UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

DESENVOLVIMENTO DO PLANEJADOR DE TRAJETÓRIA E DO SISTEMA DE CONTROLE EM MALHA ABERTA DE UM MANIPULADOR ROBÓTICO DE GEOMETRIA ESFÉRICA, EMBARCADOS EM UMA PLATAFORMA FPGA

ÊNIO PRATES VASCONCELOS FILHO

ORIENTADOR: CARLOS HUMBERT LLANOS QUINTERO
CO-ORIENTADOR: GUILHERME CARIBÉ DE CARVALHO

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

PUBLICAÇÃO: ENM.DM – 18A/12
BRASÍLIA/DF: FEVEREIRO – 2012

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

DESENVOLVIMENTO DO PLANEJADOR DE TRAJETÓRIA E DO SISTEMA DE CONTROLE EM MALHA ABERTA DE UM MANIPULADOR ROBÓTICO DE GEOMETRIA ESFÉRICA, EMBARCADOS EM UMA PLATAFORMA FPGA

ÊNIO PRATES VASCONCELOS FILHO

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISÍTOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS.

APROVADA POR:

_____

PROF. DR CARLOS HUMBERTO LLANOS QUINTERO (ENM-UnB)
(Orientador)

_____

PROF. DR. JOSÉ MAURÍCIO S. T. DA MOTTA (ENM-UnB)
(Examinador Interno)

_____

PROF. DR. EDWARD D. MORENO (UFSE)
(Examinador Externo)

BRASÍLIA/DF, 24 DE AGOSTO DE 2012

FICHA CATALOGRÁFICA

VASCONCELOS FILHO, ÊNIO PRATES
DESENVOLVIMENTO DO PLANEJADOR DE TRAJETÓRIA E DO SISTEMA DE CONTROLE EM MALHA ABERTA DE UM MANIPULADOR ROBÓTICO DE GEOMETRIA ESFÉRICA, EMBARCADOS EM UMA PLATAFORMA FPGA
[Distrito Federal] 2012.
xvii, 155p., 210 x 297 mm (ENC/FT/UnB, Mestre, Sistemas Mecatrônicos, 2012).
Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.
Departamento de Engenharia Mecânica.
1.Robótica 2.*FPGA*
3.Manipuladores Robóticos 4.Eletrônica Embarcada
I. ENC/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

VASCONCELOS FILHO, E. P. (2012). Desenvolvimento do planejador de trajetória e do sistema de controle em malha aberta de um manipulador robótico de geometria esférica, embarcados em uma plataforma FPGA.
Publicação ENM.DM-53A/12, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 155p.

CESSÃO DE DIREITOS

AUTOR: Ênio Prates Vasconcelos Filho.
TÍTULO: Desenvolvimento do planejador de trajetória e do sistema de controle em malha aberta de um manipulador robótico de geometria esférica, embarcados em uma plataforma FPGA.
GRAU: Mestre ANO: 2012

É concedida à Universidade de Brasília permissão para reproduzir cópias desta dissertação de mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte dessa dissertação de mestrado pode ser reproduzida sem autorização por escrito do autor.

_____

Ênio Prates Vasconcelos Filho
SQN 408, Bl J, apt 303
70856-100 Brasília – DF – Brasil.

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais, que me incentivaram e amaram durante todo o tempo e me deram condições de chegar até aqui. Que souberam me guiar e me ensinaram a ser quem sou hoje.

A minha irmã que acredita mais em mim do que eu mesmo.

A toda a minha família que está distante fisicamente, mas sempre no coração, torcendo ensinando e acreditando.

Ao professor Llanos, que sempre deu condições e apoio, principalmente nas horas difíceis. E que também não desistiu apesar de todas as dificuldades.

Ao professor Guilherme Caribé, por sua paciência em corrigir e recorrigir algumas coisas.

Aos colegas do Graco, que compartilharam os momentos de desespero e de alegria ao longo desse período. Assim como aos colegas da Innovix que acompanharam as mals dormidas noites e a expectativa da entrega.

Aos amigos, que tornaram a vida com certeza mais divertida e animada nesses tempos de mestrado.

Aos companheiros do Futebolzinho, que não me deixaram ficar tão estressado quanto deveria.

