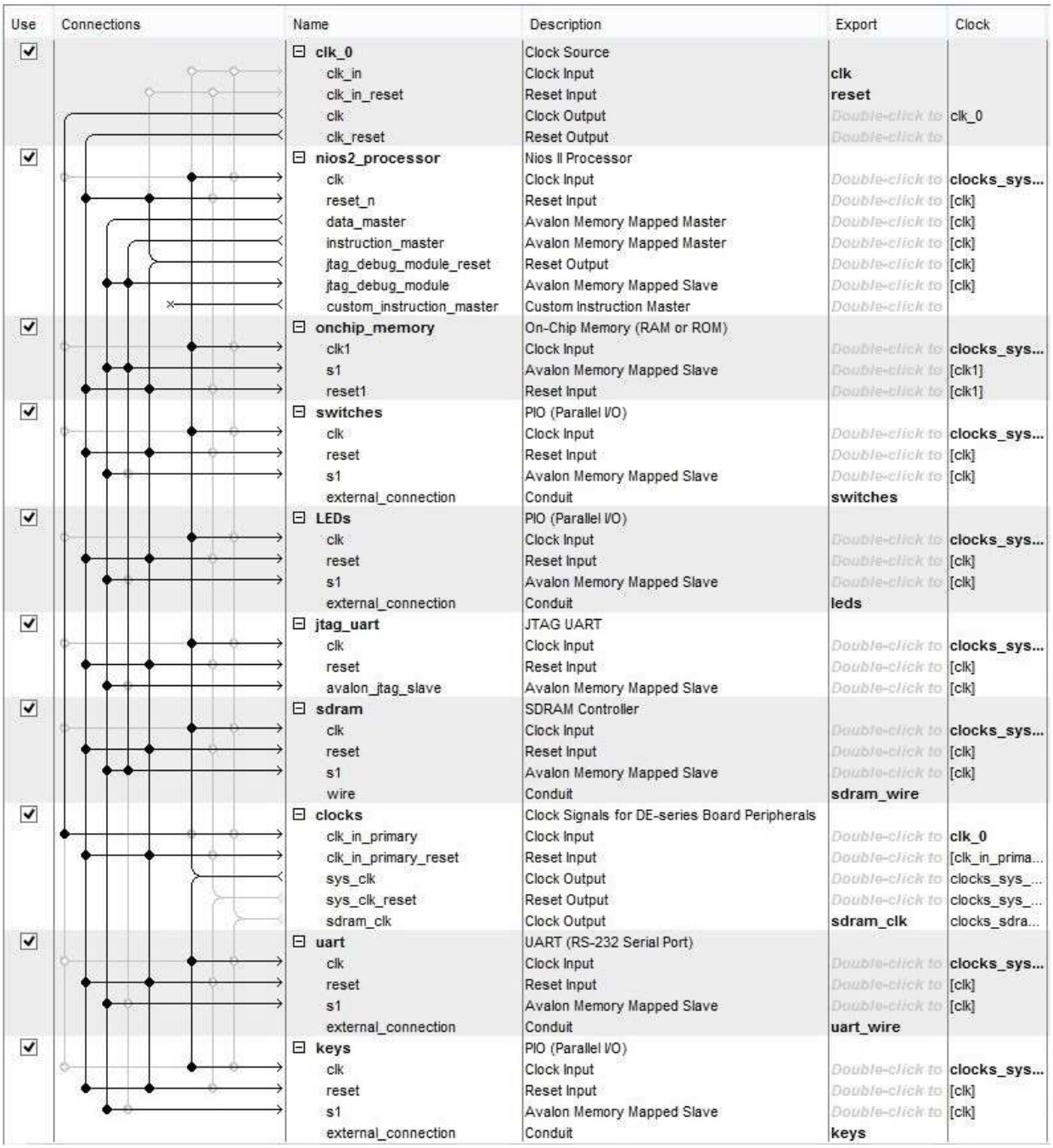


Figura 3.3: Placa DE2-115



Use	Connections	Name	Description	Export	Clock
☑		⊟ **clk_0**	Clock Source		
		clk_in	Clock Input	**clk**	
		clk_in_reset	Reset Input	**reset**	
		clk	Clock Output	Double-click to	clk_0
		clk_reset	Reset Output	Double-click to	
☑		⊟ **nios2_processor**	Nios II Processor		
		clk	Clock Input	Double-click to	**clocks_sys...**
		reset_n	Reset Input	Double-click to	[clk]
		data_master	Avalon Memory Mapped Master	Double-click to	[clk]
		instruction_master	Avalon Memory Mapped Master	Double-click to	[clk]
		jtag_debug_module_reset	Reset Output	Double-click to	[clk]
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click to	[clk]
		custom_instruction_master	Custom Instruction Master	Double-click to	
☑		⊟ **onchip_memory**	On-Chip Memory (RAM or ROM)		
		clk1	Clock Input	Double-click to	**clocks_sys...**
		s1	Avalon Memory Mapped Slave	Double-click to	[clk1]
		reset1	Reset Input	Double-click to	[clk1]
☑		⊟ **switches**	PIO (Parallel I/O)		
		clk	Clock Input	Double-click to	**clocks_sys...**
		reset	Reset Input	Double-click to	[clk]
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]
		external_connection	Conduit	**switches**	
☑		⊟ **LEDs**	PIO (Parallel I/O)		
		clk	Clock Input	Double-click to	**clocks_sys...**
		reset	Reset Input	Double-click to	[clk]
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]
		external_connection	Conduit	**leds**	
☑		⊟ **jtag_uart**	JTAG UART		
		clk	Clock Input	Double-click to	**clocks_sys...**
		reset	Reset Input	Double-click to	[clk]
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]
☑		⊟ **sdram**	SDRAM Controller		
		clk	Clock Input	Double-click to	**clocks_sys...**
		reset	Reset Input	Double-click to	[clk]
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]
		wire	Conduit	**sdram_wire**	
☑		⊟ **clocks**	Clock Signals for DE-series Board Peripherals		
		clk_in_primary	Clock Input	Double-click to	**clk_0**
		clk_in_primary_reset	Reset Input	Double-click to	[clk_in_prima...
		sys_clk	Clock Output	Double-click to	clocks_sys_...
		sys_clk_reset	Reset Output	Double-click to	clocks_sys_...
		sdram_clk	Clock Output	**sdram_clk**	clocks_sdra...
☑		⊟ **uart**	UART (RS-232 Serial Port)		
		clk	Clock Input	Double-click to	**clocks_sys...**
		reset	Reset Input	Double-click to	[clk]
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]
		external_connection	Conduit	**uart_wire**	
☑		⊟ **keys**	PIO (Parallel I/O)		
		clk	Clock Input	Double-click to	**clocks_sys...**
		reset	Reset Input	Double-click to	[clk]
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]
		external_connection	Conduit	**keys**	

Figura 3.4: Instanciação de dispositivos no *Qsys*

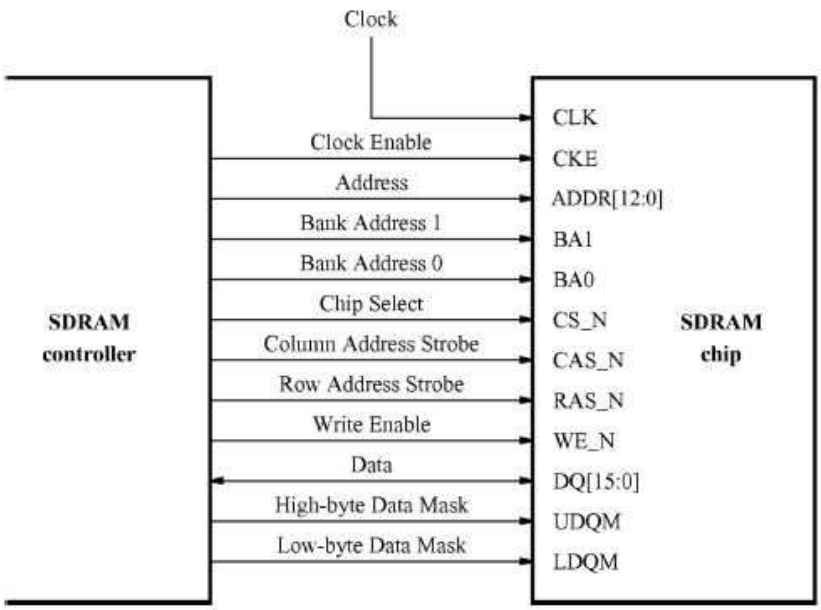


Figura 3.5: Conexões entre a unidade *SDRAM* e seu controlador

será vista mais a frente. As interfaces de e/s que se conectam aos *swicthes*, botões e *LEDs* foram implementados usando-se módulos pré-definidos disponíveis na ferramenta *Qsys*. Uma interface especial *JTAG UART* foi usada para conectar o circuito que fornece a conexão *USB* ao computador ao qual o *FPGA está conectado*. Ele é necessário para se realizar operações tais como baixar programas do *Nios II* para a memória ou iniciar, examinar e parar a execução de tais programas. Para a comunicação entre a placa e o controlador do robô, foi instanciados o módulo *UART* para controlar a comunicação serial via porta *RS-232*.

Como dito anteriormente, a memória interna do *Nios II* é insuficiente para o armazenamento do código implementado, sendo assim optou-se por usar a memória *SDRAM* presente no kit da *Altera*. Para isso deve-se criar um módulo responsável pela comunicação e controle da unidade *SDRAM*. O sinais necessários para se comunicar com o chip *SDRAM* são mostrados na figura 3.5. Todos os sinais, exceto o *clock* provêm do seu controlador e são gerados pela ferramenta *Qsys*. Um ponto a se destacar é que no caso do kit DE2-115, o clock da unidade *SDRAM* deve possuir um atraso de 3 nanosegundos em relação ao clock do CPU. Para que isto seja realizado, pode-se usar a ferramenta *Clock Signal IP* fornecido pelo *Altera University Program*. Ela é um módulo que, a partir do clock do FPGA, gera um clock para o CPU e outro clock para a SDRAM com um atraso de 3 ns.

O próximo passo na ferramenta *Qsys* é atualizar os endereços de memória dos dispositivos, além de configurar o *Nios II* para inicializar a partir da memória SDRAM. Por fim, é necessário realizar todas as conexões entre os módulos e exportar as conexões que serão realizadas com dispositivos externos e então gerar os arquivos em *Verilog* do sistema que serão integrados ao projeto no programa *Quartus*, que configura o *FPGA*. A ferramenta *Qsys* gera um módulo em *Verilog* que define o sistema *Nios II* desejado. O módulo apresenta, como variáveis de e/s, todas as conexões exportadas pela ferramenta *Qsys*. Portanto, no *Quartus* deve-se importar os arquivos gerados pelo

Qsys, que contém o módulo do sistema *Nios II*. Com o módulo incluso no projeto, cria-se um novo módulo principal onde se instancia o módulo *Nios II* e realiza todas conexões do sistema ao pinos do *FPGA*. O código em *Verilog* pode ser observado abaixo.

3.10 Configuração e implementação do *sofware* de controle do robô

[13] O *Nios II SBT* para *Eclipse* é um *GUI* de fácil uso que automatiza o gerenciamento da construção do projeto, além de integrar um editor de texto, um debugador e o programador para *Nios II*. Através dele é possível compilar um código em C para sua execução no sistema *Nios II*. Para iniciar o desenvolvimento do código, deve-se criar um projeto e importar o arquivo (*.sopcinfo*) que contém toda a descrição necessária referente ao sistema criado anteriormente na ferramenta *Qsys*. A partir deste arquivo, o *Nios II SBT* gera o pacote BSP com as bibliotecas referentes ao sistema *hardware*, tais como os *drivers* dos componentes do sistema. Deve-se destacar três importantes arquivos deste pacote:

- **sistem.h**, que é o arquivo que encapsula o sistema *hardware*;
- **alt-sys-init.c**, que é o arquivo de inicialização para os dispositivos do sistema e
- **linker.h**, que é o arquivo que contém informações sobre o *layout* da memória de ligação.

A partir deste pacote e de outras bibliotecas da linguagem C que podem ser adicionadas ao projeto, é possível desenvolver o código que realizará os cálculos do controle cinemático do manipulador.

Capítulo 4

Resultados Experimentais

4.1 Introdução

Este capítulo é dividido em duas grandes seções: a primeira apresenta as diversas simulações realizadas e as suas correções, enquanto que a segunda seção trata dos testes realizados no robô.

4.2 Simulações

As simulações foram realizadas como descrito no capítulo anterior. Contudo, ela foi dividida em algumas partes de forma que fosse possível dividir em etapas a implementação e correção do código em seus diversos pontos. Primeiramente, trabalha-se considerando como variáveis cartesianas somentes as três variáveis de posição (p_x, p_y, p_z). Vale destacar que, no ciclo de execução do código de simulação, adicionou-se um tempo de espera, que representa o tempo gasto na comunicação entre *FPGA* e o controlador do robô.

4.2.1 Comprovação do movimento no espaço nulo

Antes das simulações de trajetórias, simulou-se a movimentação do robô no espaço nulo do jacobiano para sua comprovação. Tendo como critério a soma dos quadrados das variáveis de junta, adotou-se a posição inicial apresentada na tabela 4.1

Posição inicial (variáveis de junta)	$5°, 45°, 15°, 60°, 27°, 18°, 65°$

Tabela 4.1: Posição inicial do robô simulação do movimento no espaço nulo (posições inicial e final)

As figuras 4.1, 4.2, 4.3, 4.4 e 4.5 comprovam a eficiência do método. Existe um pequeno erro na posição cartesiana, que é corrigido com o passar do tempo, enquanto que a configuração do robô é rearranjada de forma que a soma dos quadrados das variáveis de junta seja a menor possível. Vale a pena ressaltar, que apesar de o índice de manipulabilidade aumentar com o decorrer do movimento,