Agradeço a Francieli, amor que encontrei ao longo do mestrado e que levarei para todo o sempre, ao apoio e força de vontade que me impulsionou a chegar até aqui.

A Deus, pelo dom da vida e pela capacidade de continuar lutando.

E a todos os outros que apoiaram, acreditaram e que de alguma forma fizeram parte dessa caminhada.

Ênio Prates Vasconcelos Filho

DEDICATÓRIA

Dedico esse trabalho aos meus pais, irmã e a toda a minha família.
Também, aos amigos que acreditaram e que torceram sempre.
E principalmente a Francieli,amor da minha vida,
que não me deixou esquecer o quanto é importante sempre seguir em frente.

Ênio Prates Vasconcelos Filho

RESUMO

Esse trabalho descreve o desenvolvimento e a implementação de um controlador de trajetória retilínea em um robô esférico de 5 graus de liberdade. Para tanto, foi desenvolvida uma arquitetura de controle em malha aberta embarcada em uma *FPGA* para os três primeiros graus de liberdade do manipulador. Nesse intuito, apresenta-se, nesse trabalho, a modelagem cinemática direta e inversa do manipulador, bem como seu Jacobiano. Essa modelagem permite o controle da trajetória do robô em um caminho retilíneo descrito em coordenadas cartesianas. Na implementação do controle embarcado na *FPGA*, foi utilizado o microprocessador *NIOS II*, da *Altera*. Esse é o responsável pelos cálculos de posicionamento e velocidade do manipulador durante sua movimentação. Também são explicitadas as interfaces de acionamento e controle de cada um dos eixos do manipulador e seus respectivos motores. São ainda apresentadas as experiências de validação dos algoritmos implementados, através de simulações computacionais, bem como a validação das equações utilizadas. Além disso, são apresentados os resultados de movimentação do manipulador, seguindo uma trajetória pré-estabelecida, buscando validar na prática o controle implementado.

ABSTRACT

This paper describes the development and implementation of a controller for straight path trajectory in a spherical robot of five degrees of freedom. To do that, an open loop control architecture (embedded in an FPGA) was developed, for the first three degrees of freedom of the manipulator. Therefore, the direct and inverse kinematic models of the manipulator as well as its Jacobian are presented in this work. This modeling allows us to control the trajectory of the robot in a straight path described in Cartesian coordinates. In the implementation of the embedded controller in the FPGA, we have used the *NIOS* II microprocessor, from Altera. This is responsible for calculating the position and speed of the manipulator during its motion. Also the interfaces with the controllers of each axis of the handler and their respective engines are specified. We also present experiments to validate the implemented algorithms through computer simulations, as well as the validation of the equations used. Finally, the results are presented of the manipulator motion, following a predetermined path, in order to validate the control implemented in practice.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS E ABREVIATURAS

a	- Aceleração do corpo
ASIC	- *Application Specific Integrated Circuit*
AWO	- *All Windings Off Input*
CI	- Circuito Integrado
CM	- Controle de Movimentação.
D3	- Coordenada da junta 3 do manipulador;
DIR	- *Direction Input*
DSP	- *Digital Signal Processor*.
DSP	- *Digital Signal Processor*
EDA	- *Electronic Design Automation*
EURON	- *European Robotics Research Network*
F	- Força
FPGA	- *Field Programmable Gate Array*.
g	- Aceleração da Gravidade
GS	- Gerenciamento de Soldagem.
I/O	- *Input/Output*
IP	- *Intelectual Property*.
IREQ	- *Hydro-Quebec Research Institute*
ISO	- *International Organization for Standarization*
Iα	- Momento de inércia
JIRA	- *Japanese Industrial Robot Association*.
LE	- *Logical Elements*
M	- Massa
NIOS	- Microprocessador da Altera.
OPTO	- *Opto Isolator Supply*
P	- *Pitch*
PC	- Computador
PLL	- *Phase Locked Loop*
PULSE	- *Pulse Input*
R	- Raio
R	- *Roll*
RDCE	- *Reduce Current Input*

RIA	- *Robot Institute of America*.
Rx	- Coordenada cartesiana da rotação em torno do eixo x
Ry	- Coordenada cartesiana da rotação em torno do eixo y
Rz	- Coordenada cartesiana da rotação em torno do eixo z
SCARA	- *Selective Compliance Assembly Robot Arm*
SOPC Builder	- *System on Programmable chip*
T	- Tração
TEI	- Tempo entre interações
Tx	- Coordenada cartesiana da posição sobre o eixo x
Ty	- Coordenada cartesiana da posição sobre o eixo y
Tz	- Coordenada cartesiana da posição sobre o eixo z
VC	- Visão computacional.
Y	-*Yaw*
Θ1	- Coordenada da junta 1 do manipulador;
Θ2	- Coordenada da junta 2 do manipulador;
Θ4	- Coordenada da junta 4 do manipulador;
Θ5	- Coordenada da junta 5 do manipulador;

1 INTRODUÇÃO

Cada vez mais, busca-se automatizar atividades repetitivas, que consomem longos prazos de execução ou que são executadas em condições insalubres. Tratam-se, aqui, de operações diversas, que demandam pouco ou nada da criatividade humana, necessitando apenas do conhecimento dos procedimentos para repetição. Com base nessa ideia, diversos setores já vêm utilizando robôs para desempenhar esse tipo de função, garantindo, assim, um padrão de qualidade e, mantendo seu capital humano investido em operações que realmente demandem criatividade e capacidade de raciocínio.

Dentre essas atividades, encontram-se alguns processos de soldagem. Pode-se observar, em inúmeras indústrias automotivas (Jimenez & Almeida, 1998), por exemplo, que o processo de soldagem é realizado, em sua maior parte, por robôs especializados, o que aumenta sua capacidade produtiva. Esses robôs realizam atividades como soldagem, montagem, testes de resistência, entre outras. Tais atividades demandam precisão, confiabilidade e repetibilidade. Desse modo, o processo de produção e montagem dos automóveis tem seu custo e tempo reduzidos, uma vez que, utilizando os robôs é possível produzir em larga escala num espaço menor de tempo.

Outro aspecto importante no processo de automação das indústrias refere-se à repetibilidade do processo. Isso ocorre porque uma atividade como a soldagem, por exemplo, ao ser realizada pelo homem pode apresentar qualidade variável, entre uma peça e outra (KAPP, 2000).

É nesse contexto que se encontra o projeto financiado pelas Centrais Elétricas do Norte do Brasil – Eletronorte -, em parceria com a Universidade de Brasília – UnB –, por meio dos laboratórios de Robótica e Soldagem Robotizada do Grupo de controle e Automação em Processos de Fabricação – Graco.

O problema foco deste projeto de pesquisa é a automatização dos procedimentos de recuperação de pás de rotores de turbinas hidráulicas, danificadas por fenômenos de cavitação e fadiga, por meio de processos de soldagem. A Figura **1.1** mostra um exemplo de pá de turbina.

Figura 1.1: Exemplo de pá de turbina

Os rotores das turbinas da maior parte das usinas hidrelétricas do Brasil estão sujeitos à erosão provocada pelo fenômeno da cavitação. A cavitação ocorre nos locais onde a pressão da água dentro da turbina atinge a pressão de vapor do líquido. As bolhas de ar contidas na água ao chegarem nestas áreas de alta pressão se comprimem até implodirem (LEAL, 2005). A energia liberada pela implosão das bolhas é suficiente para danificar as paredes metálicas do rotor, facilitando assim a erosão. Devido ao desgaste causado pela cavitação, que ocasiona furos e trincas, são necessárias paradas periódicas para reparos das pás do rotor (CONTO, 2003).

Atualmente, durante as manutenções programadas para realizar esses reparos, é selecionada uma equipe de soldagem para localizar e tratar este tipo de problema. Tal recuperação é feita através da deposição de material por soldagem na cratera, buscando-se, assim, recompor o perfil geométrico dela. Normalmente, a operação demanda muitas horas de trabalho, durante as quais a turbina permanece desligada. Quanto mais tempo ela passa desligada, maior o prejuízo da empresa. Além disso, o ambiente é bastante úmido e escuro, o que o caracteriza como insalubre para o trabalho humano.