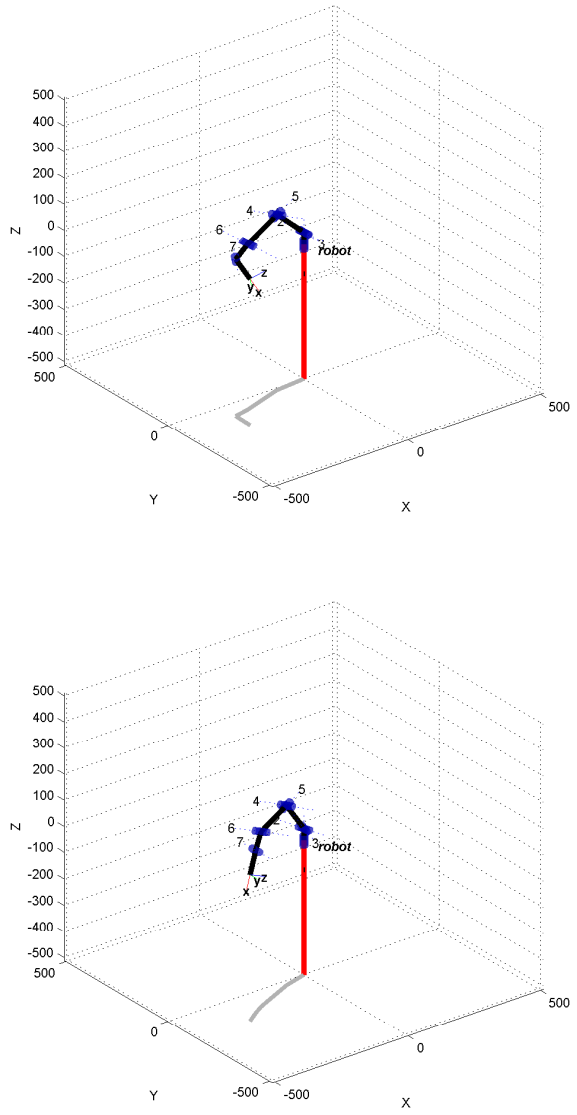


Figura 4.1: Simulação do movimento no espaço nulo

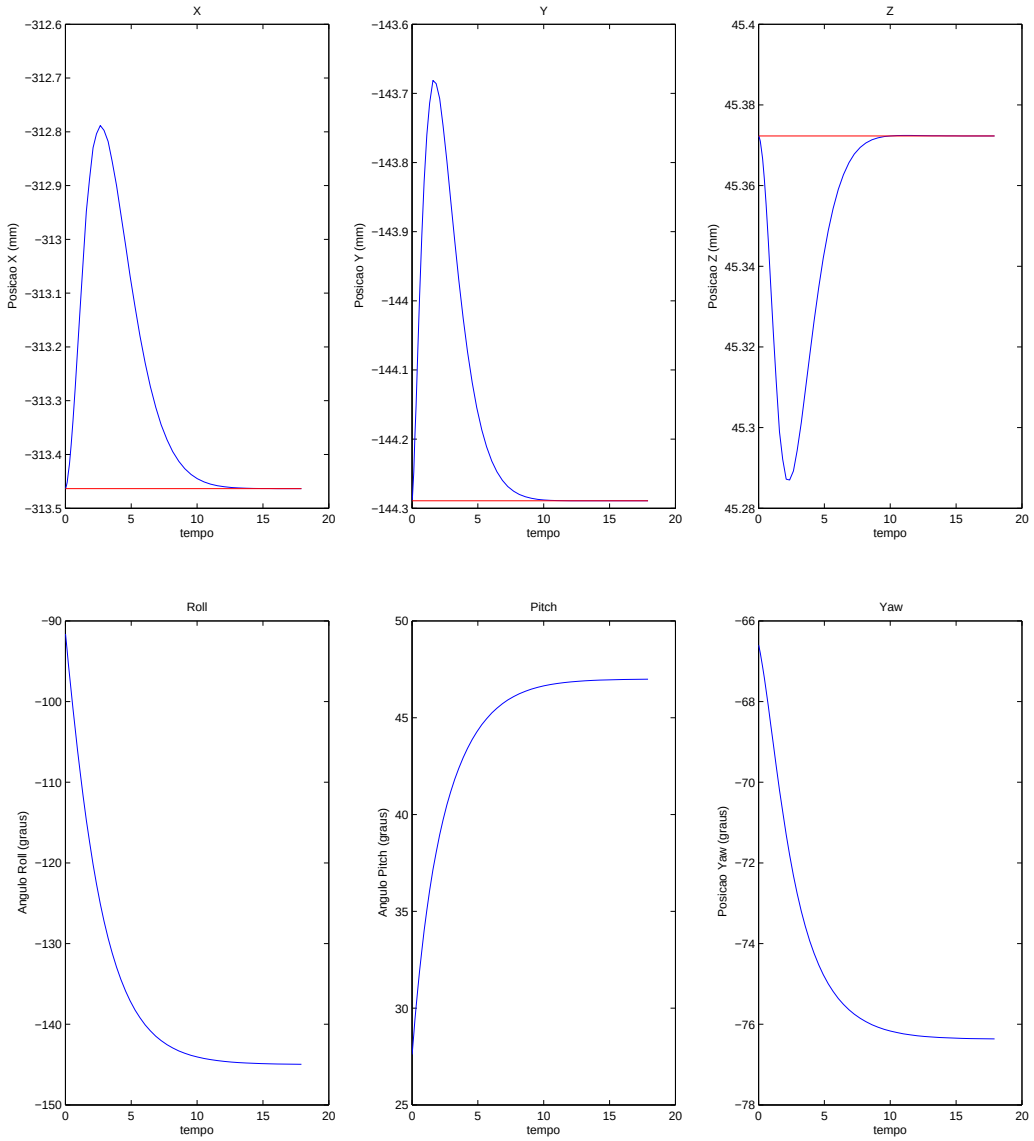


Figura 4.2: Gráficos do movimento no espaço nulo

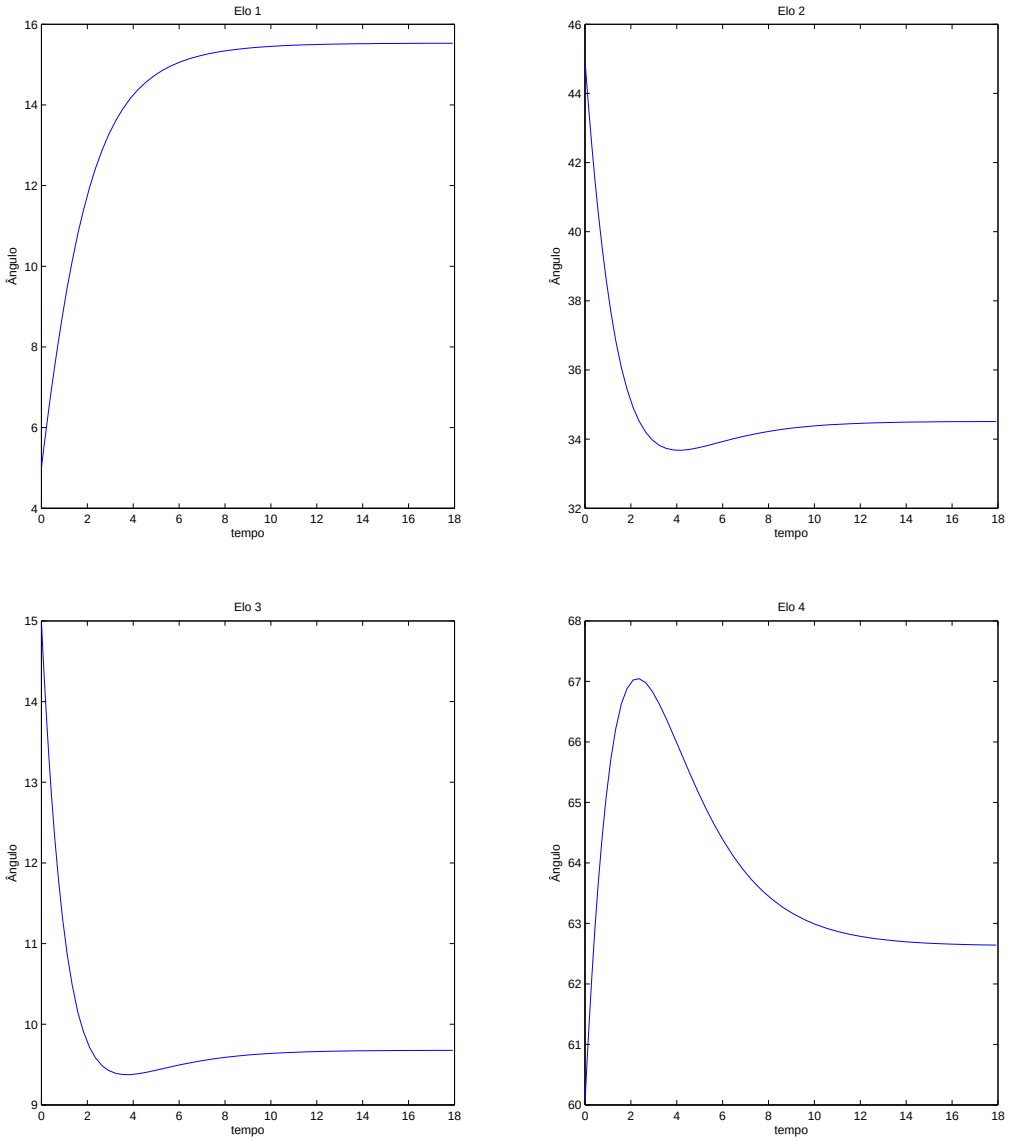


Figura 4.3: Gráficos da movimentação das juntas 1, 2, 3 e 4 no movimento no espaço nulo

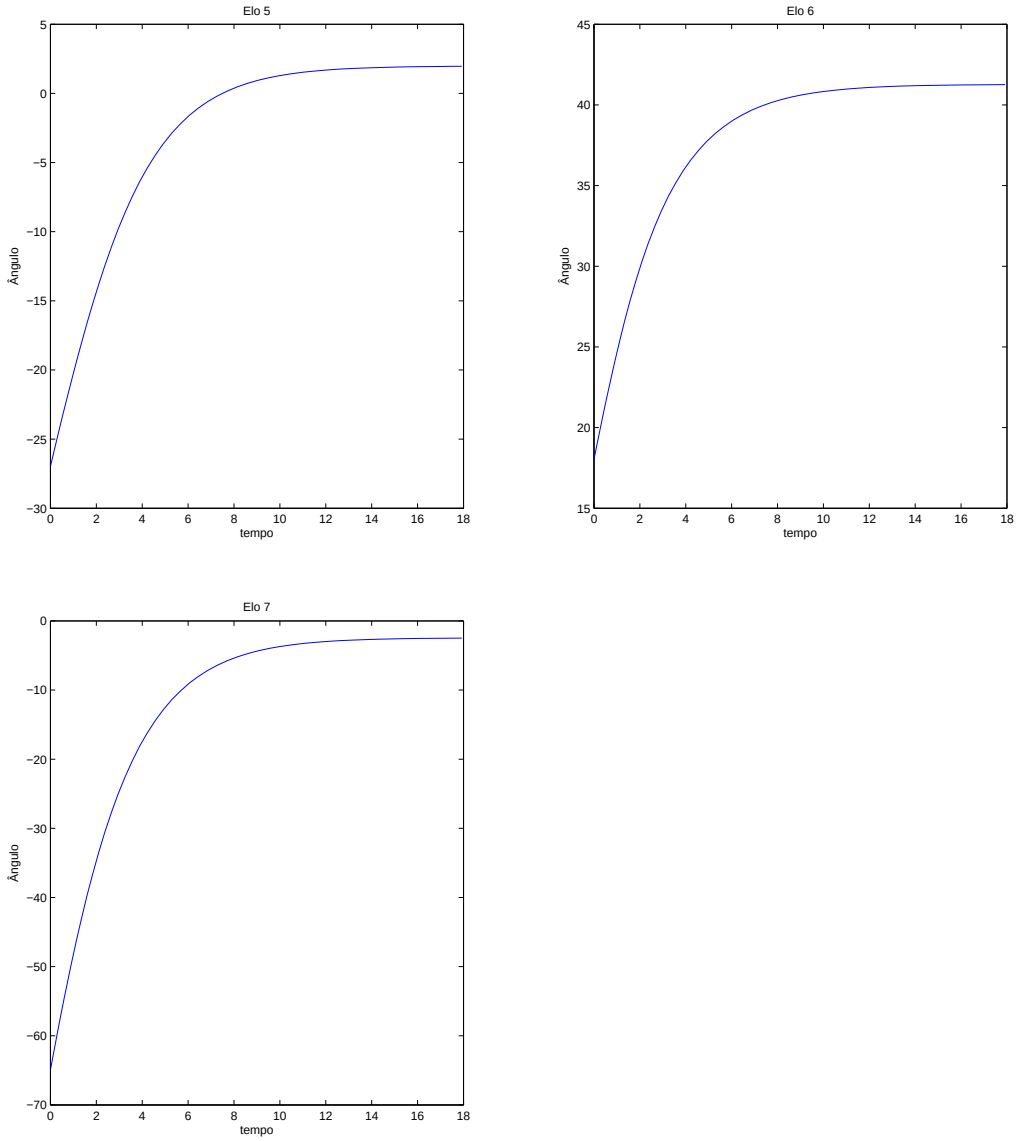


Figura 4.4: Gráficos da movimentação das juntas 5, 6 e 7 no movimento no espaço nulo

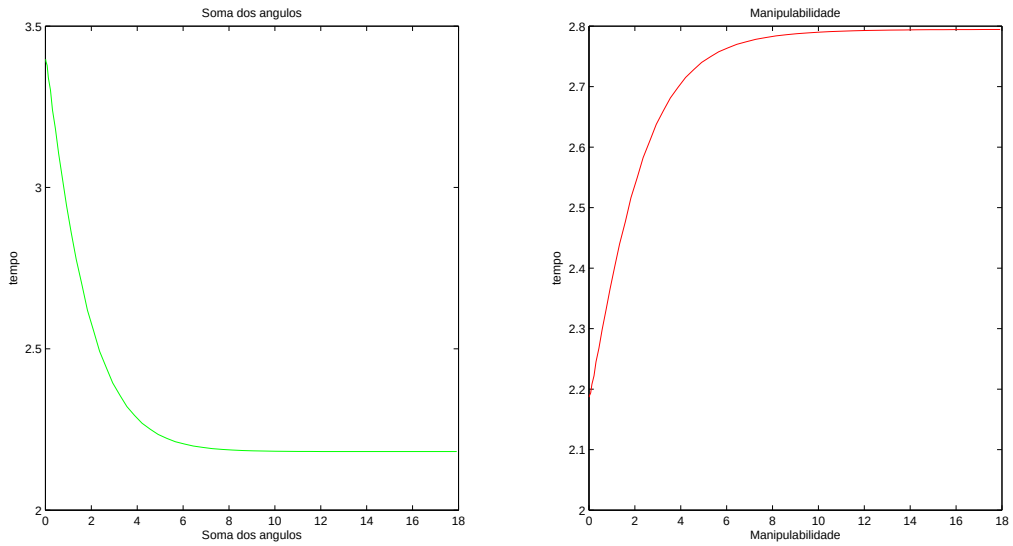


Figura 4.5: Índices de soma de ângulos e de manipulabilidade - espaço nulo

este índice não está relacionado ao critério adotado na simulação. Entretanto, para este trabalho não há ferramentas disponíveis para o cálculo do gradiente do índice de manipulabilidade. Dessa forma, será usado o critério apresentado nesta seção.

4.2.2 Trajetória linear (orientação desprezada)

A primeira simulação foi realizada com os parâmetros apresentados na tabela 4.2. Uma vez com o método da pseudo-inversa tradicional e outra vez com o método de projeção do gradiente.

Posição inicial (variáveis de junta)	$5°, -45°, 5°, -80°, -15°, -30°, 15°$
Posição final (variáveis cartesianas	$x = 280$ $y = 165$ $z = 350$

Tabela 4.2: Parâmetros da primeira simulação

Nas figuras 4.6, 4.7, 4.8, 4.9 e 4.10 pode-se observar que o robô realizou todo o movimento corretamente, seguindo a trajetória definida e sem atingir nenhum ponto de singularidade. Já os gráficos das figuras 4.12 e 4.13 apresenta a variação dos ângulos de cada junta ao decorrer da trajetória. Nas trajetórias retilíneas, o algoritmo de cinemática inversa apresentou resultados satisfatórios.

Pode-se observar na figura 4.12 que nenhum dos elos atingiu o seu limite. Entretanto, nenhum dos algoritmos apresenta nenhum tratamento para evitá-lo. Na figura 4.14 pode-se observar que o índice de manipulabilidade caiu drasticamente nos momentos finais da trajetória. Isto ocorre devido o aumento da proximidade ao limite da zona de trabalho do robô. Na comparação entre ambos métodos, podemos observar que, apesar de gerarem a mesma trajetória no espaço cartesiano, eles percorrem trajetórias diferentes no espaço das juntas. Coincidentemente, o método com projeção do gradiente apresenta um índice de manipulabilidade um pouco maior ao decorrer da trajetória.

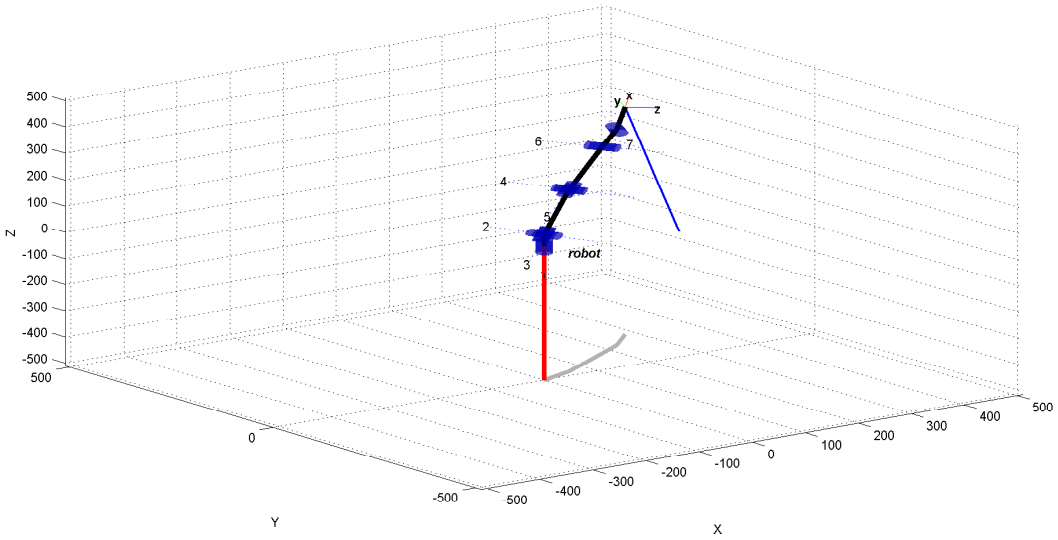


Figura 4.6: Simulação da trajetória linear - sem PG

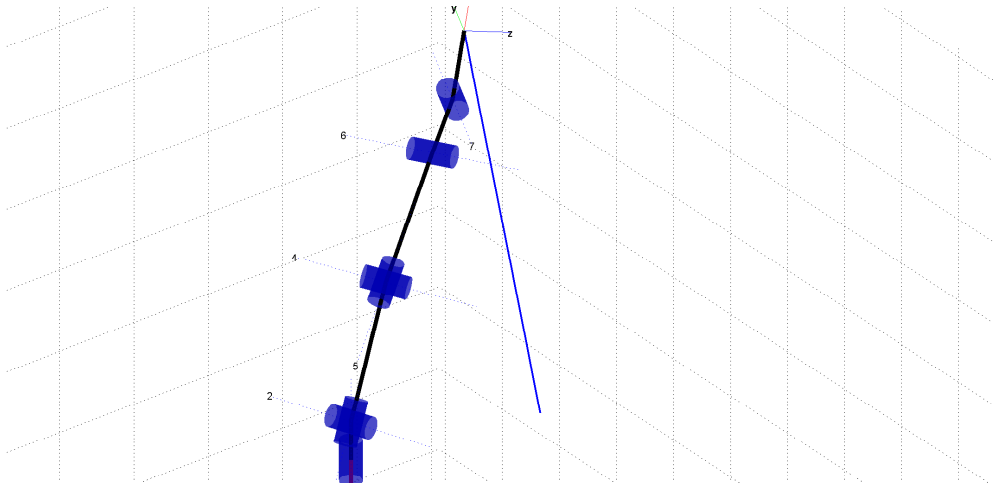


Figura 4.7: Trajetória linear em detalhe - sem PG

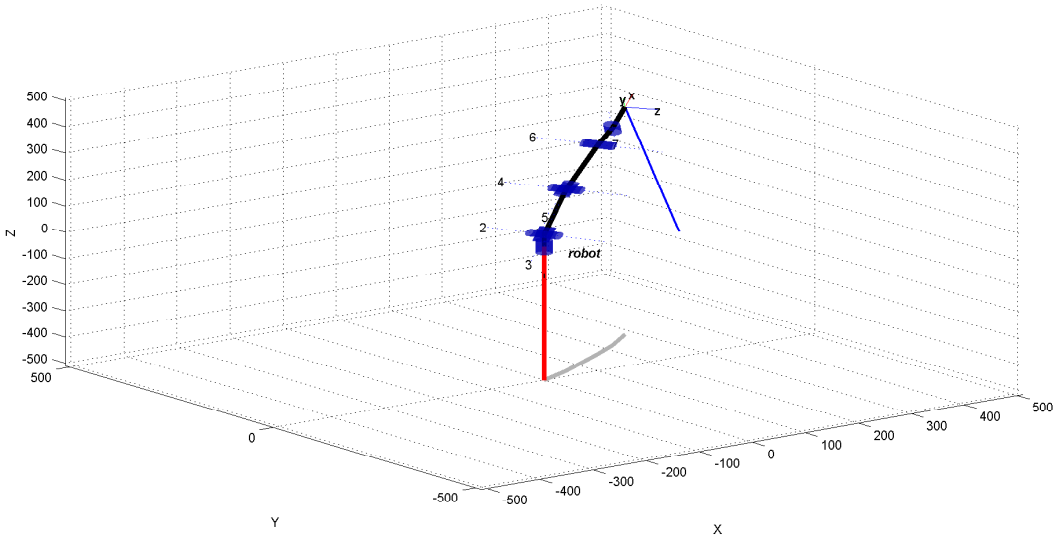


Figura 4.8: Simulação da trajetória linear - com PG

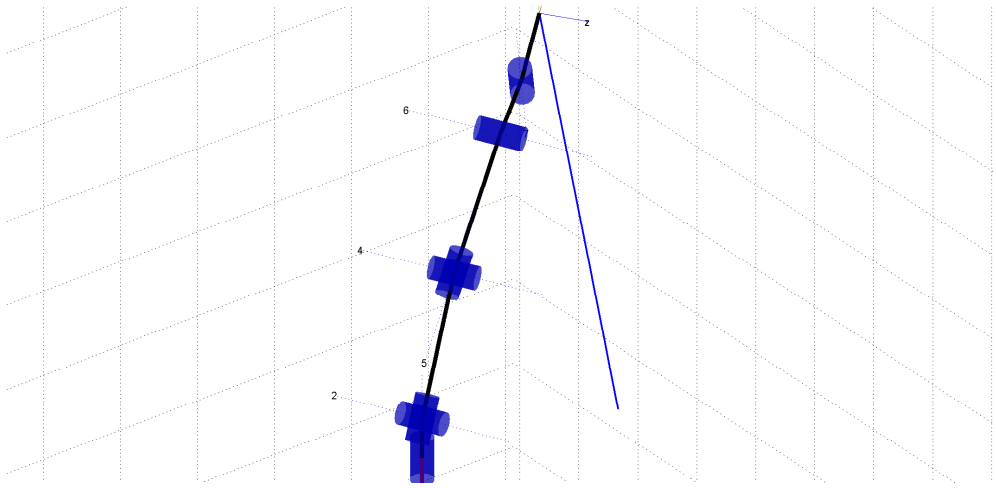


Figura 4.9: Trajetória linear em detalhe - com PG

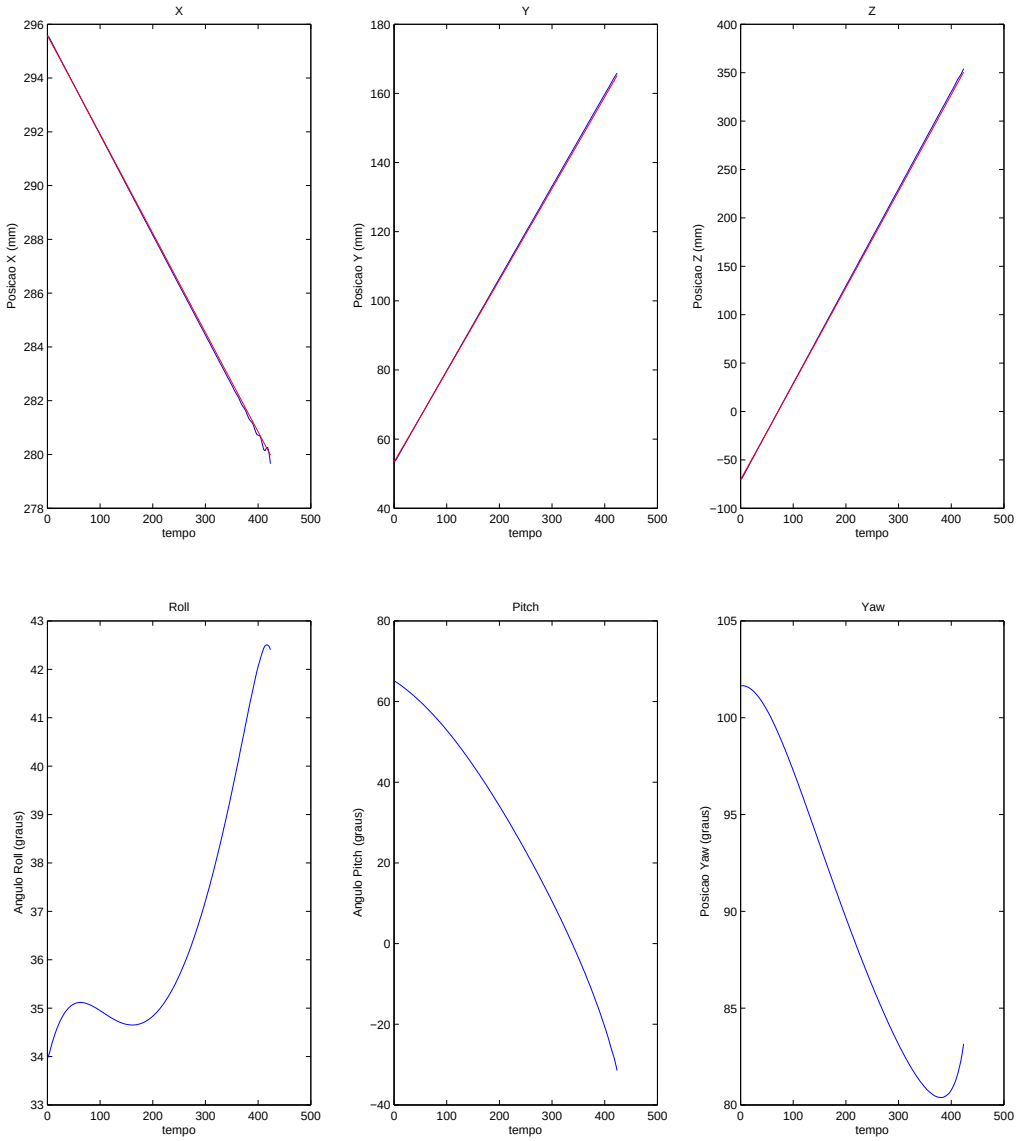


Figura 4.10: Gráficos da trajetória linear - sem PG

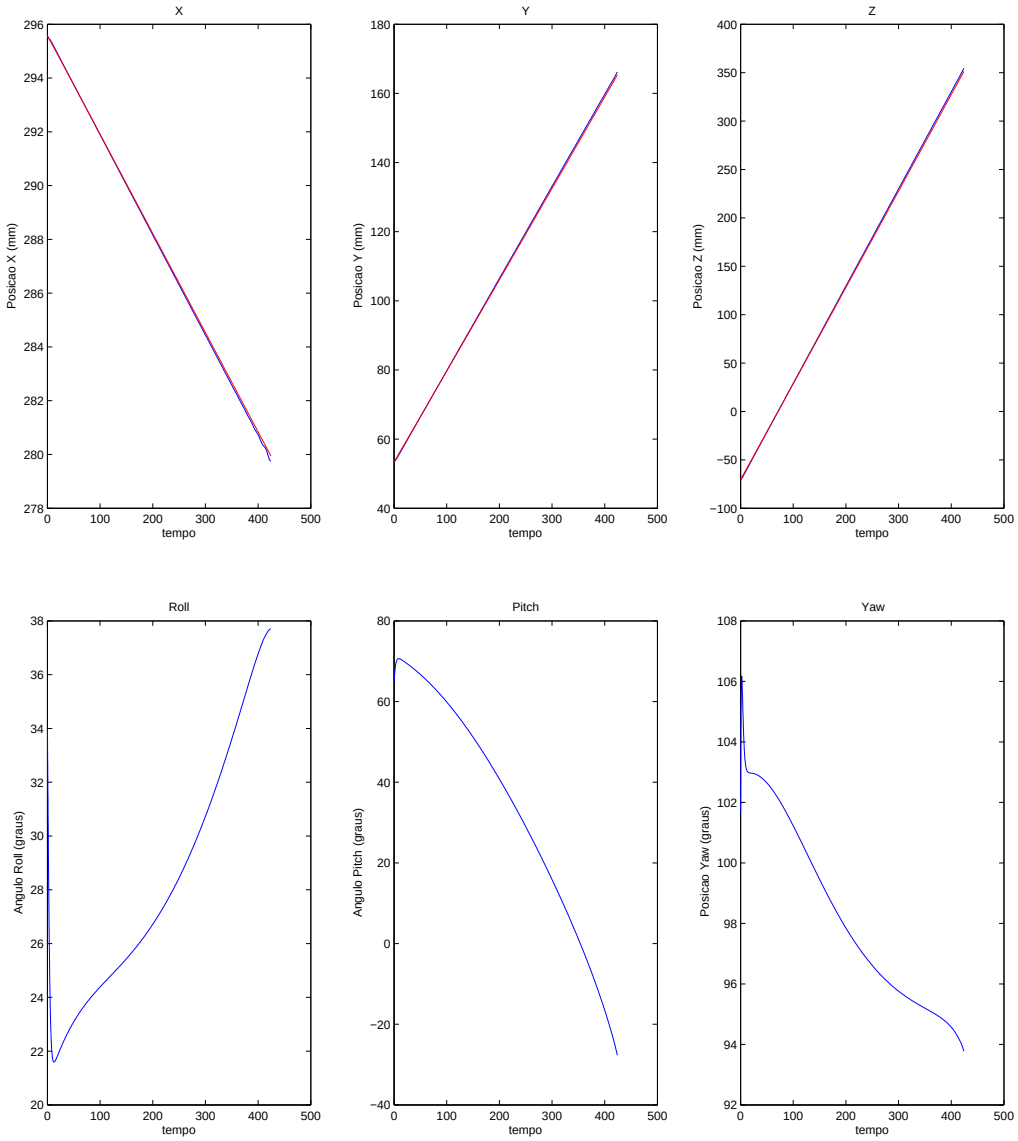


Figura 4.11: Gráficos da trajetória linear - com PG

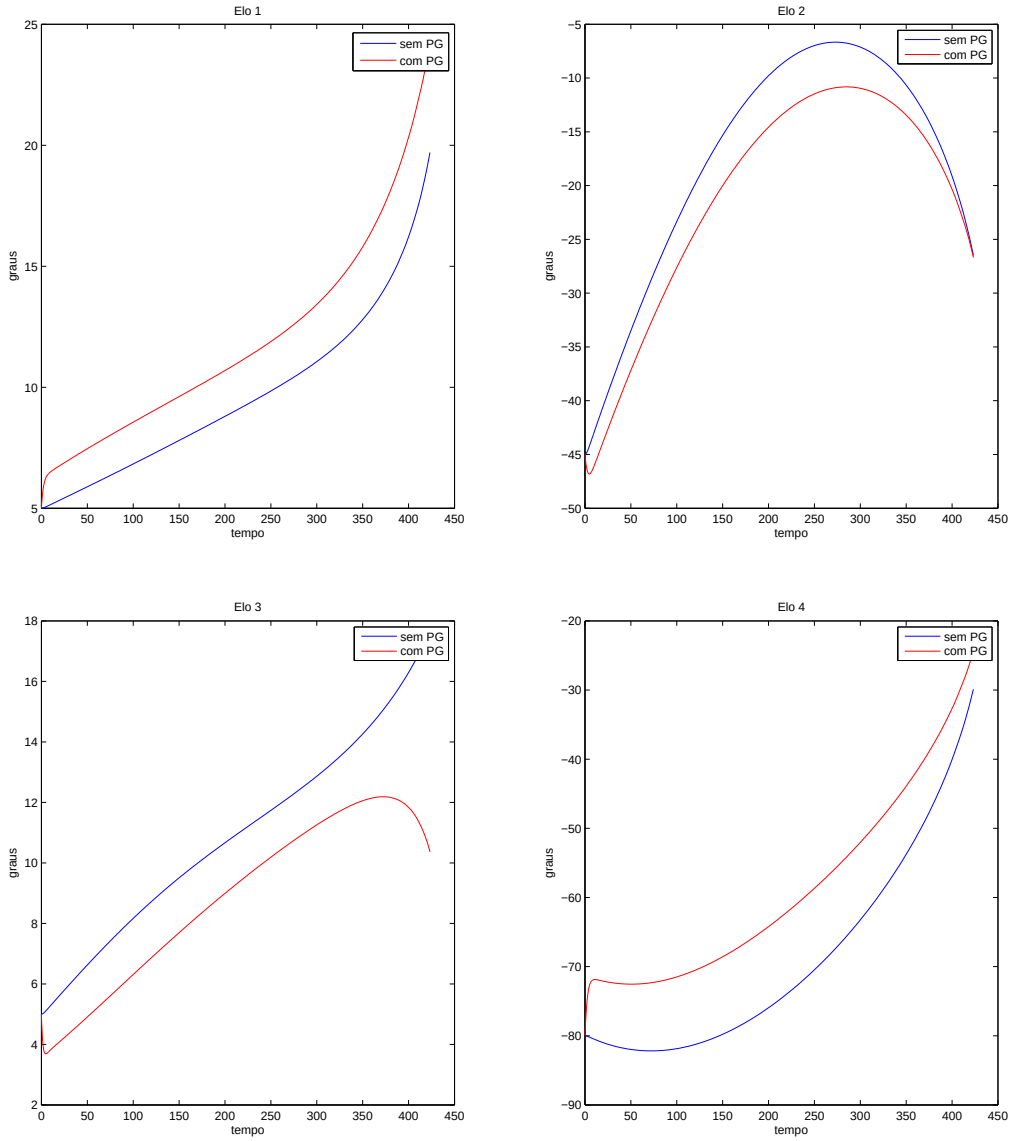


Figura 4.12: Gráficos da movimentação das juntas 1, 2, 3 e 4 na trajetória linear

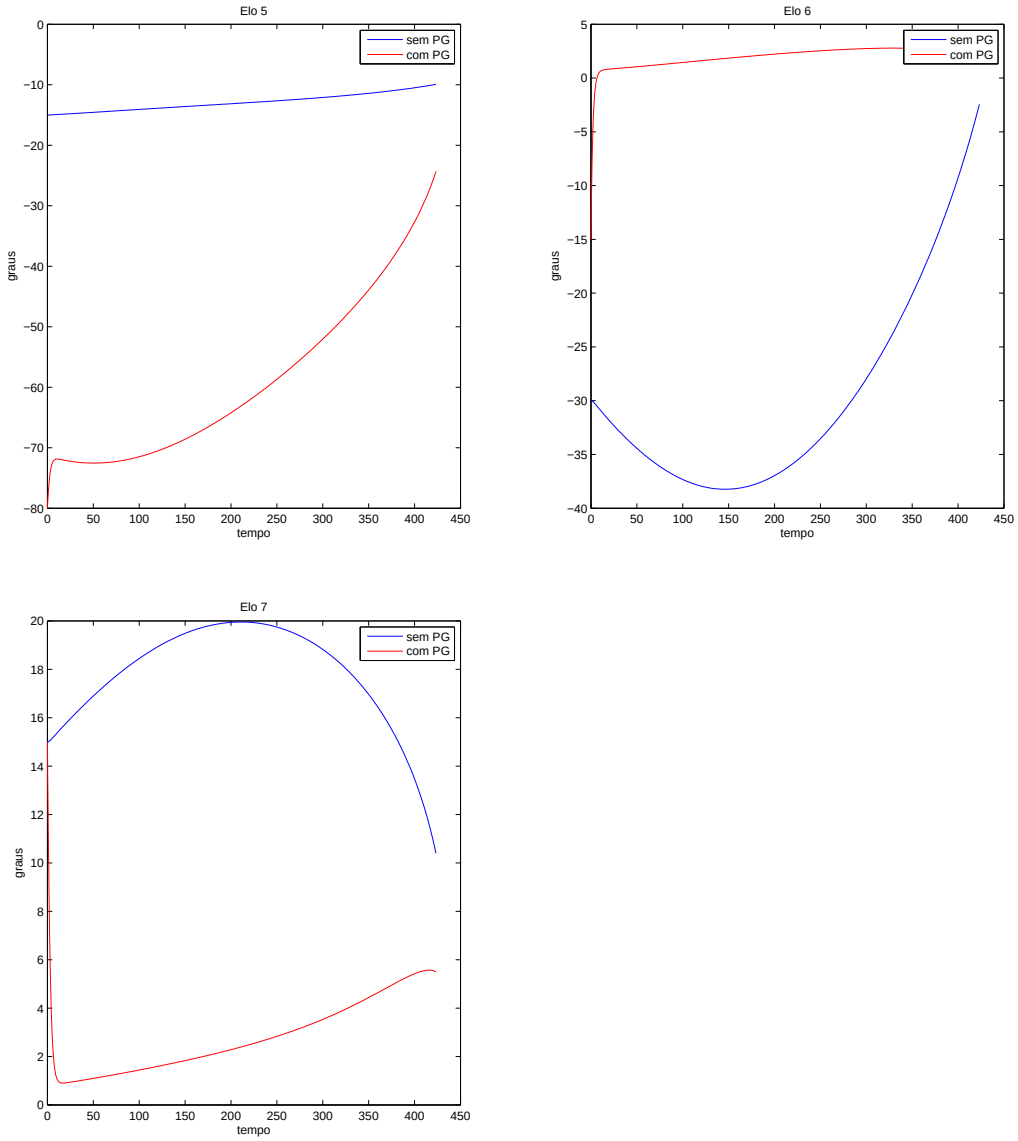


Figura 4.13: Gráficos da movimentação das juntas 5, 6 e 7 na trajetória linear

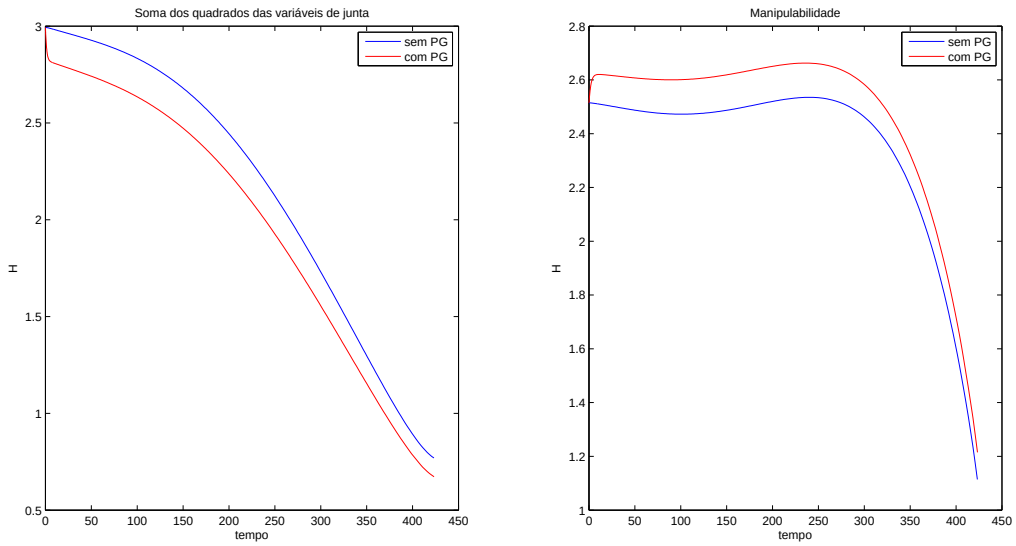


Figura 4.14: Índices de soma de ângulos e de manipulabilidade - trajetória linear

4.2.3 Trajetória circular (orientação desprezada)

Em relação à trajetória circular, ocorre aceleração diferente a cada instante no espaço cartesiano, logo é necessário realizar o cálculo da nova velocidade a cada instante. A primeira simulação foi realizada com os parâmetros da tabela 4.3.

Posição inicial (variáveis de junta)	$0°, -35°, 25°, -60°, 15°, 30°, -20°$
Raio	$60mm$
Período	$6.3s$
Plano da circunferência	yz

Tabela 4.3: Parâmetros da segunda simulação

Nas figuras 4.15 e 4.17 pode-se observar que o robô segue a trajetória com um certo erro. Depois de 4 voltas da trajetória, o erro no eixo x acumula em 0.6 mm para implementação sem projeção do gradiente e 0.35 mm para implementação com método PG.

Uma análise das figuras 4.21 e 4.22 mostra que, no método tradicional sem PG, algumas juntas não seguem um movimento repetitivo, apesar de o end-effector possuir uma trajetória cíclica. Na implementação com PG, essa questão é corrigida, uma vez que o movimento no espaço nulo sempre busca a configuração com menor valor da soma dos ângulos das juntas.

Por fim, foi possível também observar na comparação entre ambas implementações que o método com PG possuiu um índice de manipulabilidade maior que do outro caso. Em todas simulações realizadas, observou-se que em todos os casos o índice de manipulabilidade do método com PG foi maior ou igual ao índice do método tradicional sem PG.