No contexto supracitado, surgiu a demanda pela concepção e construção de um robô capaz de realizar esse tipo de operação, de maneira mais rápida, eficiente e padronizada. Com isso, busca-se uma economia de tempo e de recursos durante o processo. Além disso, busca-se aumentar o tempo entre as manutenções, ao aumentar a qualidade dos reparos.

Em outros trabalhos de pesquisa e desenvolvimentos semelhantes, algumas empresas já desenvolvem robôs com essa finalidade. É o caso da canadense *Hydro-Quebec Research Institute* (IREQ), que, em parceria com universidades locais, desenvolveu o *Scompi*.

No Brasil, para a mesma finalidade, foi desenvolvido o *Roboturb*, por meio de uma parceria entre a Universidade de Santa Catarina (UFSC) e o Instituto de Tecnologia para o Desenvolvimento (LACTEC) do Paraná.

No projeto em que se insere esta dissertação, desenvolvido pela UNB, em parceria com a Eletronorte, busca-se desenvolver um manipulador com finalidade semelhante aos citados anteriormente, melhorando a produtividade no processo de reparo, ao diminuir o tempo de execução da manutenção e aumentando sua qualidade. Contudo, existe ainda uma característica diferente em relação aos anteriores, uma vez que se busca desenvolver um manipulador que funcione de maneira autônoma, capaz de encontrar o dano, analisá-lo e realizar o reparo automaticamente.

1.1 DESCRIÇÃO

O projeto desse robô consiste na união de três módulos principais, que podem ser vistos na Figura 1.2. Estes são descritos a seguir:

- *Visão Computacional* (VC): sistema capaz de realizar uma varredura sobre a superfície da pá da turbina e produzir uma reconstrução virtual da superfície danificada, a ser tratada;
- *Gerenciamento de Soldagem* (GS): sistema responsável por analisar a superfície gerada pelo sistema VC e determinar a melhor estratégia de soldagem, para reparo da turbina. Além disso, esse módulo é responsável por determinar os parâmetros de soldagem, como velocidade de movimentação e corrente a ser utilizada pela fonte de soldagem;
- *Controle de movimentação* (CM): Sistema que tem capacidade de receber do GS os parâmetros da soldagem, as posições de início e fim dos cordões de solda e mover-se de acordo com a trajetória definida;

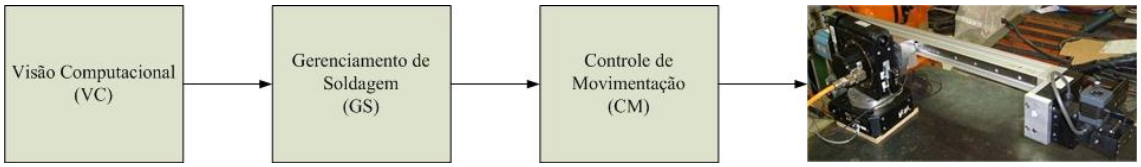


Figura 1.2: Visão geral do projeto

Neste trabalho, trata-se especificamente do Controle de Movimentação. Os outros dois sistemas também estão sendo desenvolvidos na UnB, contudo, serão objetos de outros trabalhos.

É importante citar que o módulo de controle de movimentação e suas ferramentas estão sendo projetados para um manipulador responsável pela execução de soldagem de turbinas danificadas por cavitação. Contudo, o sistema pode ser modificado para a realização de outros tipos de processos de soldagem, caso necessário.

Outro importante aspecto desse trabalho é a utilização de uma *FPGA* como controladora do processo. A utilização dessa tecnologia, além de ser uma alternativa aos problemas provocados pela arquitetura de *von Neumann*, apresenta a flexibilidade como característica marcante. Isso porque uma mudança no projeto de *hardware* pode ser feita internamente à *FPGA*, sem demandar a reconstrução da placa ou do *hardware* externo a ela.

1.2 METODOLOGIA

A metodologia adotada neste trabalho envolve o desenvolvimento das equações resultantes da modelagem cinemática direta e inversa do manipulador, incluindo a transformação de velocidades, de modo a possibilitar o controle do manipulador no espaço cartesiano. A partir dos modelos desenvolvidos, desenvolver-se-ão a arquitetura de *hardware*, as estratégias de programação e os algoritmos específicos para implementação do controlador em um *FPGA*. Incluir-se-ão nessa arquitetura as interfaces de comunicação necessárias ao controlador para permitir seu interfaceamento com os diversos sistemas externos ao controlador.