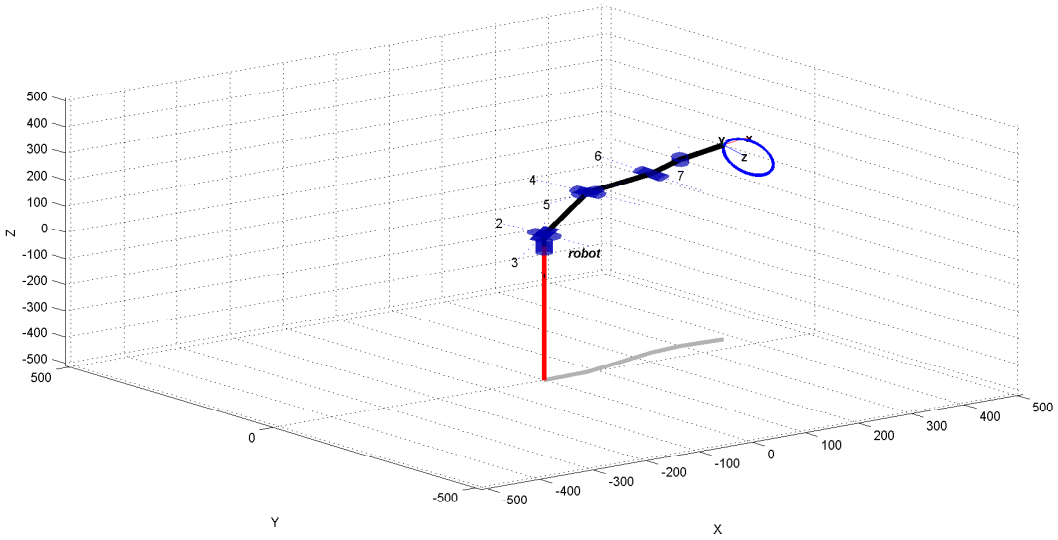


Figura 4.15: Simulação da trajetória circular - sem PG

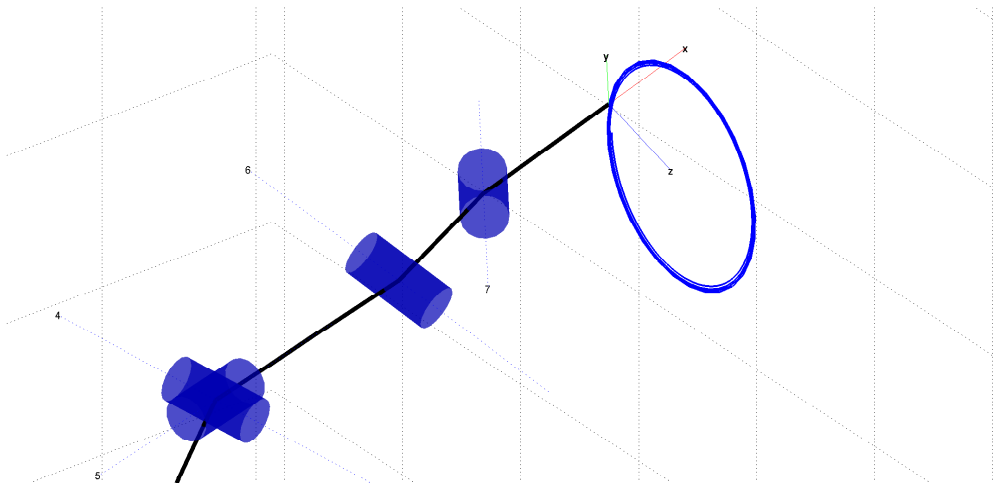


Figura 4.16: Trajetória circular em detalhe - sem PG

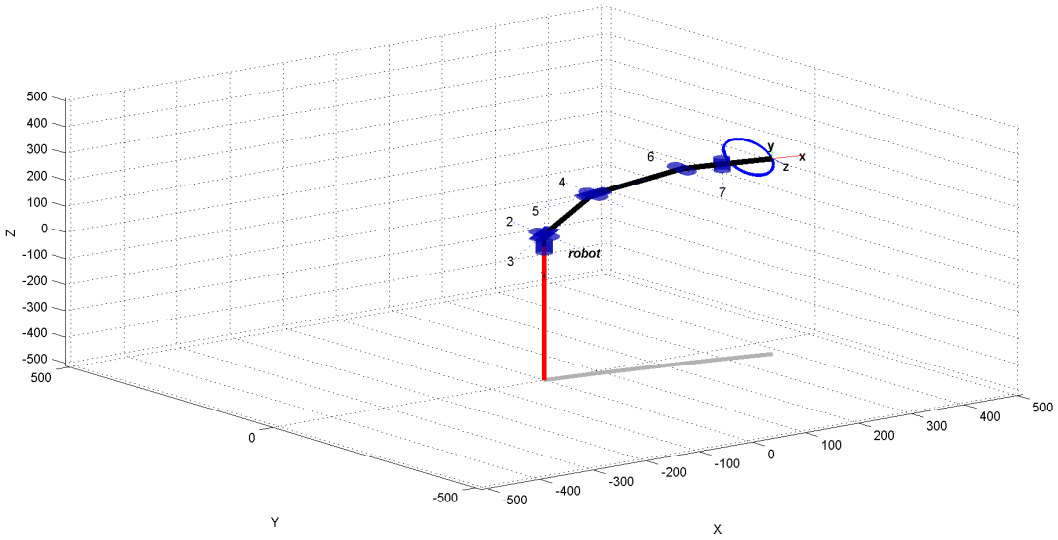


Figura 4.17: Simulação da trajetória circular - com PG

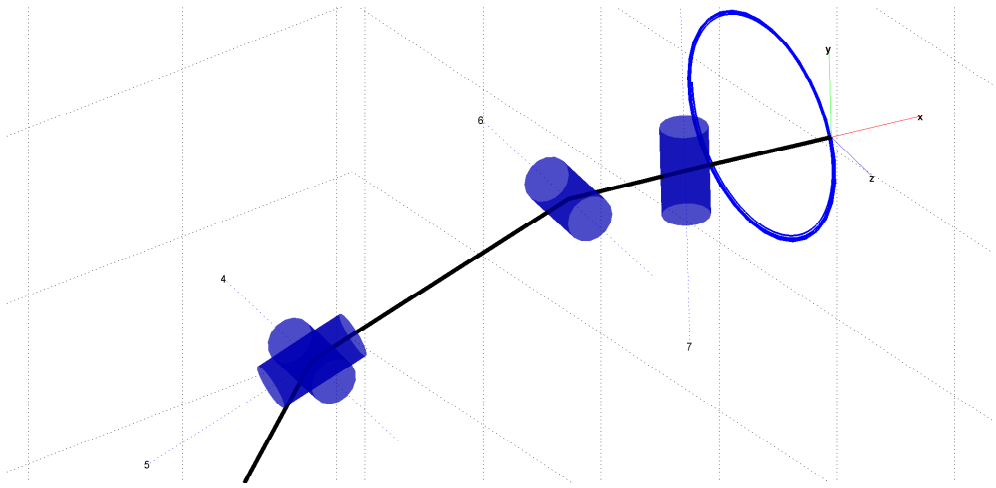


Figura 4.18: Trajetória circular em detalhe - com PG

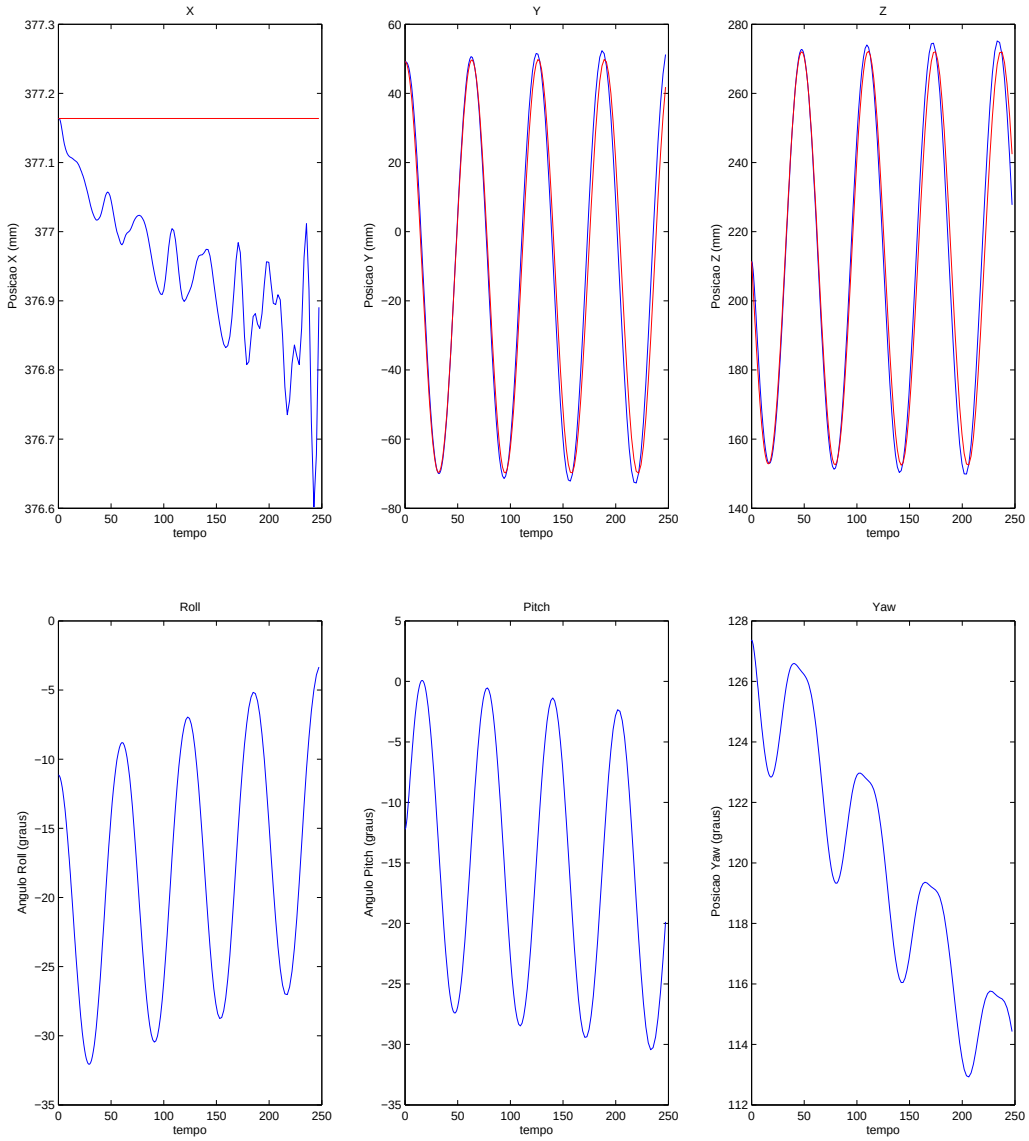


Figura 4.19: Gráficos da trajetória circular - sem PG

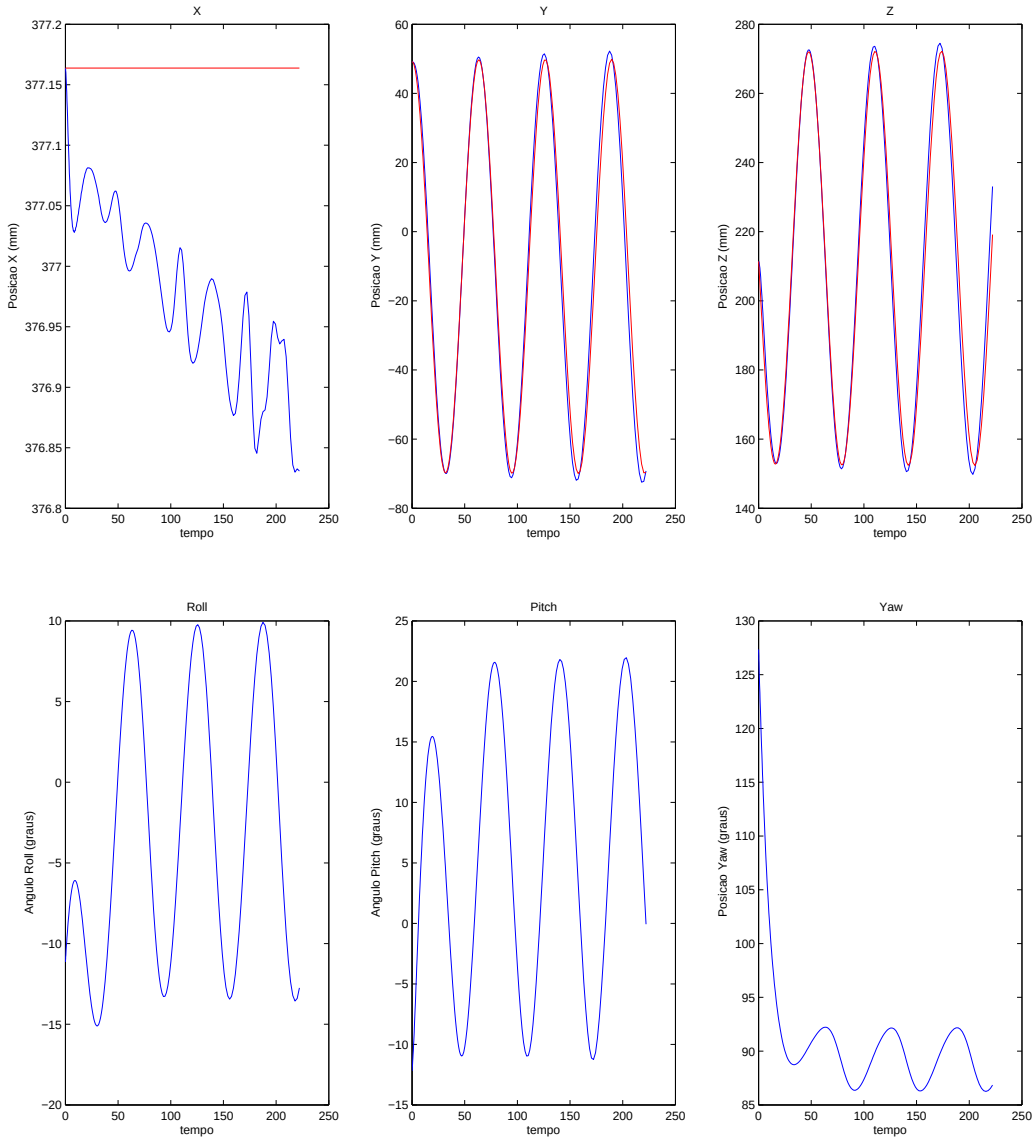


Figura 4.20: Gráficos da trajetória circular - com PG

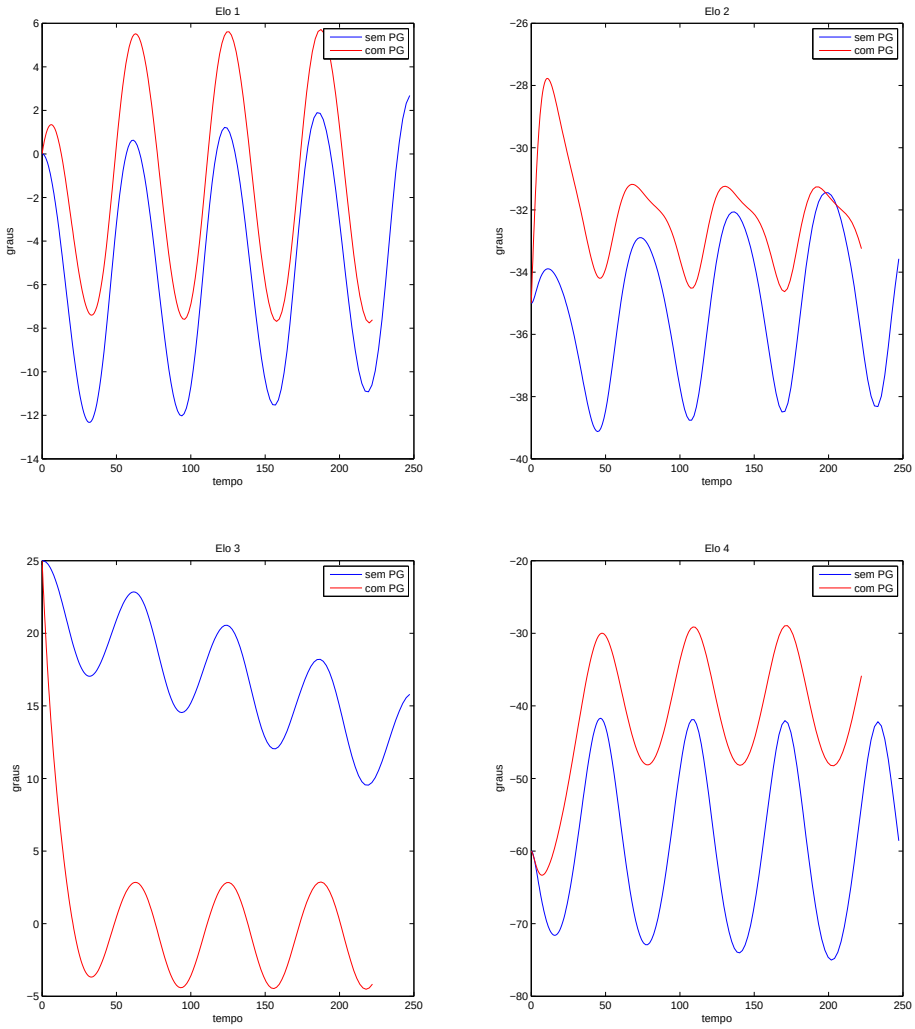


Figura 4.21: Gráficos da movimentação das juntas 1, 2, 3 e 4 na trajetória circular

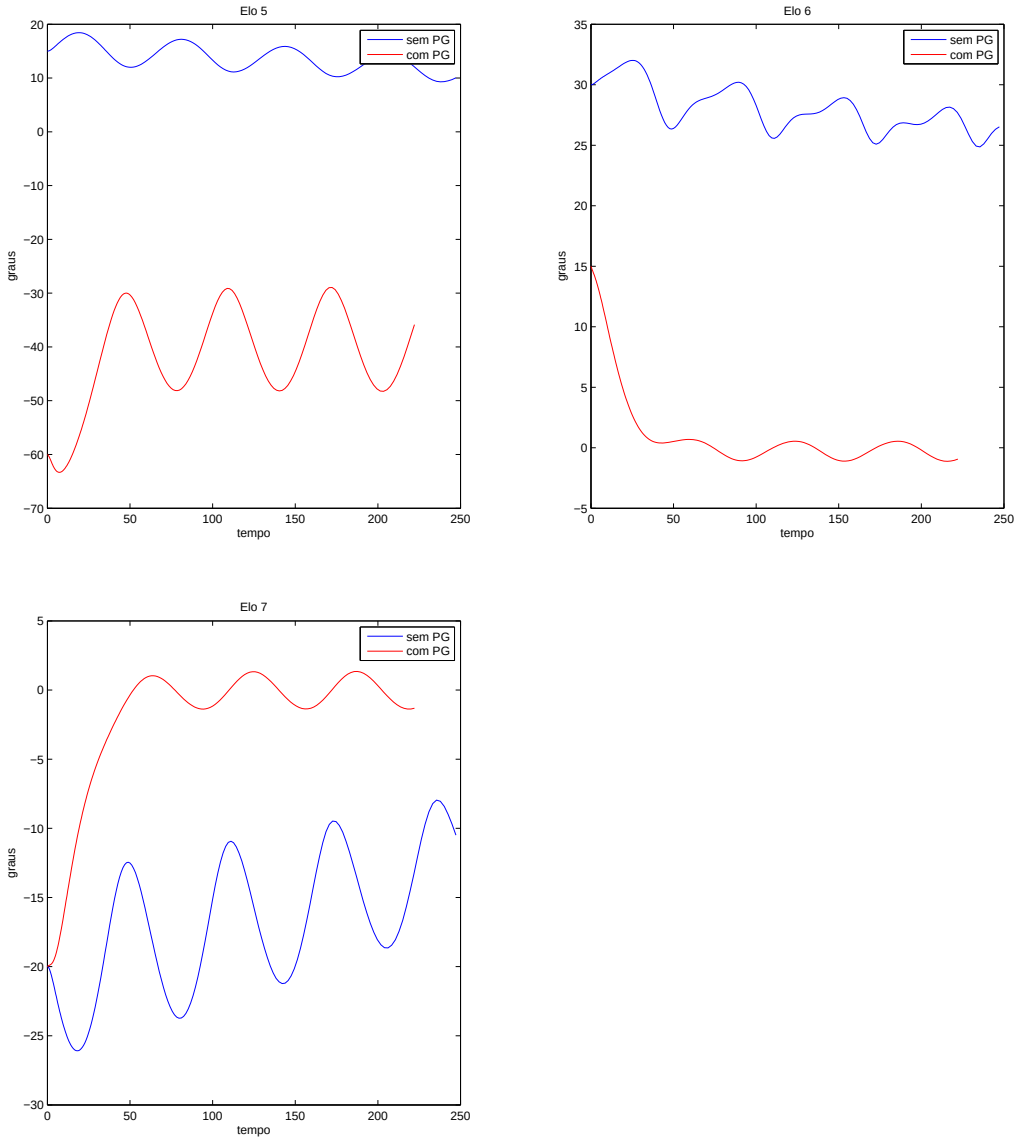


Figura 4.22: Gráficos da movimentação das juntas 5, 6 e 7 na trajetória circular

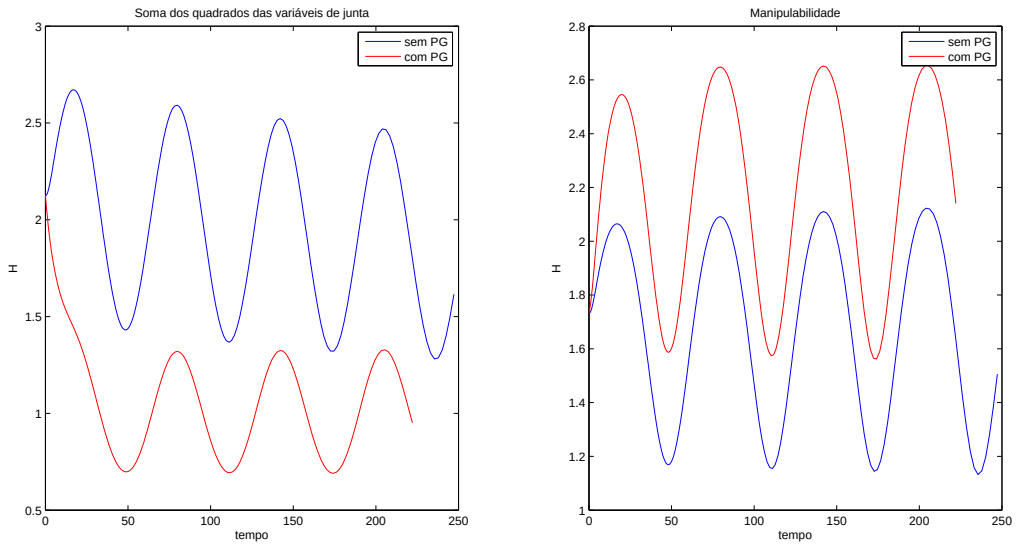


Figura 4.23: Índices de soma de ângulos e de manipulabilidade - trajetória circular

4.2.4 Trajetória circular (com orientação constante)

Na última simulação, levou-se em conta a orientação do manipulador. De qualquer forma, o robô continua redundante, já que agora são 6 variáveis cartesianas e 7 variáveis de junta. Realizou a simulação para os dois métodos apresentados neste trabalho e os resultados foram comparados.

Posição inicial (variáveis de junta)	$0°, 60°, 20°, -65°, 20°, -20°, 60°$
Raio	$20mm$
Período	$6.3s$
Plano da circunferência	yz

Tabela 4.4: Parâmetros da terceira simulação

O primeiro ponto a se destacar na comparação entre os gráficos das figuras 4.28 e 4.29 é a diferença no erro da trajetória. A trajetória com implementação da PG tem erro máximo no eixo x igual à metade do erro máximo na trajetória sem PG. Em relação aos ângulos *pitch* e *yaw*, houve também um erro menor para a trajetória com PG, entretanto a diferença é muito pequena.

Já em relação ao índice de manipulabilidade de ambas implementações, o gráfico da figura 4.32 mostra que a implementação sem PG possui um valor maior do índice de manipulabilidade. Contudo, nesta implementação, a variação deste índice não é necessariamente constante, sendo possível que em algum instante ela atinja um valor menor em relação ao índice apresentado pelo outro método.

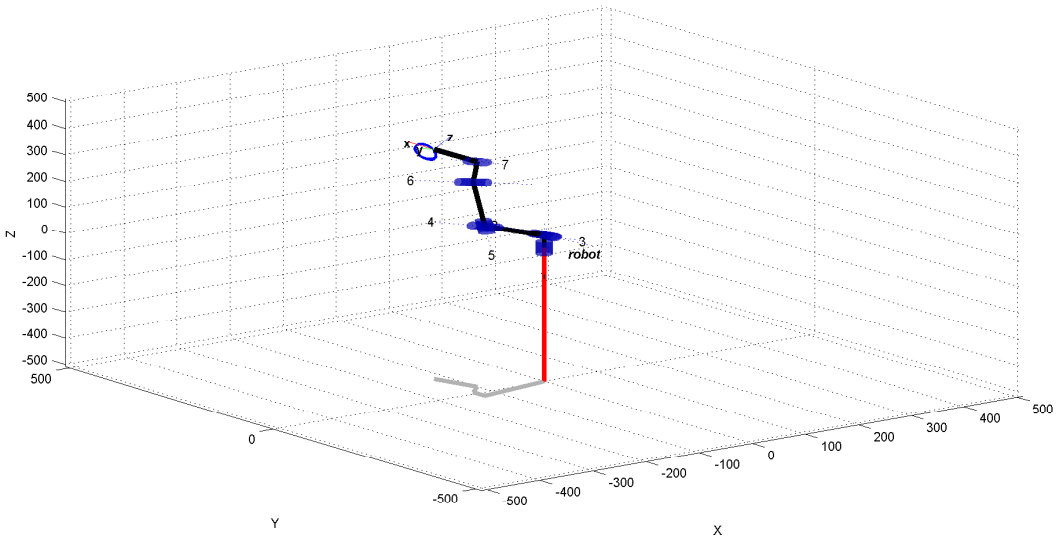


Figura 4.24: Simulação da trajetória circular (com orientação constante) - sem PG

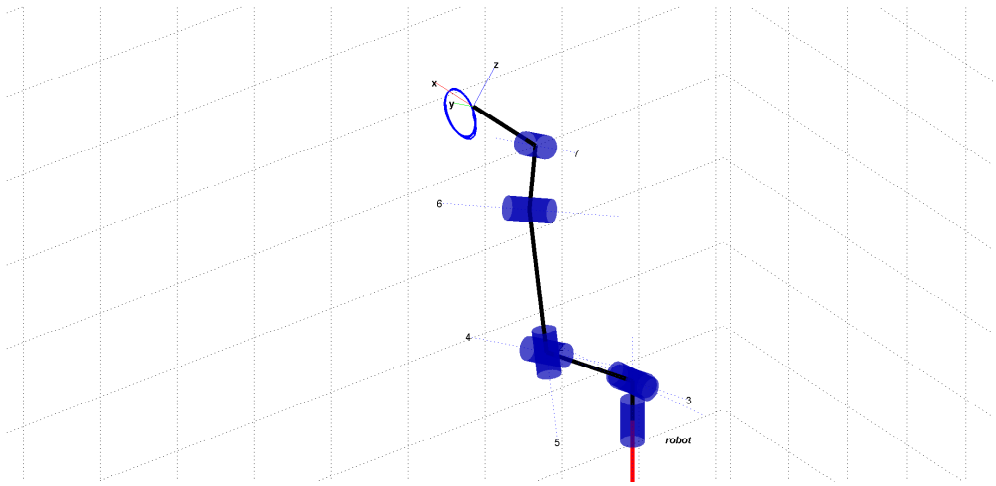


Figura 4.25: Trajetória circular em detalhe (com orientação constante) - sem PG

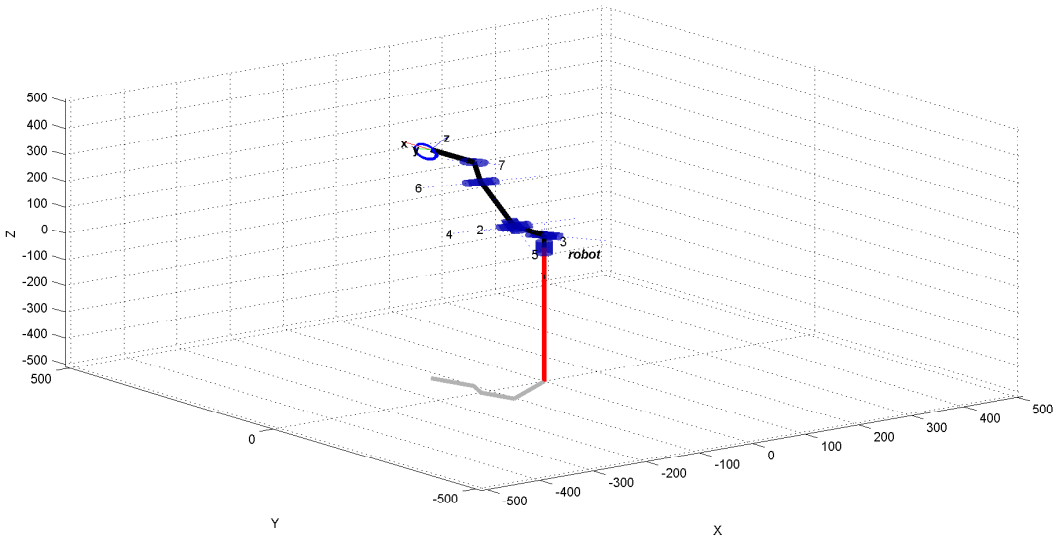


Figura 4.26: Simulação da trajetória circular (com orientação constante) - com PG

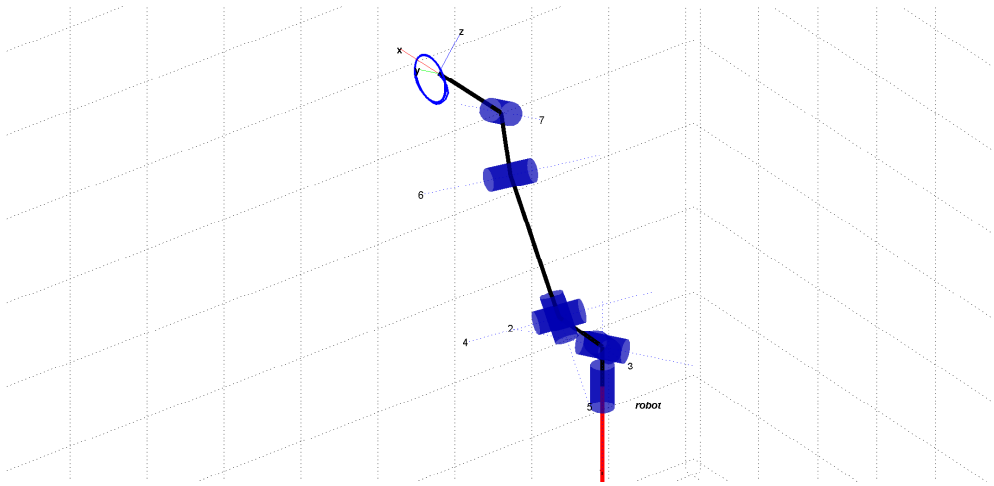


Figura 4.27: Trajetória circular em detalhe (com orientação constante) - com PG

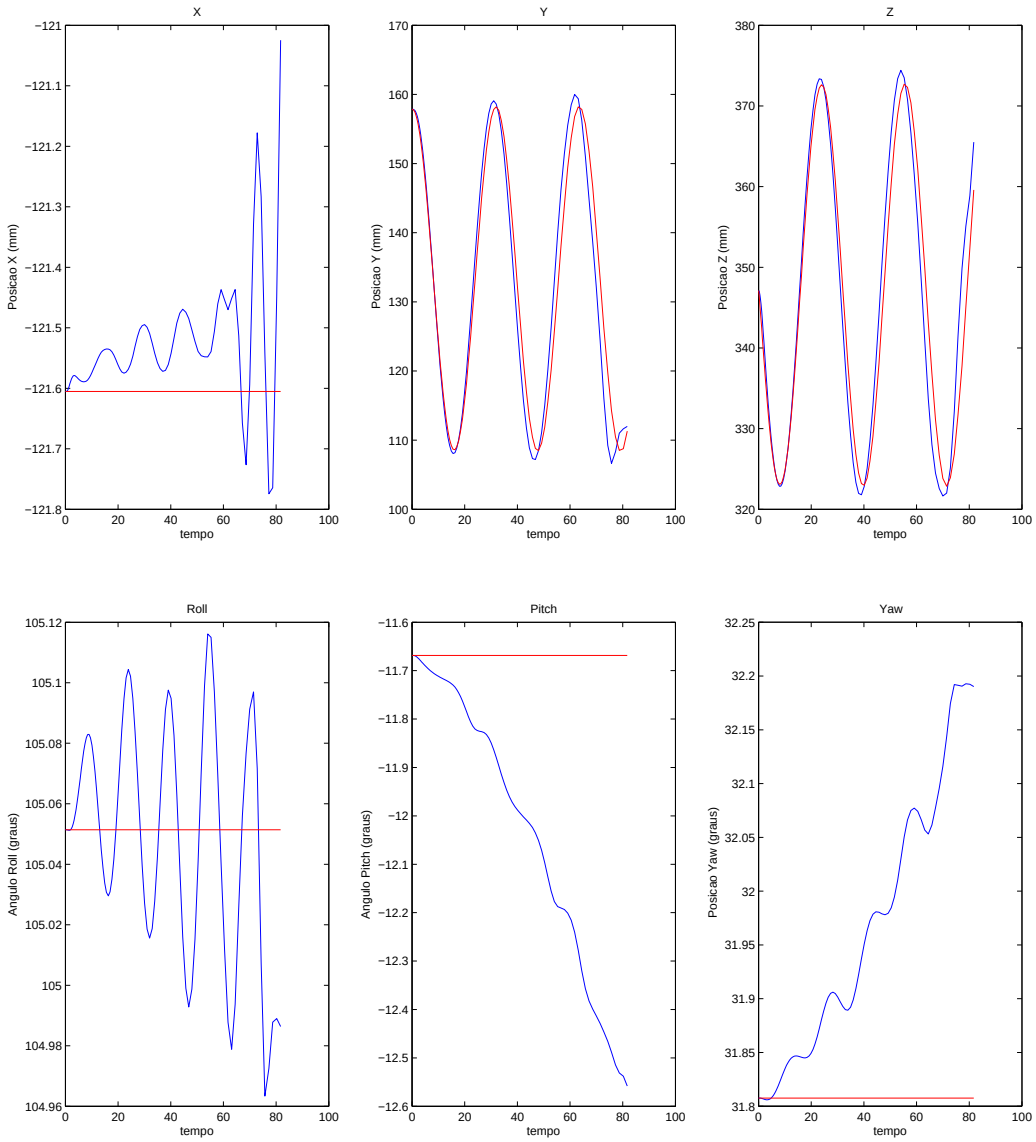


Figura 4.28: Gráficos da trajetória circular (com orientação constante) - sem PG

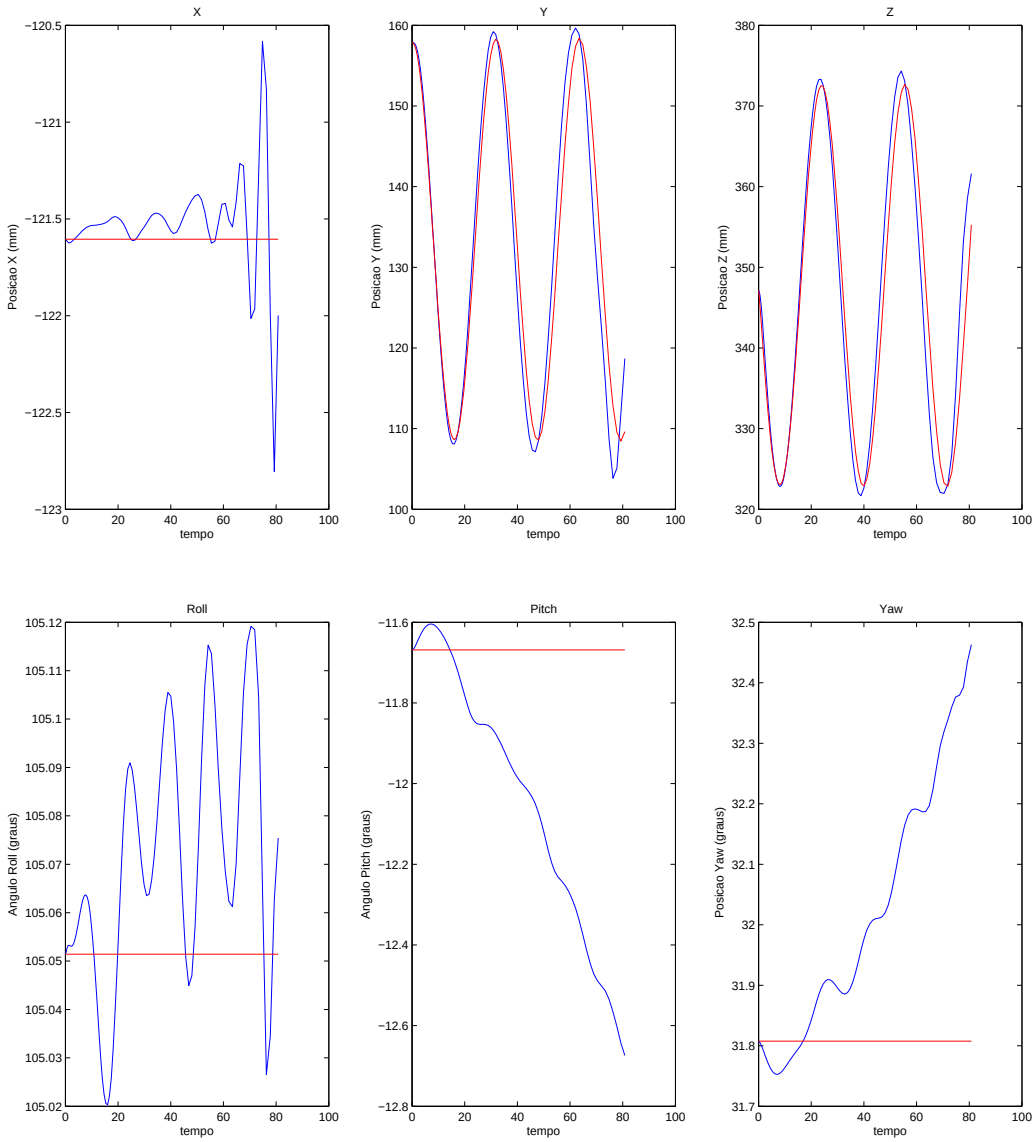


Figura 4.29: Gráficos da trajetória circular (com orientação constante) - com PG

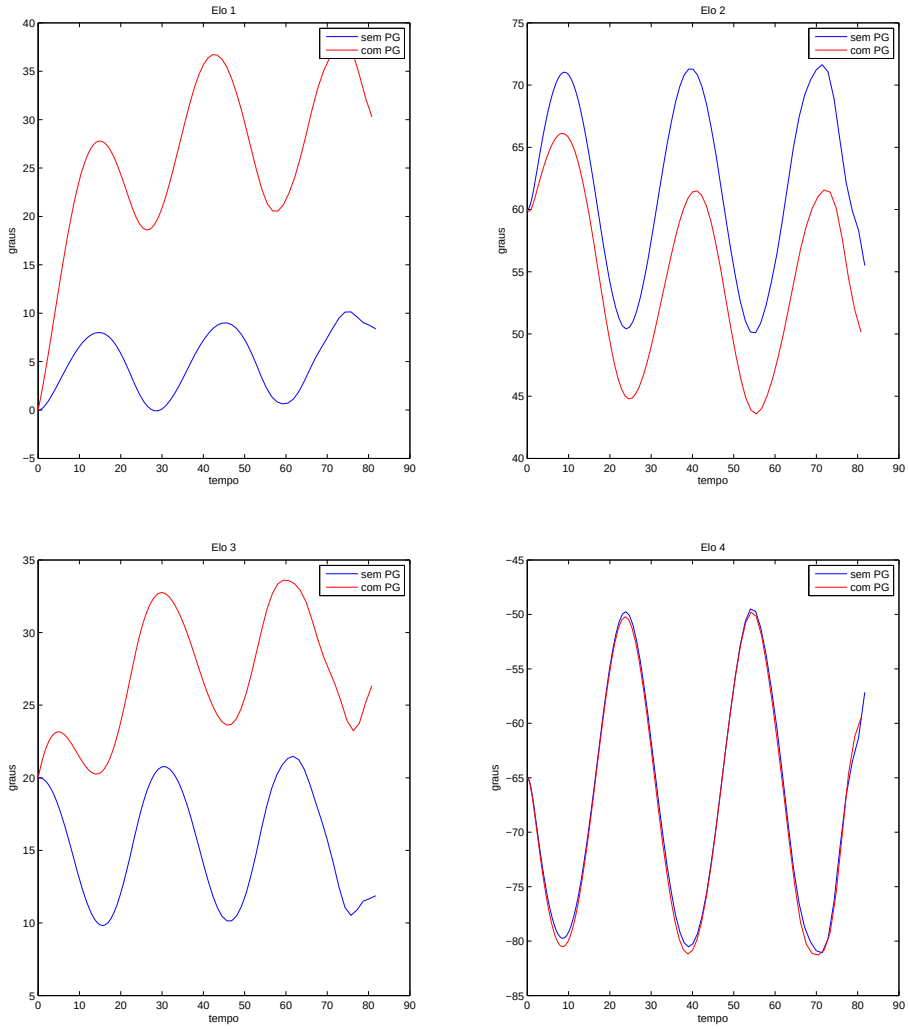


Figura 4.30: Gráficos da movimentação das juntas 1, 2, 3 e 4 na trajetória circular (com orientação constante)