Uma visão integrada dos elementos desenvolvidos nesse trabalho, com os módulos externos, pertencentes ao projeto pode ser vista na Figura 1.3.

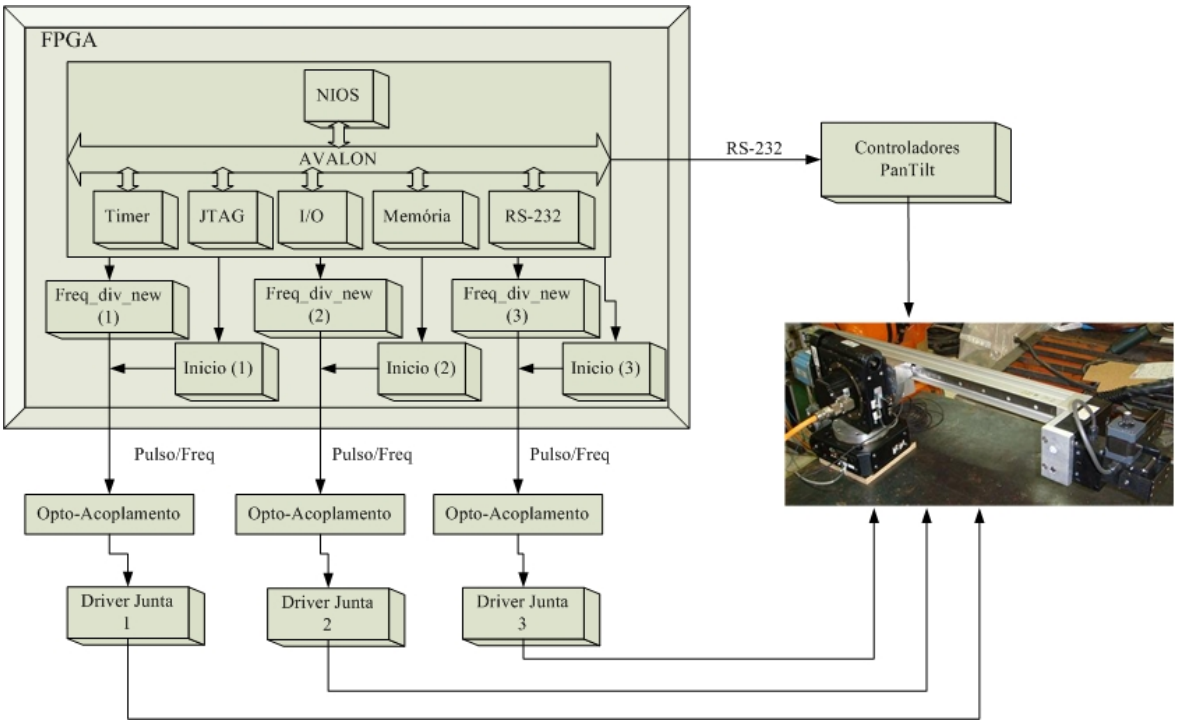


Figura 1.3: Arquitetura a ser implementada

1.3 OBJETIVOS

1.3.1 Objetivos gerais

Esse trabalho possui, por objetivo principal, a concepção de um sistema controlador, capaz de receber dados indicando pontos no espaço, tratar essas informações, determinar trajetórias retilíneas entre eles, e atuar sobre as juntas do manipulador de modo a garantir a correção dessa trajetória.

Além disso, busca-se demonstrar a flexibilidade e a facilidade de trabalho resultante da utilização de *hardware* reconfigurável para uma aplicação complexa, que envolve uma grande quantidade de cálculos em tempo real. Deseja-se ainda mostrar a viabilidade da utilização do modelo conhecido de programação sequencial, associada ao microprocessador embarcado na *FPGA*, e ainda ter acesso às vantagens do uso da *FPGA*, para aceleração de processos críticos ou mesmo para fazer a interface com os periféricos de maneira mais direta.