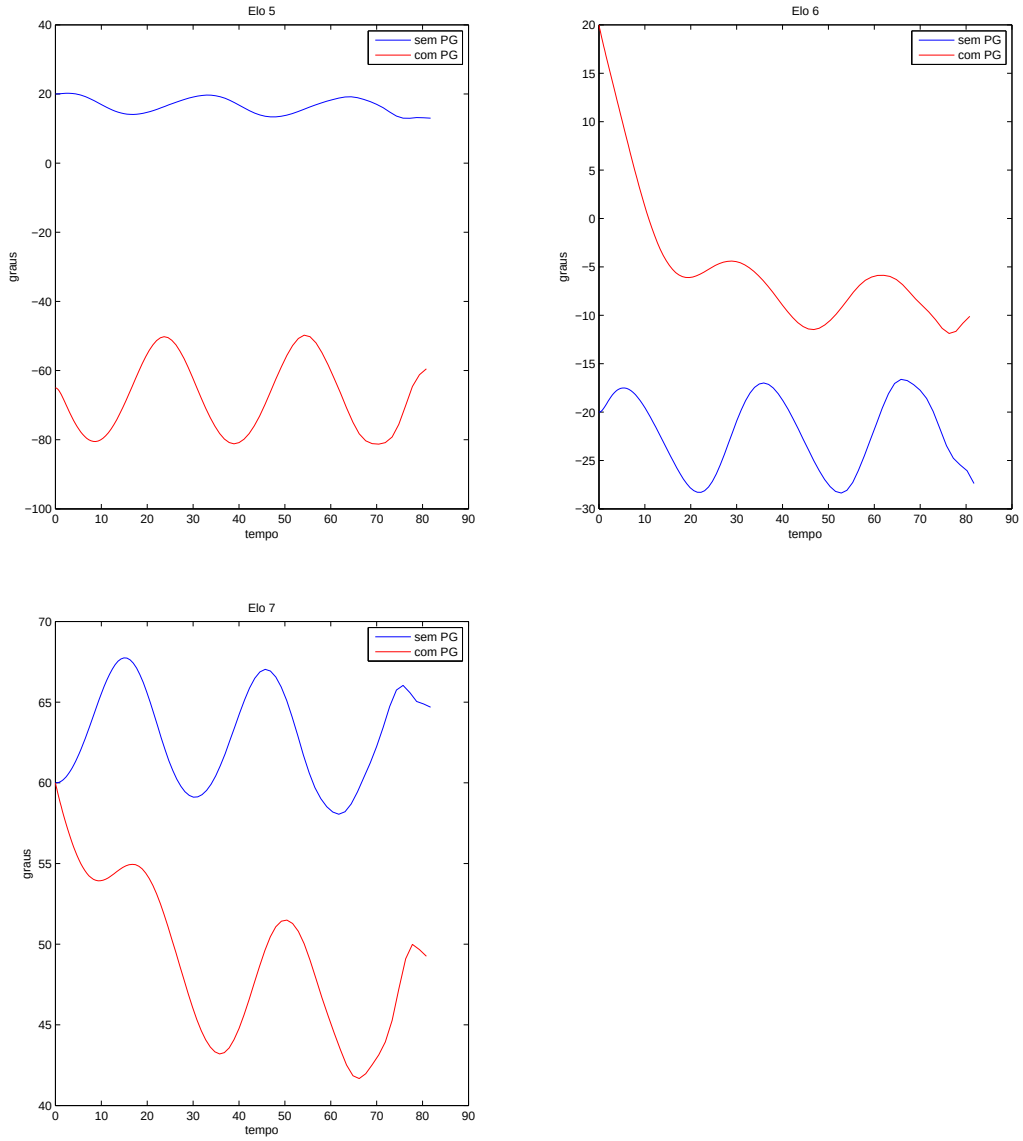


Figura 4.31: Gráficos da movimentação das juntas 5, 6 e 7 na trajetória circular (com orientação constante)

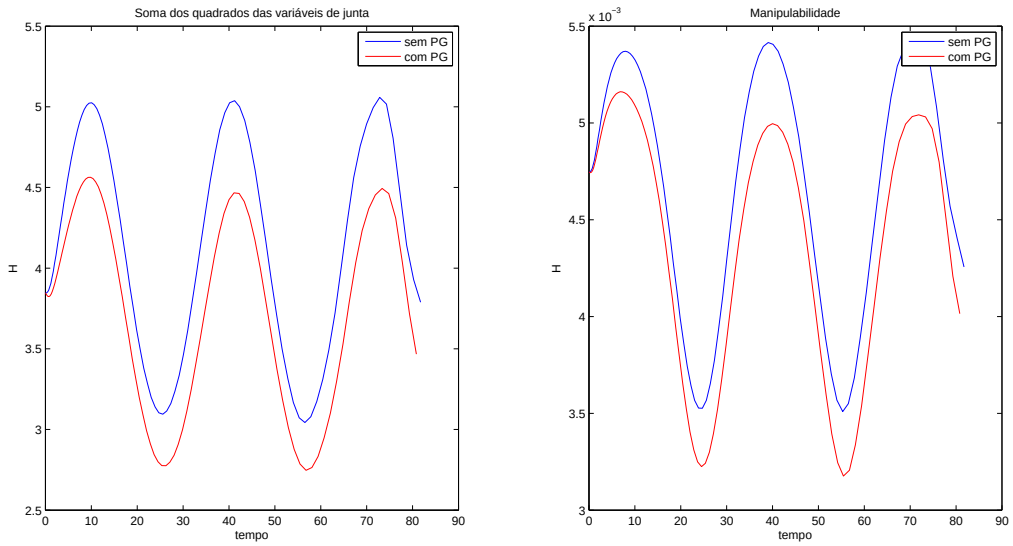


Figura 4.32: Índices de soma de ângulos e de manipulabilidade - trajetória circular (com orientação constante)

4.3 Implementação no *FPGA*

O código implementado para execução da trajetória está presente no Anexo. Em relação ao código de simulação foi-se necessário algumas alterações. Primeiramente, foi-se necessário a criação de uma função para envio de valor de velocidade para o controlador do robô e uma função para leitura de posição. O controle dos servomotores é realizado por um controlador *SSC-32* da *LynxMotion*. A comunicação entre FPGA e controlador é serial, sendo que os comandos tem codificação *ASCII*. A partir deste controlador, é possível realizar o comando de velocidade para cada junta e a leitura de posição atual. Para leitura de posição, deve-se enviar o comando

QP <arg> <cr>

onde <**arg**> é o valor da junta em questão. Para o envio de velocidade, o comando necessário é

 # <arg> P <pw> S <spd> <cr>

onde <**pw**> é a posição final e <spd> é a velocidade para <**arg**>.

Além disso, também foi necessário fazer a conversão dos valores das variáveis de junta, uma vez que o controlador mede a posição dos servomotores em pulsos. Uma variação de 170° equivale a 1500 passos e a posição 0° é equivalente a posição 1500 passos. Logo, a equação de conversão é:

$$p = 1500 + \frac{\alpha * 1500}{170} \tag{4.1}$$

A princípio levou-se em conta somente as três variáveis de posição x, y e z no cálculo da trajetória, a fim de simplificar os cálculos realizados pelo *FPGA*.

Para o teste, utilizou-se a posição inicial: $0°, 60°, 20°, -65°, 20°, -20°, 60°$ e o movimento na direção do eixo z.

Contudo, o teste não obteve êxito devido aos seguintes motivos:

- Cada ciclo dura um longo tempo (em volta de 3,5 segundos) o que compromete a corretude da trajetória;
- A comunicação entre controlador do robô e *FPGA* apresentou problemas com grande frequência. Frequentemente, a *FPGA* perdia capacidade de leitura e escrita de posição de alguma junta e somente a incapacidade de leitura das variáveis de junta impede a resolução da cinemática direta, que é um cálculo primordial para a malha de cálculo do movimento.

Capítulo 5

Conclusões

Com a teoria referente à modelagem cinemática e trajetória aplicadas à um robô manipulador redundante em mãos, este trabalho apresentou e analisou algumas características especiais relacionadas à este tipo de manipulador. Um manipulador redundante apresenta a vantagem de poder possuir diversas configurações no espaço de juntas para uma mesma posição no espaço cartesiano. Isto se mostra útil, uma vez que pode ser usado para evitar obstáculos durante uma trajetória ou então para diminuir torque em juntas, por exemplo. Porém, esta mesma vantagem carrega uma dificuldade, pois aumenta consideravelmente a complexidade dos cálculos, além de aumentar o número de posições singulares e exigir cálculos de otimização para seguir determinadas restrições de trajetória.

A partir de simulações, foram apresentados e discutidos dois métodos de cálculo da trajetória: a *CLIK* com e sem a Projeção de Gradiente tendo como restrição o alcance disponível das juntas. A restrição levada em conta não é a mais adequada para evitar posições singulares, contudo apresentou resultados satisfatórios. O uso do índice de manipulabilidade como critério de otimização da redundância apresentou-se como um método inviável, para o robô utilizado no trabalho.

Com o sucesso obtido nas simulações, partiu-se para a execução do código no *FPGA* para controle do manipulador. Contudo, verificou-se alguns problemas que inviabilizaram a sua execução de forma satisfatória. Em um primeiro momento, observou-se que cada ciclo demandava uma grande quantidade de tempo, em torno de 3.5 segundos, o que não é interessante para a realização do controle de trajetória. Já em um segundo ponto, destaca-se o problema de comunicação entre *FPGA* e controlador, pois em diversos momentos o *FPGA* perde o acesso a leitura e escrita da posição de algumas juntas.

Tendo em vista tudo que foi apresentado, o próximo passo a ser seguido em trabalhos futuros de continuação deste, seria a implementação em *Hardware* do código de controle da trajetória, uma vez que nos testes realizados o problema do longo tempo de ciclo é causado pela limitação de computação do processador *NIOS II*. Outro passo que pode ser trabalhado futuramente, é a realização controle dos servomotores do manipulador pelo próprio *FPGA* de forma a eliminar a necessidade do controlador do robô como interface entre robô e *FPGA*.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SANTOS, V. M. F. *Robótica Industrial.* [S.l.]: Universidade de Aveiro, 2003.

[2] CRAIG, J. *Introduction to Robotics: Mechanics and Control.* [S.l.]: Pearson Education, Incorporated, 2005.

[3] RIVIN, E. I. *Robots; Design and constuction.* [S.l.]: McGraw-Hill, 1988.

[4] SICILIANO L. SCIAVICCO, L. V. B.; ORIOLO, G. *Robotics: Modelling, Planning and Control.* [S.l.]: Springer Publishing Company, Inc., 2009.

[5] VUKOBRATOVIC, M. K. e M. Contribution to control of redundant robotic manipulators in a environment with obstacles. 1986.

[6] YOSHIKAWA, T. Manipulability of robotic mechanisms. The International Hournal of Robotics Research, vol.4, no.2, pp. 3-9, 1985.

[7] SICILIANO, B. Kinematic control of redundant robot manipulators: A tutorial. Journal of lntelligent and Robotic Systems 3: 201-212, 1990.

[8] SLABAUGH, G. G. Computing euler angles from a rotation matrix. 1999.

[9] WHITNEY, D. E. *The mathematics of coordinated control of prosthetic arms and manipulator.* [S.l.]: Trans. ASME J. Dynamic Systems, Measurement and Control, 1972.

[10] WANG YANGMIN LI, X. Z. J. Inverse kinematics and control of a 7-dof redundant manipulator based on the closed-loop algorithm. International Journal of Advanced Robotic Systems 7.4: 1-9, 2010.

[11] BORGES, G. A. Gmatrix: uma biblioteca matricial para c/c++. http://lara.unb.br/ gaborges/recursos/programacao/gmatrix/gmatrixdoc.pdf, 2005.

[12] CORKE, P. *Robotics, Vision and Control.* [S.l.]: http://www.petercorke.com/RVC/.

[13] INTRODUCTION to the Altera Qsys System Integration Tool. Outubro, 2012.

APÊNDICES

I. CÓDIGO DA SIMULAÇÃO GRÁFICA EM MATLAB

I.1 Simulação

```
close all; clear all; clc;

%Carregamento de dados calculados para trajetória
array;

%Carregamento do robô animado
mdl_robot;
deg = pi/180;
tam = size(pTraj);
t_total = tam(1);

%Loop que cria animação da trajetória do manipulador e gera deslocamento do
%end-effector com base na cinemática direta do manipulador
for i = 1:1:t_total

      q(1) = yout(i,1);
      qt1(i) = q(1);
      q(2) = yout(i,2);
      qt2(i) = q(2);
      q(3) = yout(i,3);
      qt3(i) = q(3);
      q(4) = yout(i,4);
      qt4(i) = q(4);
      q(5) = yout(i,5);
      qt5(i) = q(5);
      q(6) = yout(i,6);
      qt6(i) = q(6);
      q(7) = yout(i,7);
      qt7(i) = q(7);

    MatrTransf;
    p2(i,:) = CinemDir(A0T);

    r1.plot(yout(i,:));
    hold on;
    plot3(p2(1:i,1), p2(1:i,2), p2(1:i,3),'LineWidth',2);
    drawnow;
end

%Impressão dos diversos gráficos de informações
figure(2);
subplot(1,3,1);
```

```
plot(t(1:i),p2(1:i,1));
hold on;
plot(t(1:i),pTraj(1:i,1),'r');
xlabel('tempo');
ylabel('Posicao X (mm)');
title('X');

subplot(1,3,2);
plot(t(1:i),p2(1:i,2));
hold on;
plot(t(1:i),pTraj(1:i,2),'r');
%hold on;
%plot(t,e(t,2),'g');
xlabel('tempo');
ylabel('Posicao Y (mm)');
title('Y');

subplot(1,3,3);
plot(t(1:i),p2(1:i,3));
hold on;
plot(t(1:i),pTraj(1:i,3),'r');
%hold on;
%plot(t,e(t,3),'g');
xlabel('tempo');
ylabel('Posicao Z (mm)');
title('Z');

figure(3)
subplot(1,3,1);
plot(t(1:i),p2(1:i,4)/deg);
hold on;
plot(t(1:i),pTraj(1:i,4),'r');
xlabel('tempo');
ylabel('Angulo Roll (graus)');
title('Roll');

subplot(1,3,2);
plot(t(1:i),p2(1:i,5)/deg);
hold on;
plot(t(1:i),pTraj(1:i,5),'r');
xlabel('tempo');
ylabel('Angulo Pitch (graus)');
title('Pitch');

subplot(1,3,3);
plot(t(1:i),p2(1:i,6)/deg);
hold on;
plot(t(1:i),pTraj(1:i,6),'r');
xlabel('tempo');
ylabel('Posicao Yaw (graus)');
title('Yaw');
```

```
figure(5);

subplot(1,2,1);
plot(t(1:i),qt1(1:i)*180/3.1415);
xlabel('tempo');
ylabel('Ângulo');
title('Elo 1');
subplot(1,2,2);
plot(t(1:i),qt2(1:i)*180/3.1415);
xlabel('tempo');
ylabel('Ângulo');
title('Elo 2');

figure(6)
subplot(1,2,1);
plot(t(1:i),qt3(1:i)*180/3.1415);
xlabel('tempo');
ylabel('Ângulo');
title('Elo 3');
subplot(1,2,2);
plot(t(1:i),qt4(1:i)*180/3.1415);
xlabel('tempo');
ylabel('Ângulo');
title('Elo 4');

figure(7)
subplot(1,2,1);
plot(t(1:i),qt5(1:i)*180/3.1415);
xlabel('tempo');
ylabel('Ângulo');
title('Elo 5');
subplot(1,2,2);
plot(t(1:i),qt6(1:i)*180/3.1415);
xlabel('tempo');
ylabel('Ângulo');
title('Elo 6');

figure(8)
subplot(1,2,1);
plot(t(1:i),qt7(1:i)*180/3.1415);
xlabel('tempo');
ylabel('Ângulo');
title('Elo 7');

figure(9)

subplot(1,2,1)
plot(t(1:i),H1(1:i),'g');
xlabel('Soma dos angulos');
ylabel('tempo');
title('Soma dos angulos');
```

```
subplot(1,2,2)
plot(t(1:i),H2(1:i),'r');
xlabel('Manipulabilidade');
ylabel('tempo');
title('Manipulabilidade');

qt = [qt1; qt2; qt3; qt4; qt5; qt6; qt7];
H = [H1; H2];
tempo = t;
```

I.2 Matriz de transformação

```
t1 = q(1);
t2 = q(2);
t3 = q(3);
t4 = q(4);
t5 = q(5);
t6 = q(6);
t7 = q(7);

T01 = [cos(t1) 0 sin(t1) 0;
       sin(t1) 0 -cos(t1) 0;
       0 1 0 47;
       0 0 0 1];

T12 = [cos(t2) 0 -sin(t2) 0;
       sin(t2) 0 cos(t2) 0;
       0 -1 0 0;
       0 0 0 1];

T23 = [cos(t3) 0 sin(t3) 0;
       sin(t3) 0 -cos(t3) 0;
       0 1 0 154.3;
       0 0 0 1];

T34 = [cos(t4) 0 -sin(t4) 0;
       sin(t4) 0 cos(t4) 0;
       0 -1 0 0;
       0 0 0 1];

T45 = [cos(t5) 0 sin(t5) 0;
       sin(t5) 0 -cos(t5) 0;
       0 1 0 159.25;
       0 0 0 1];

T56 = [cos(t6+pi/2) 0 -sin(t6+pi/2) 67*cos(t6+pi/2);
       sin(t6+pi/2) 0 cos(t6+pi/2) 67*sin(t6+pi/2);
       0 -1 0 0;
       0 0 0 1];

T67 = [cos(t7) 0 sin(t7) 83*cos(t7);
```

```
        sin(t7) 0 -cos(t7) 83*sin(t7);
        0 1 0 0;
        0 0 0 1];

T07 = T01*T12*T23*T34*T45*T56*T67;
A0T = T07;
```

I.3 Cinemática direta

```
function p = CinemDir(MT)

x = MT(1,4);
y = MT(2,4);
z = MT(3,4);

psi = atan2(MT(2,1),MT(1,1));
tetha = atan2(-MT(3,1),sqrt(MT(1,1)^2+MT(2,1)^2));
phi = atan2(MT(3,2),MT(3,3));

p = [x y z psi tetha phi];
```

II. CÓDIGO DA SIMULAÇÃO DO CONTROLE DE TRAJETÓRIO EM C

```
#include <stdio.h>
#include "gmatrix.h"
#include <math.h>
#include <time.h>

    double T = 0.01;
    double Kp = 0.5;
    double Kd = 0;
    double Ki = 0;
    double deg = GMATRIXCONST_PI/180;
    double tempo_ciclo;
    double tempo_total;
    const double veloc_max = 1;
    double velocX, velocY, velocZ;
    double s1, s2, s3, s4, s5, s6, s7, c1, c2, c3, c4, c5, c6, c7;
    double H1, H2;

    GMATRIX_DECLARE(q, 7, 1);
    GMATRIX_DECLARE(dq, 7, 1);
    GMATRIX_DECLARE(p, 3, 1);
    GMATRIX_DECLARE(dp, 3, 1);
    GMATRIX_DECLARE(pTraj,3,1);
    GMATRIX_DECLARE(pInit,3,1);
    GMATRIX_DECLARE(pTemp,3,1);
    GMATRIX_DECLARE(dif,3,1);
    GMATRIX_DECLARE(e,3,1);
    GMATRIX_DECLARE(ePrev,3,1);
    GMATRIX_DECLARE(deriv,3,1);
    GMATRIX_DECLARE(integ,3,1);
    GMATRIX_DECLARE(J, 3, 7);
    GMATRIX_DECLARE(Jtransp, 7, 3);
    GMATRIX_DECLARE(JJtransp, 3, 3);
    GMATRIX_DECLARE(JJdummy, 3, 3);
    GMATRIX_DECLARE(Jinv, 7, 3);
    GMATRIX_DECLARE(Dummy, 3, 7);
    GMATRIX_DECLARE(MT, 4, 4);

    //Memória de posições
    GMATRIX_DECLARE(p1, 3, 1);
    GMATRIX_DECLARE(p2, 3, 1);
    GMATRIX_DECLARE(p3, 3, 1);
    GMATRIX_DECLARE(p4, 3, 1);

    //Definição do tipo de trajetória: 0 para linear e 1 para circular
    const float linear_circular = 0;
```

```
    //Uso da projeção do gradiente
    const float projGrad = 0;

    //Número de ciclos da simulação
    const int n_ciclos = 20;

    //Posição final quando linear
    int x = 280;
    int y = 165;
    int z = 350;

    //Dimensões da circunferência
    float raio = 60;
    float frequencia = 0.1;

//Cálculo de senos e cossenos
void calcSinCos(void)
{
    double t1, t2, t3, t4, t5, t6, t7;

    t1 = GMATRIX_DATA(q,1,1);
    t2 = GMATRIX_DATA(q,2,1);
    t3 = GMATRIX_DATA(q,3,1);
    t4 = GMATRIX_DATA(q,4,1);
    t5 = GMATRIX_DATA(q,5,1);
    t6 = GMATRIX_DATA(q,6,1);
    t7 = GMATRIX_DATA(q,7,1);

    s1 = sin(t1 );
    s2 = sin(t2 );
    s3 = sin(t3 );
    s4 = sin(t4 );
    s5 = sin(t5 );
    s6 = sin(t6 );
    s7 = sin(t7 );

    c1 = cos(t1 );
    c2 = cos(t2 );
    c3 = cos(t3 );
    c4 = cos(t4 );
    c5 = cos(t5 );
    c6 = cos(t6 );
    c7 = cos(t7 );
}

//Cálculo da matriz jacobiana, da pseudo-inversa e da velocidade no espaço das
    juntas
void calcJacobian(void)
{
    GMATRIX_DECLARE(Jtransp, 3, 7);

    //Variáveis para evitar a repetição desnecessária de cálculos
```

```
double gy = (c1*s3 + c2*c3*s1);
double by = c2*s3*s5;
double ay = c5*s1*s2*s3;
double d = (c4*gy - s1*s2*s4);
double lx = c1*s2*s3*s5;
double kx = c1*c3*s2*s4;
double jx = c6*s2*s3*s4;
double ix = s1*s2*s3*s5;
double gx = c2*c3*s4;
double fx = c1*c2*c4;
double dx = c4*s2;
double b = (s4*gy + c4*s1*s2);
double c = (c3*s1 + c1*c2*s3);
double e = (s1*s3 - c1*c2*c3);
double a = (c4*e + c1*s2*s4);
double f = (s5*e - c4*c5*c);
double g = c1*c2*s4;
double h = (c1*c3 - c2*s1*s3);
double r = (s4*e - c1*dx);
double ex = (c5*a + s5*c);
double az = s6*ex;
double bz = c6*r;
double hx = (c5*d + s5*h);
double cz = (c6*b + s6*hx);
double dz = (s5*a - c5*c);
double ez = (s5*d - c5*h);
double i = (c7*(bz + az) + s7*dz);
double j = (c7*cz + s7*ez);
double ax = (c2*s4 + c3*dx);
double sx = (c5*ax - s2*s3*s5);
double vx = (c2*c4 - c3*s2*s4);
double fy = (s5*ax + c5*s2*s3);
double ly = c6*vx;
double my = (s6*sx - ly);
double k = (c7*fy - s7*my);
double l = sqrt((j*j) + (i*i));
double m = ((j*j) + (i*i))*l;
double n = (c6*sx + s6*vx);
double o = (s2*s4 - c2*c3*c4);
double tx = (c2*s1*s4 + c3*c4*s1*s2);
double ux = (c2*c4*s1 - c3*s1*s2*s4);
double yx = (c3*s2*s5 + c4*c5*s2*s3);
double bx = (c3*c5*s2 - dx*s3*s5);
double cx = c1*c3*dx;
double cy = (c5*o + by);
double dy = (s5*o - c2*c5*s3);
double ey = c6*(dx + gx);
double hy = (s5*tx + ay);
double iy = (c6*ax + c5*s6*vx);
double jy = (s6*f - c6*s4*c);
double ky = (s6*(c5*tx - ix) - c6*ux);
double ny = (s6*cy - ey);
```

```
    double oy = (c7*(c6*a - c5*s6*r) - s5*s7*r);

    GMATRIX_DATA(J,1,1) = (1543*s1*s2)/10 + 67*c6*b + 83*c7*cz + 67*s6*hx + 83*s7*
        ez + (637*s4*gy)/4 + (637*c4*s1*s2)/4;
    GMATRIX_DATA(J,1,2) = 67*s6*(c5*(g + cx) - lx) - 67*c6*(fx - kx) - 83*c7*(c6*(
        fx - kx) - s6*(c5*(g + cx) - lx)) - (1543*c1*c2)/10 + 83*s7*(s5*(g + cx) +
        c1*c5*s2*s3) - (637*fx)/4 + (637*kx)/4;
    GMATRIX_DATA(J,1,3) = (637*s4*c)/4 - 83*c7*jy - 67*s6*f + 83*s7*(c5*e + c4*s5*c
        ) + 67*c6*s4*c;
    GMATRIX_DATA(J,1,4) = 83*c7*(c6*a - c5*s6*r) + 67*c6*a + (637*c4*e)/4 + (637*c1
        *s2*s4)/4 - 67*c5*s6*r - 83*s5*s7*r;
    GMATRIX_DATA(J,1,5) = 83*s7*ex - 67*s6*dz - 83*c7*s6*dz;
    GMATRIX_DATA(J,1,6) = 67*c6*ex - 83*c7*(s6*r - c6*ex) - 67*s6*r;
    GMATRIX_DATA(J,1,7) = 83*c7*dz - 83*s7*(bz + az);

    GMATRIX_DATA(J,2,1) = 67*bz - (1543*c1*s2)/10 + 83*c7*(bz + az) + 67*az + 83*s7
        *dz + (637*s4*e)/4 - (637*c1*dx)/4;
    GMATRIX_DATA(J,2,2) = 67*s6*(c5*tx - ix) - (1543*c2*s1)/10 + 83*s7*hy + 83*c7*
        ky - 67*c6*ux - (637*c2*c4*s1)/4 + (637*c3*s1*s2*s4)/4;
    GMATRIX_DATA(J,2,3) = 83*c7*(s6*(s5*gy - c4*c5*h) - c6*s4*h) - (637*s4*h)/4 +
        67*s6*(s5*gy - c4*c5*h) - 83*s7*(c5*gy + c4*s5*h) - 67*c6*s4*h;
    GMATRIX_DATA(J,2,4) = (637*s1*s2*s4)/4 - 67*c6*d - (637*c4*gy)/4 - 83*c7*(c6*d
        - c5*s6*b) + 67*c5*s6*b + 83*s5*s7*b;
    GMATRIX_DATA(J,2,5) = 67*s6*ez - 83*s7*hx + 83*c7*s6*ez;
    GMATRIX_DATA(J,2,6) = 67*s6*b + 83*c7*(s6*b - c6*hx) - 67*c6*hx;
    GMATRIX_DATA(J,2,7) = 83*s7*cz - 83*c7*ez;

    GMATRIX_DATA(J,3,1) = 0;
    GMATRIX_DATA(J,3,2) = 67*s6*cy - (637*dx)/4 - (1543*s2)/10 + 83*s7*dy + 83*c7*
        ny - 67*ey - (637*gx)/4;
    GMATRIX_DATA(J,3,3) = 83*c7*(s6*yx + jx) + 67*s6*yx - 83*s7*bx + (637*s2*s3*s4)
        /4 + 67*jx;
    GMATRIX_DATA(J,3,4) = - (637*c2*s4)/4 - 67*c6*ax - 83*c7*iy - 67*c5*s6*vx - 83*
        s5*s7*vx - (637*c3*dx)/4;
    GMATRIX_DATA(J,3,5) = 67*s6*fy - 83*s7*sx + 83*c7*s6*fy;
    GMATRIX_DATA(J,3,6) = - 67*c6*sx - 83*c7*n - 67*s6*vx;
    GMATRIX_DATA(J,3,7) = 83*s7*my - 83*c7*fy;


  //Indice de proximidade da posição zero
    H1 = (GMATRIX_DATA(q,1,1)*GMATRIX_DATA(q,1,1) + GMATRIX_DATA(q,2,1)*
        GMATRIX_DATA(q,2,1) + GMATRIX_DATA(q,3,1)*GMATRIX_DATA(q,3,1) + GMATRIX_DATA
        (q,4,1)*GMATRIX_DATA(q,4,1) + GMATRIX_DATA(q,5,1)*GMATRIX_DATA(q,5,1) +
        GMATRIX_DATA(q,6,1)*GMATRIX_DATA(q,6,1) + GMATRIX_DATA(q,7,1)*GMATRIX_DATA(q
        ,7,1));

//Indice de manipulabilidade
    GMATRIX_TRANSPOSE_COPY(Jtransp, J);
    GMATRIX_MULTIPLY_COPY(JJtransp, J, Jtransp);
    double det = GMATRIX_DETERMINANT(JJtransp, JJdummy);
    H2 = sqrt(det)/10000000;
```

```
// Calculo dq = J^(-1)*dp
    GMATRIX_TRANSPOSE_COPY(Jtransp, J);
    GMATRIX_PSEUDOINVERSE(Jinv, J, Dummy); //se matriz não for singular usar
        gmatrix_pseudoinverse_copy pois tem um custo muito menor
    GMATRIX_MULTIPLY_COPY(dq, Jinv, dp);

//Adição da movimentação no espaço nulo dq = J^(-1)*dp + (I-J*J^+)q0
    if(projGrad)
    {
        GMATRIX_DECLARE(JJ,7,7);
        GMATRIX_DECLARE(temp,7,7);
        GMATRIX_DECLARE(temp2,7,1);
        GMATRIX_DECLARE(grad,7,1);
        GMATRIX_IDENTITY(temp);
        GMATRIX_MULTIPLY_COPY(JJ, Jinv, J);
        GMATRIX_SUBSTRACT(temp, JJ);
        GMATRIX_COPY(grad,q);
        GMATRIX_MULTIPLY_CONST(grad,(-0.1));
        GMATRIX_MULTIPLY_COPY(temp2, temp, grad);
        GMATRIX_ADD(dq,temp2);
    }
}

//Calculo da matriz de transformação
void transMatrix(void)
{
    double gy = (c1*s3 + c2*c3*s1);
    double dx = c4*s2;
    double b = (s4*gy + c4*s1*s2);
    double c = (c3*s1 + c1*c2*s3);
    double d = (c4*gy - s1*s2*s4);
    double e = (s1*s3 - c1*c2*c3);
    double a = (c4*e + c1*s2*s4);
    double f = (s5*e - c4*c5*c);
    double g = c1*c2*s4;
    double h = (c1*c3 - c2*s1*s3);
    double r = (s4*e - c1*dx);
    double ex = (c5*a + s5*c);
    double hx = (c5*d + s5*h);
    double ax = (c2*s4 + c3*dx);
    double fy = (s5*ax + c5*s2*s3);
    double vx = (c2*c4 - c3*s2*s4);
    double sx = (c5*ax - s2*s3*s5);
    double ly = c6*vx;
    double my = (s6*sx - ly);
    double az = s6*ex;
    double bz = c6*r;
    double cz = (c6*b + s6*hx);
    double dz = (s5*a - c5*c);
    double ez = (s5*d - c5*h);

    GMATRIX_DATA(MT,1,1) = c7*(bz + az) + s7*dz;
```

```
    GMATRIX_DATA(MT,1,2) = c6*ex - s6*r;
    GMATRIX_DATA(MT,1,3) = s7*(bz + az) - c7*dz;
    GMATRIX_DATA(MT,1,4) = 67*bz - (1543*c1*s2)/10 + 83*c7*(bz + az) + 67*az + 83*
        s7*dz + (637*s4*e)/4 - (637*c1*dx)/4;

    GMATRIX_DATA(MT,2,1) = - c7*cz - s7*ez;
    GMATRIX_DATA(MT,2,2) = s6*b - c6*hx;
    GMATRIX_DATA(MT,2,3) = c7*ez - s7*cz;
    GMATRIX_DATA(MT,2,4) = - (1543*s1*s2)/10 - 67*c6*b - 83*c7*cz - 67*s6*hx - 83*
        s7*ez - (637*s4*gy)/4 - (637*c4*s1*s2)/4;

    GMATRIX_DATA(MT,3,1) = - s7*fy - c7*my;
    GMATRIX_DATA(MT,3,2) = - c6*sx - s6*vx;
    GMATRIX_DATA(MT,3,3) = c7*fy - s7*my;
    GMATRIX_DATA(MT,3,4) = (1543*c2)/10 + (637*c2*c4)/4 - 67*s6*sx - 83*s7*fy - 83*
        c7*my + 67*ly - (637*c3*s2*s4)/4 + 47;

    GMATRIX_DATA(MT,4,1) = 0;
    GMATRIX_DATA(MT,4,2) = 0;
    GMATRIX_DATA(MT,4,3) = 0;
    GMATRIX_DATA(MT,4,4) = 1;
}

//Calculo da cinematica inversa
void dirKinem(void)
{
    double y, x, a, b;
    GMATRIX_DATA(p,1,1) = GMATRIX_DATA(MT,1,4);
    GMATRIX_DATA(p,2,1) = GMATRIX_DATA(MT,2,4);
    GMATRIX_DATA(p,3,1) = GMATRIX_DATA(MT,3,4);

    y = GMATRIX_DATA(MT,2,1);
    x = GMATRIX_DATA(MT,1,1);
    GMATRIX_DATA(p,4,1) = atan2(y, x)/deg;

    y = -1*GMATRIX_DATA(MT,3,1);
    a = GMATRIX_DATA(MT,1,1);
    b = GMATRIX_DATA(MT,2,1);
    x = sqrt(a*a+b*b);
    GMATRIX_DATA(p,5,1) = atan2(y,x)/deg;

    y = GMATRIX_DATA(MT,3,2);
    x = GMATRIX_DATA(MT,3,3);
    GMATRIX_DATA(p,6,1) = atan2(y,x)/deg;
}

void PID(int i)
{
    if(i!=0)
        GMATRIX_COPY(ePrev,e);
    else
        GMATRIX_ZEROES(ePrev);
```

```
    GMATRIX_ZEROES(e);
    GMATRIX_SUBSTRACT_COPY(e,pTraj,p);

    //Proporcional
    GMATRIX_MULTIPLY_CONST(e,Kp);

    //Derivativo
    GMATRIX_SUBSTRACT_COPY(deriv,pTraj,ePrev);
    GMATRIX_MULTIPLY_CONST(deriv,Kd);
    GMATRIX_PRINT(deriv);

    GMATRIX_COPY(ePrev,pTraj);

    //Integrativo
    GMATRIX_DECLARE(temp,3,1);
    GMATRIX_MULTIPLY_CONST_COPY(temp,e,T);
    GMATRIX_ADD(integ,temp);
    GMATRIX_MULTIPLY_CONST(integ,Ki);

    //Soma
    GMATRIX_ADD_COPY(dp,e,deriv);
    GMATRIX_ADD(dp,integ);
}

//Discretização da trajetória
void trajectory(float tempo, float tempo_ciclo, int tipo)
{
    GMATRIX_COPY(pTemp,pTraj);

    if(tipo == 1)
    {
        GMATRIX_DATA(pTraj,1,1) = velocX*tempo_ciclo;
        GMATRIX_DATA(pTraj,2,1) = velocY*tempo_ciclo;
        GMATRIX_DATA(pTraj,3,1) = velocZ*tempo_ciclo;
    }

    if(tipo == 2)
    {
        GMATRIX_DATA(pTraj,1,1) = 0;
        GMATRIX_DATA(pTraj,2,1) = -raio*frequencia*(sin(frequencia*tempo))*
            tempo_ciclo;
        GMATRIX_DATA(pTraj,3,1) = -raio*frequencia*(cos(frequencia*tempo))*
            tempo_ciclo;
    }

    if(tempo==0)
        GMATRIX_ADD(pTraj,pInit);
    else
        GMATRIX_ADD(pTraj,pTemp);
}
```

```
int main(void)
{
    int flag;
    clock_t clk;
    double aux_tempo=0;

    GMATRIX_ZEROES(integ);
    FILE *fp;
    fp = fopen("C:\\Users\\Diogo\\Documents\\TG2\\Matlab\\array.m","w");

    calcSinCos();

    //Posição inicial
    GMATRIX_DATA(q,1,1) = 0*deg;
    GMATRIX_DATA(q,2,1) = 45*deg;
    GMATRIX_DATA(q,3,1) = 45*deg;
    GMATRIX_DATA(q,4,1) = 45*deg;
    GMATRIX_DATA(q,5,1) = -45*deg;
    GMATRIX_DATA(q,6,1) = 45*deg;
    GMATRIX_DATA(q,7,1) = 0*deg;

    GMATRIX_DATA(p2,1,1) = x;
    GMATRIX_DATA(p2,2,1) = y;
    GMATRIX_DATA(p2,3,1) = z;

    calcSinCos();
    transMatrix();
    dirKinem();
    GMATRIX_COPY(pInit,p);


    int i;

        clk = clock();
        for(i=0;i<n_ciclos;i++)
        {
            if(!linear_circular)
            {
                flag = linear();
                if(flag == 1)
                    return;
            }

            trajectory(aux_tempo,tempo_ciclo,linear_circular+1);
            PID(i);
            calcSinCos();
            calcJacobian();
            tempo_ciclo = (double) clk/CLOCKS_PER_SEC;
            GMATRIX_MULTIPLY_CONST(dq,tempo_ciclo);
            GMATRIX_ADD(q,dq);
            tempo_ciclo = (double) clk/CLOCKS_PER_SEC;
            transMatrix();
```

```
            dirKinem();
            fprintf(fp,"\nH1(%i) = %f",i+1,H1);
            fprintf(fp,"\nH2(%i) = %f",i+1,H2);
            GMATRIX_PRINT_ROBOT(q,i+1,fp);
            GMATRIX_PRINT_ROBOT(pTraj,i+1,fp);
            GMATRIX_PRINT_ROBOT(e,i+1,fp);
            usleep(900000);
            usleep(900000);
            usleep(900000);
            clk = clock() - clk;
            printf("tempo do ciclo: %f\n", tempo_ciclo);
            aux_tempo+=tempo_ciclo;
            fprintf(fp,"\nt(%d) = [%f]",(i+1),aux_tempo);
        }
    fclose(fp);
}

int linear(void)
{
 /*   GMATRIX_SUBSTRACT_COPY(dif,p2,pTraj);
    double maior, x, y, z;

    x = GMATRIX_DATA(dif,1,1);
    y = GMATRIX_DATA(dif,2,1);
    z = GMATRIX_DATA(dif,3,1);
    if(x>=y)
        maior=x;
    else
        maior=y;
    if(z>maior)
        maior=z;

    if(maior < 0.5 && maior > -0.5)
        return 1;

    tempo_total = maior/veloc_max;
    velocX = x/tempo_total;
    velocY = y/tempo_total;
    velocZ = z/tempo_total;*/

    velocX = 0;
    velocY = 0;
    velocZ = 0.000001;
    return 0;
}
```