1.3.2 Objetivos específicos

Além dos objetivos gerais explicitados no item 1.3.1, estabelecem-se como objetivos específicos os enumerados a seguir:

a) Demonstrar a viabilidade da aplicação de *FPGAs* no contexto da robótica;
b) Dar continuidade à ideia de desenvolver um sistema embarcado na *FPGA* envolvendo um processador *NIOS*, resolvendo funções de alto nível, em parceria com outras estruturas embarcadas também na *FPGA*, de modo a alcançar maior eficiência;
c) Desenvolver ferramentas que validem os modelos cinemáticos elaborados, tornando mais rápida a descoberta e solução de problemas nestes modelos;
d) Prover comunicação de modo eficiente entre os diversos módulos do projeto (internos e externos);
e) Desenvolver um estudo a respeito de manipuladores e robôs industriais que permitam a concepção de outros projetos;

1.4 MOTIVAÇÃO

A grande motivação para o presente trabalho refere-se à construção de um *hardware* embarcado numa *FPGA* capaz de controlar um processo de alta precisão e de grande complexidade.

Desse modo, busca-se também analisar a viabilidade da utilização de *FPGAs* associadas à robótica, visando aumentar a eficiência do processo de controle dos robôs.

Deseja-se, também, estudar uma alternativa à programação normalmente utilizada, desenvolvida com a utilização de microprocessadores e PCs industriais.

Além disso, a utilização da *FPGA* como plataforma de controle do manipulador permite a mudança do *hardware* utilizado no projeto sem alterações físicas, demandando apenas uma reprogramação deste.

Uma motivação a mais para o desenvolvimento desse projeto advém do fato de que os *software*s/*hardware*s desenvolvidos na *FPGA* são *royalties-free*. Desse modo, num

desenvolvimento para produção, tem-se uma economia de projeto importante e bastante significativa.

Outro importante fator motivador para esse trabalho foi a ideia de abrir caminho para a pesquisa de formas otimizadas para o cálculo de cinemáticas inversas e diretas, que são gargalos na execução da movimentação cartesiana de manipuladores robóticos.

1.5 ESTRUTURA DESTE TRABALHO

Nesse trabalho, inicialmente, são destacados aspectos teóricos referentes às principais tecnologias envolvidas: Robôs e *FPGA*. Além disso, são comentadas algumas aplicações semelhantes, que podem ser vistas no capítulo 2.

O terceiro capítulo trata o modelo cinemático do manipulador. São descritas as metodologias e as equações encontradas para o modelo cinemático direto e inverso do manipulador. No mesmo capítulo apresenta-se o modelo cinemático relacionado ao controle de velocidade cartesiana, por meio da obtenção do Jacobiano do manipulador.

As plataformas utilizadas para a concepção desse projeto são abordadas no capítulo 4. São comentados os aspectos referentes à *FPGA* utilizada, bem como dos *softwares* necessários à configuração e funcionamento do sistema. Também são explicitadas as interfaces de comunicação e de acoplamento entre os diversos componentes do projeto.

A arquitetura do sistema, tanto de *hardware* quanto de *software* é apresentada no capítulo 5. Nele, são mostrados os algoritmos e as descrições do sistema. São comentados os detalhes dos componentes do processador embarcado e seus periféricos. Além disso, descrevem-se de forma detalhada os processos instanciados no processador e suas funções.

Já o capitulo 6 apresenta os resultados obtidos ao longo do trabalho. Trata-se da validação dos modelos e algoritmos implementados. Além disso, são feitas considerações em relação aos testes realizados com o sistema implementado no manipulador.

O capitulo 7 apresenta as considerações finais, bem como algumas sugestões de projetos futuros. Após este capítulo, são apresentadas as fontes de referências bibliográficas, assim como os anexos.

Os anexos A, B e C apresentam, respectivamente, o código fonte implementado no *NIOS*, os *datasheets* dos principais CIs externos à placa da *FPGA* utilizados no trabalho e o projeto completo de *hardware* implementado na *FPGA*.

2 REVISÃO BIBLIOGRÁFICA

Esse capítulo traz os aspectos teóricos referentes aos conceitos básicos usados neste trabalho, em termos de fundamentos e tecnologias utilizadas no seu desenvolvimento. Inicialmente, apresenta-se um estudo sobre o estado da arte da robótica. Além disso, são apontados alguns aspectos referentes à automação como um processo contínuo de melhoria. Em seguida, é abordada a tecnologia de *hardware* utilizada nesse trabalho, baseados em sistemas reconfiguráveis (usando *Field Programmable Gates Arrays – FPGAs*). Adicionalmente, são apontadas, questões e evoluções a respeito das *FPGAs* e suas atuais aplicações. Ao final, são analisados alguns trabalhos recentes que relacionam as *FPGAs* e a robótica.