III. CÓDIGO DE CONTROLE VIA *FPGA*

```
#include <stdio.h>
#include <string.h>
#include "system.h"
#include "math.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_uart_regs.h"
#include "altera_avalon_jtag_uart_regs.h"
#include "Gmatrix.h"
#include "funcoes.h"

        int valorPos[7][4];// = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        int valorVeloc[7][3];
        double T = 0.01;
    double Kp = 1;
    double Kd = 0;
    double Ki = 0;
    double deg = GMATRIXCONST_PI/180;
    double tempo_ciclo;
    double tempo_total;
    const double veloc_max = 1;
    double velocX, velocY, velocZ;
    double s1, s2, s3, s4, s5, s6, s7, c1, c2, c3, c4, c5, c6, c7;

    GMATRIX_DECLARE(q, 7, 1);
    GMATRIX_DECLARE(dq, 7, 1);
    GMATRIX_DECLARE(p, 3, 1);
    GMATRIX_DECLARE(dp, 3, 1);
    GMATRIX_DECLARE(pTraj,3,1);
    GMATRIX_DECLARE(pInit,3,1);
    GMATRIX_DECLARE(pTemp,3,1);
    GMATRIX_DECLARE(dif,3,1);
    GMATRIX_DECLARE(e,3,1);
    GMATRIX_DECLARE(ePrev,3,1);
    GMATRIX_DECLARE(deriv,3,1);
    GMATRIX_DECLARE(integ,3,1);
    GMATRIX_DECLARE(J, 3, 7);
    GMATRIX_DECLARE(Jinv, 7, 3);
    GMATRIX_DECLARE(Dummy, 3, 7);
    GMATRIX_DECLARE(MT, 4, 4);

    //Memória de posições
    GMATRIX_DECLARE(p1, 3, 1);
    GMATRIX_DECLARE(p2, 3, 1);

int main(void)
{
```

```
    int flag;
    int init=1;
GMATRIX_ZEROES(integ);
FILE *fp;

calcSinCos();

GMATRIX_DATA(q,1,1) = 0*deg;
GMATRIX_DATA(q,2,1) = 50*deg;
GMATRIX_DATA(q,3,1) = 0*deg;
GMATRIX_DATA(q,4,1) = 45*deg;
GMATRIX_DATA(q,5,1) = 0*deg;
GMATRIX_DATA(q,6,1) = 45*deg;
GMATRIX_DATA(q,7,1) = 0*deg;

envioPos();

calcSinCos();
transMatrix();
dirKinem();
GMATRIX_COPY(pInit,p);
GMATRIX_PRINT(q);
transMatrix();
dirKinem();
GMATRIX_PRINT(p);
int i;

envioPos();
    for(i=0;i<10000;i++)
    {
            posAtual();
        transMatrix();
        dirKinem();
        if(linear() ==1)
        {
                        trajectory(init,1);
                        PID(i);
                        calcSinCos();
                        calcJacobian();
                        GMATRIX_PRINT(p);
                        GMATRIX_PRINT(q);
                        GMATRIX_PRINT(dq);
                        GMATRIX_ADD(q,dq);
                        GMATRIX_PRINT(q);
                        //GMATRIX_PRINT_ROBOT(q,i+1,fp);
                        //GMATRIX_PRINT_ROBOT(pTraj,i+1,fp);
                        //GMATRIX_PRINT_ROBOT(e,i+1,fp);
                        envio();
                        init=0;
        }
        else
        {
```

```
                parar();
                init = 0;
            }
            printf("CICLO\n");
//              fprintf(fp,"\nt(%d) = [%f]",(i+1),aux_tempo);
        }
}

int linear(void)
{
        int direcao = IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE);
/*      int sentido = IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE);

        if(sentido%2 == 0)
                sentido = 1;
        else
                sentido = -1;

        if(direcao == 15)
        {
                velocX = 0;
            velocY = 0;
            velocZ = 0;
                return 0;
        }
        else
        {
                if(direcao== 6 || direcao == 7)
                {
                        velocX = 1*sentido;
                    velocY = 0;
                    velocZ = 0;
                }
                if(direcao== 10 || direcao == 11)
                {
                        velocX = 0;
                    velocY = 1*sentido;
                    velocZ = 0;
                }
                if(direcao== 12 || direcao == 13)
                {
                        velocX = 0;
                    velocY = 0;
                    velocZ = 1*sentido;
                }
                return 1;
        }*/

        if(direcao != 15)
        {
                velocX = 0;
                velocY = 0;
```

```
                velocZ = -10;
                return 1;
        }
        else
                return 0;

}

void trajectory(int i, int tipo)
{
    GMATRIX_COPY(pTemp,pTraj);

    if(tipo == 1)
    {
        GMATRIX_DATA(pTraj,1,1) = velocX;
        GMATRIX_DATA(pTraj,2,1) = velocY;
        GMATRIX_DATA(pTraj,3,1) = velocZ;
    }

    if(tipo == 2)
    {
        GMATRIX_DATA(pTraj,1,1) = 0;
        GMATRIX_DATA(pTraj,2,1) = -0.002*(sin(0.0006*i));
        GMATRIX_DATA(pTraj,3,1) = -0.002*(cos(0.0006*i));
    }

    if(i==0)
        GMATRIX_ADD(pTraj,pInit);
    else
        GMATRIX_ADD(pTraj,pTemp);
}

void PID(int i)
{           //USAR METODO DE NEWTON!!!!!!!!!!!!!!!!!!! (trabalho: falar sobre
    métodos numéricos e método da descida de gradiente)
    if(i!=0)
        GMATRIX_COPY(ePrev,e);
    else
        GMATRIX_ZEROES(ePrev);

    GMATRIX_ZEROES(e);
    GMATRIX_SUBSTRACT_COPY(e,pTraj,p);
//      GMATRIX_PRINT(pTraj);
//      GMATRIX_PRINT(p);
    GMATRIX_PRINT(pTraj);
    GMATRIX_PRINT(p);
     GMATRIX_PRINT(e);

    //Proporcional
    GMATRIX_MULTIPLY_CONST(e,Kp);

    //Derivativo
```

```
    GMATRIX_SUBSTRACT_COPY(deriv,pTraj,ePrev);
    GMATRIX_MULTIPLY_CONST(deriv,Kd);
//      GMATRIX_PRINT(deriv);

    GMATRIX_COPY(ePrev,pTraj);

    //Integrativo
    GMATRIX_DECLARE(temp,3,1);
    GMATRIX_MULTIPLY_CONST_COPY(temp,e,T);
    GMATRIX_ADD(integ,temp);
    GMATRIX_MULTIPLY_CONST(integ,Ki);

    //Soma
    GMATRIX_ADD_COPY(dp,e,deriv);
    GMATRIX_ADD(dp,integ);
}

void calcSinCos(void)
{
    double t1, t2, t3, t4, t5, t6, t7;

    t1 = GMATRIX_DATA(q,1,1);
    t2 = GMATRIX_DATA(q,2,1);
    t3 = GMATRIX_DATA(q,3,1);
    t4 = GMATRIX_DATA(q,4,1);
    t5 = GMATRIX_DATA(q,5,1);
    t6 = GMATRIX_DATA(q,6,1);
    t7 = GMATRIX_DATA(q,7,1);

    s1 = sin(t1);
    s2 = sin(t2);
    s3 = sin(t3);
    s4 = sin(t4);
    s5 = sin(t5);
    s6 = sin(t6);
    s7 = sin(t7);

    c1 = cos(t1);
    c2 = cos(t2);
    c3 = cos(t3);
    c4 = cos(t4);
    c5 = cos(t5);
    c6 = cos(t6);
    c7 = cos(t7);
}

void calcJacobian(void)
{
//      GMATRIX_DECLARE(Jtransp, 3, 7);

    //Variáveis para evitar a repetição desnecessária de cálculos
    double gy = (c1*s3 + c2*c3*s1);
```

```
double by = c2*s3*s5;
double ay = c5*s1*s2*s3;
double d = (c4*gy - s1*s2*s4);
double lx = c1*s2*s3*s5;
double kx = c1*c3*s2*s4;
double jx = c6*s2*s3*s4;
double ix = s1*s2*s3*s5;
double gx = c2*c3*s4;
double fx = c1*c2*c4;
double dx = c4*s2;
double b = (s4*gy + c4*s1*s2);
double c = (c3*s1 + c1*c2*s3);
double e = (s1*s3 - c1*c2*c3);
double a = (c4*e + c1*s2*s4);
double f = (s5*e - c4*c5*c);
double g = c1*c2*s4;
double h = (c1*c3 - c2*s1*s3);
double r = (s4*e - c1*dx);
double ex = (c5*a + s5*c);
double az = s6*ex;
double bz = c6*r;
double hx = (c5*d + s5*h);
double cz = (c6*b + s6*hx);
double dz = (s5*a - c5*c);
double ez = (s5*d - c5*h);
double i = (c7*(bz + az) + s7*dz);
double j = (c7*cz + s7*ez);
double ax = (c2*s4 + c3*dx);
double sx = (c5*ax - s2*s3*s5);
double vx = (c2*c4 - c3*s2*s4);
double fy = (s5*ax + c5*s2*s3);
double ly = c6*vx;
double my = (s6*sx - ly);
double n = (c6*sx + s6*vx);
double o = (s2*s4 - c2*c3*c4);
double tx = (c2*s1*s4 + c3*c4*s1*s2);
double ux = (c2*c4*s1 - c3*s1*s2*s4);
double yx = (c3*s2*s5 + c4*c5*s2*s3);
double bx = (c3*c5*s2 - dx*s3*s5);
double cx = c1*c3*dx;
double cy = (c5*o + by);
double dy = (s5*o - c2*c5*s3);
double ey = c6*(dx + gx);
double hy = (s5*tx + ay);
double iy = (c6*ax + c5*s6*vx);
double jy = (s6*f - c6*s4*c);
double ky = (s6*(c5*tx - ix) - c6*ux);
double ny = (s6*cy - ey);

GMATRIX_DATA(J,1,1) = (1543*s1*s2)/10 + 67*c6*b + 83*c7*cz + 67*s6*hx + 83*s7*
    ez + (637*s4*gy)/4 + (637*c4*s1*s2)/4;
GMATRIX_DATA(J,1,2) = 67*s6*(c5*(g + cx) - lx) - 67*c6*(fx - kx) - 83*c7*(c6*(
```

```
    fx - kx) - s6*(c5*(g + cx) - lx)) - (1543*c1*c2)/10 + 83*s7*(s5*(g + cx) +
    c1*c5*s2*s3) - (637*fx)/4 + (637*kx)/4;
GMATRIX_DATA(J,1,3) = (637*s4*c)/4 - 83*c7*jy - 67*s6*f + 83*s7*(c5*e + c4*s5*c
    ) + 67*c6*s4*c;
GMATRIX_DATA(J,1,4) = 83*c7*(c6*a - c5*s6*r) + 67*c6*a + (637*c4*e)/4 + (637*c1
    *s2*s4)/4 - 67*c5*s6*r - 83*s5*s7*r;
GMATRIX_DATA(J,1,5) = 83*s7*ex - 67*s6*dz - 83*c7*s6*dz;
GMATRIX_DATA(J,1,6) = 67*c6*ex - 83*c7*(s6*r - c6*ex) - 67*s6*r;
GMATRIX_DATA(J,1,7) = 83*c7*dz - 83*s7*(bz + az);

GMATRIX_DATA(J,2,1) = 67*bz - (1543*c1*s2)/10 + 83*c7*(bz + az) + 67*az + 83*s7
    *dz + (637*s4*e)/4 - (637*c1*dx)/4;
GMATRIX_DATA(J,2,2) = 67*s6*(c5*tx - ix) - (1543*c2*s1)/10 + 83*s7*hy + 83*c7*
    ky - 67*c6*ux - (637*c2*c4*s1)/4 + (637*c3*s1*s2*s4)/4;
GMATRIX_DATA(J,2,3) = 83*c7*(s6*(s5*gy - c4*c5*h) - c6*s4*h) - (637*s4*h)/4 +
    67*s6*(s5*gy - c4*c5*h) - 83*s7*(c5*gy + c4*s5*h) - 67*c6*s4*h;
GMATRIX_DATA(J,2,4) = (637*s1*s2*s4)/4 - 67*c6*d - (637*c4*gy)/4 - 83*c7*(c6*d
    - c5*s6*b) + 67*c5*s6*b + 83*s5*s7*b;
GMATRIX_DATA(J,2,5) = 67*s6*ez - 83*s7*hx + 83*c7*s6*ez;
GMATRIX_DATA(J,2,6) = 67*s6*b + 83*c7*(s6*b - c6*hx) - 67*c6*hx;
GMATRIX_DATA(J,2,7) = 83*s7*cz - 83*c7*ez;

GMATRIX_DATA(J,3,1) = 0;
GMATRIX_DATA(J,3,2) = 67*s6*cy - (637*dx)/4 - (1543*s2)/10 + 83*s7*dy + 83*c7*
    ny - 67*ey - (637*gx)/4;
GMATRIX_DATA(J,3,3) = 83*c7*(s6*yx + jx) + 67*s6*yx - 83*s7*bx + (637*s2*s3*s4)
    /4 + 67*jx;
GMATRIX_DATA(J,3,4) = - (637*c2*s4)/4 - 67*c6*ax - 83*c7*iy - 67*c5*s6*vx - 83*
    s5*s7*vx - (637*c3*dx)/4;
GMATRIX_DATA(J,3,5) = 67*s6*fy - 83*s7*sx + 83*c7*s6*fy;
GMATRIX_DATA(J,3,6) = - 67*c6*sx - 83*c7*n - 67*s6*vx;
GMATRIX_DATA(J,3,7) = 83*s7*my - 83*c7*fy;

//    GMATRIX_TRANSPOSE_COPY(Jtransp, J);
//
//    GMATRIX_MULTIPLY_ADD(Jtransp,J,Jtransp);
//    int i=GMATRIX_RANK(Jtransp);
//    printf("==============> %d\n",i);

GMATRIX_PSEUDOINVERSE(Jinv, J, Dummy); //se matriz não for singular usar
    gmatrix_pseudoinverse_copy pois tem um custo muito menor

GMATRIX_MULTIPLY_COPY(dq, Jinv, dp);

GMATRIX_DECLARE(JJ,7,7);
GMATRIX_DECLARE(temp,7,7);
GMATRIX_DECLARE(temp2,7,1);
GMATRIX_DECLARE(grad,7,1);
GMATRIX_IDENTITY(temp);
GMATRIX_MULTIPLY_COPY(JJ, Jinv, J);
GMATRIX_SUBSTRACT(temp, JJ);
GMATRIX_COPY(grad,q);
```

```
    GMATRIX_MULTIPLY_CONST(grad,(-0.1));
    GMATRIX_MULTIPLY_COPY(temp2, temp, grad);
    GMATRIX_ADD(dq,temp2);

}

void transMatrix(void)
{
    double gy = (c1*s3 + c2*c3*s1);
    double dx = c4*s2;
    double b = (s4*gy + c4*s1*s2);
    double c = (c3*s1 + c1*c2*s3);
    double d = (c4*gy - s1*s2*s4);
    double e = (s1*s3 - c1*c2*c3);
    double a = (c4*e + c1*s2*s4);
    double h = (c1*c3 - c2*s1*s3);
    double r = (s4*e - c1*dx);
    double ex = (c5*a + s5*c);
    double hx = (c5*d + s5*h);
    double ax = (c2*s4 + c3*dx);
    double fy = (s5*ax + c5*s2*s3);
    double vx = (c2*c4 - c3*s2*s4);
    double sx = (c5*ax - s2*s3*s5);
    double ly = c6*vx;
    double my = (s6*sx - ly);
    double az = s6*ex;
    double bz = c6*r;
    double cz = (c6*b + s6*hx);
    double dz = (s5*a - c5*c);
    double ez = (s5*d - c5*h);

    GMATRIX_DATA(MT,1,1) = c7*(bz + az) + s7*dz;
    GMATRIX_DATA(MT,1,2) = c6*ex - s6*r;
    GMATRIX_DATA(MT,1,3) = s7*(bz + az) - c7*dz;
    GMATRIX_DATA(MT,1,4) = 67*bz - (1543*c1*s2)/10 + 83*c7*(bz + az) + 67*az + 83*
        s7*dz + (637*s4*e)/4 - (637*c1*dx)/4;

    GMATRIX_DATA(MT,2,1) = - c7*cz - s7*ez;
    GMATRIX_DATA(MT,2,2) = s6*b - c6*hx;
    GMATRIX_DATA(MT,2,3) = c7*ez - s7*cz;
    GMATRIX_DATA(MT,2,4) = - (1543*s1*s2)/10 - 67*c6*b - 83*c7*cz - 67*s6*hx - 83*
        s7*ez - (637*s4*gy)/4 - (637*c4*s1*s2)/4;

    GMATRIX_DATA(MT,3,1) = - s7*fy - c7*my;
    GMATRIX_DATA(MT,3,2) = - c6*sx - s6*vx;
    GMATRIX_DATA(MT,3,3) = c7*fy - s7*my;
    GMATRIX_DATA(MT,3,4) = (1543*c2)/10 + (637*c2*c4)/4 - 67*s6*sx - 83*s7*fy - 83*
        c7*my + 67*ly - (637*c3*s2*s4)/4 + 47;

    GMATRIX_DATA(MT,4,1) = 0;
    GMATRIX_DATA(MT,4,2) = 0;
    GMATRIX_DATA(MT,4,3) = 0;
```

```
    GMATRIX_DATA(MT,4,4) = 1;
}

void dirKinem(void)
{
    double y, x, a, b;
    GMATRIX_DATA(p,1,1) = GMATRIX_DATA(MT,1,4);
    GMATRIX_DATA(p,2,1) = GMATRIX_DATA(MT,2,4);
    GMATRIX_DATA(p,3,1) = GMATRIX_DATA(MT,3,4);

/*     y = GMATRIX_DATA(MT,2,1);
    x = GMATRIX_DATA(MT,1,1);
//     printf("roll: y = %f, x = %f, atan = %f\n",y,x,atan2(y,x));
    GMATRIX_DATA(p,4,1) = atan2(y, x)/deg;

    y = -1*GMATRIX_DATA(MT,3,1);
    a = GMATRIX_DATA(MT,1,1);
    b = GMATRIX_DATA(MT,2,1);
    x = sqrt(a*a+b*b);
//     printf("pitch: y = %f, x = %f\n",y,x);
    GMATRIX_DATA(p,5,1) = atan2(y,x)/deg;

    y = GMATRIX_DATA(MT,3,2);
    x = GMATRIX_DATA(MT,3,3);
//     printf("yaw: y = %f, x = %f\n",y,x);
    GMATRIX_DATA(p,6,1) = atan2(y,x)/deg;*/
}

void conversor(int i)
{
        GMATRIX_ADD(q,dq);
        double x = GMATRIX_DATA(q,i+1,1);
        int y = graus2passos(x);
//      printf("y: %f\n",y);
        valorPos[i][0] = y/1000;
        valorPos[i][1] = (y%1000)/100;
        valorPos[i][2] = ((y%1000)%100)/10;
        valorPos[i][3] = ((y%1000)%100)%10;


        x = GMATRIX_DATA(dq,i+1,1);
        y = graus2passos(x);
        y = y*10;
//      printf("Velocidade %d: %d\n",i,y);
        y = y*10;
        valorVeloc[i][0] = y/100;
        valorVeloc[i][1] = (y%100)/10;
        valorVeloc[i][2] = (y%100)%10;

}

void envio()
```

```
{
        printf("\nEnviar!\n");
        int i;
        for(i = 0; i <7; i++)
        {
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,35);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,i+48);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,80);
        usleep(3500);

        conversor(i);
        if(valorPos[0]!=0)
        {
                IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][0]+48);
                usleep(3500);
        }

        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][1]+48);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][2]+48);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][3]+48);
        usleep(3500);

        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,83);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,48);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020, valorVeloc[i][0]+48);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorVeloc[i][1]+48);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorVeloc[i][2]+48);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,13);
        usleep(4000);
        }
}

void envioPos()
{
        printf("\nEnvio\n");
        int i;
        for(i = 0; i <7; i++)
        {
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,35);
        usleep(5000);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,i+48);
        usleep(5000);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,80);
```

```
        usleep(5000);

        conversor(i);
        if(valorPos[0]!=0)
        {
                IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][0]+48);
                usleep(5000);
        }

        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][1]+48);
        usleep(5000);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][2]+48);
        usleep(5000);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,valorPos[i][3]+48);
        usleep(5000);

        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,13);
        usleep(5000);
        }
}


int graus2passos(double alpha)
{
        int p;
//      printf("a: %f\n",alpha);
        p = (int) 1500+alpha*1500/(170*deg); //170ř equivalente a 1500 passos
//      printf("passo: %d\n",p);

        return p;
}

void posAtual()
{
//QP <arg> <cr>
        int i, x;
        x=0;

        for(i=0;i<7;i++)
        {
                        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,81);
                        usleep(3500);
                        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,80);
                        usleep(3500);
                        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,48+i);
                        usleep(3500);
                        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,13);
                        usleep(3500);

                        x = IORD_ALTERA_AVALON_UART_RXDATA(0x2020);
                        usleep(3500);
                        x = x*10;
```

```
//                  printf("==> %d\n",x);
                    if(x!=0)
                            passos2graus(x, i);
        }
}

void passos2graus(int p, int i)
{
        double x;
        x = (((p-1500)*170*deg)/1500);
        printf("angulo %d: %d  ==>  %f\n",i,p,x);
        GMATRIX_DATA(q,i,1) = x;

}

void parar()//83 84 79 80
{
        printf("\nParar\n");
        int i;
        for(i = 0; i <7; i++)
        {
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,83);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,84);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,79);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,80);
        usleep(3500);
        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,48+i);
        usleep(3500);

        IOWR_ALTERA_AVALON_UART_TXDATA(0x2020,13);
        usleep(4000);
        }
}
```