2.1 ROBÔS

Com respeito ao que representa o termo "robô", existem várias definições, propostas pelos mais diversos centros de pesquisa e empresas. Essas definições podem variar bastante entre um instituto e outro, ou dependendo dos autores. Contudo, algumas são vistas com mais frequência na literatura:

a) *JIRA* (*Japanese Industrial Robot Association*): Qualquer artefato que substitua o trabalho humano é considerado um robô (Soska, 1985);
b) *RIA (Robot Institute of America)*: Um robô industrial é um manipulador reprogramável, multifuncional, projetado para mover materiais, peças, ferramentas ou dispositivos especiais em movimentos variáveis programados para a realização de diversas tarefas (Schlussel, 1985).

A definição da *RIA* refere-se, exclusivamente a robôs industriais, sendo também muito mais restritiva que a japonesa. Tal definição pode passar a impressão que a robótica é um braço da tecnologia de manufatura (McKerrow, 1993). Contudo, seu elemento chave, trata da questão da capacidade de reprogramação, indicando adaptabilidade e utilidade do robô (Spong, 2006).

A *JIRA* classifica ainda os robôs em 6 categorias:

1) *Robôs Manipulados*: dispositivos com vários graus de liberdade operados manualmente;

2) *Robôs de Sequência Fixa*: dispositivos manipuladores que desempenham sucessivas tarefas de acordo com um método predeterminado, muito difícil de ser modificado;
3) *Robôs de Sequência Variável*: dispositivos que desempenham tarefas sucessivas que podem ser facilmente modificadas;
4) *Robôs Repetitivos*: dispositivos que repetem uma sequência de tarefas gravadas e são conduzidos ou controlados por operador humano;
5) *Robôs de Controle Numérico*: o operador desenvolve o programa de controle do manipulador e o insere num controlador, responsável por compilar o projeto e controlar a sua movimentação;
6) *Robôs Inteligentes*: robôs que apresentam a capacidade de compreender seu ambiente e a habilidade de terminar tarefas apesar das mudanças nas circunstâncias.

Dentro da perspectiva apontada pela *RIA*, apenas os modelos 4, 5 e 6 da classificação japonesa seriam considerados robôs.

Outra classificação bastante aceita é dada pela ISO (*International Organization for Standarization*), na ISO 8373 (Asimo, 2009), a respeito de manipuladores robóticos: um sistema autônomo, reprogramável, de múltiplos propósitos e possuidor de três ou mais eixos de liberdade, que possa ser ou fixo ou móvel, para aplicações em automação industrial. Tal definição é utilizada pela *International Federation of Robotics* (IFR, 2009) e pela EURON (*European Robotics Research Network*)(Euron, 2009).

2.1.1 Robótica

A robótica é um campo relativamente novo da tecnologia moderna, que atravessa os limites da engenharia tradicional. Entender a complexidade e as aplicações dos robôs exige conhecimento de aspectos da engenharia elétrica, engenharia mecânica, ciência da computação, assim como economia e matemática (Spong, 2006). Nesses estudos, diversos aspectos devem ser abordados, como as aplicações atuais e os novos desenvolvimentos.

Este trabalho refere-se especificamente à área de *robótica industrial*, uma vez que se busca desenvolver o controle de um manipulador robótico, capaz de realizar um processo de soldagem. Desse modo, dar-se-á aqui uma explicação mais detalhada sobre os componentes desse tipo de robô. A Figura 2.1 apresenta alguns desses componentes.

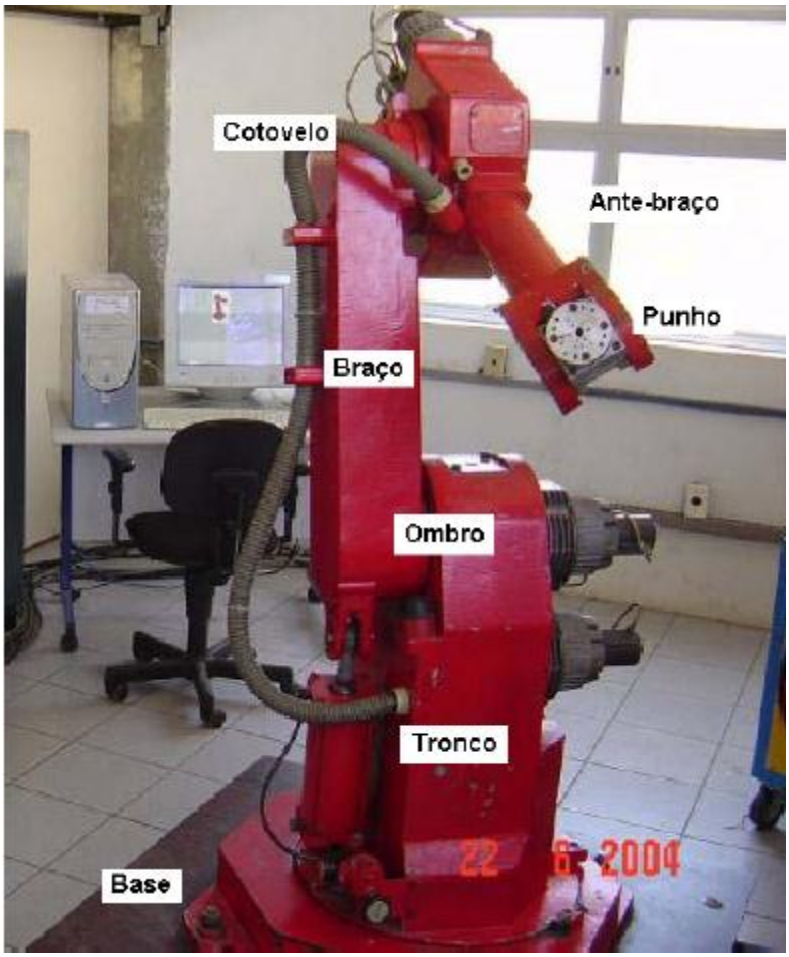


Figura 2.1: Robô manipulador típico **(LEAL, 2005)**

Esses componentes são listados abaixo:

a) *Estrutura Mecânica*: composta por elos, base, punho, juntas, etc;
b) *Atuadores*: motores, cilindros pneumáticos, etc, que fornecem a força e torques necessários à movimentação das juntas do manipulador;
c) *Controlador*: elemento que provê controle às movimentações do manipulador e é interface com o usuário;
d) *Ferramenta*: elemento terminal do manipulador, que permite a manipulação de objetos ou realização de tarefas específicas, como a soldagem.

Os robôs industriais podem ser classificados de diversas maneiras, tais como a forma de atuação, tipo do controlador e muitas outras. Uma dessas maneiras refere-se aos parâmetros geométricos de classificação de um manipulador. Tal classificação permite

uma análise de diversos modelos de manipuladores modernos, incluindo o modelo tratado nesta dissertação.

Os parâmetros geométricos dos manipuladores referem-se ao tipo de cadeia cinemática apresentada por cada um (Tartari Filho, 2006):

- *Seriados*: Possuem uma única cadeia cinemática aberta, onde uma das pontas é fixa e a outra carrega a ferramenta/elemento terminal (Figura 2.2 - A);
- *Paralelos*: Robôs com várias cadeias cinemáticas fechadas, sendo que seus ligamentos têm uma de suas extremidades associada a uma plataforma fixa, enquanto a outra é associada a uma plataforma móvel, onde se encontra o elemento terminal (Figura 2.2 - B);
- *Híbridos*: possuem algumas juntas baseadas em arquitetura serial e outras baseadas em paralelo (Figura 2.2 - C);
- *Arborescentes*: Robôs de múltiplas cadeias cinemáticas que compartilham parte de sua estrutura. Em geral, possuem mais de um elemento terminal (Figura 2.2 - D);
- *Hiper-Redundantes*: Robôs seriados de muitos graus de liberdade. Esses manipuladores possuem mais graus de liberdade do que serão utilizados nos movimentos (Figura 2.2 - E).

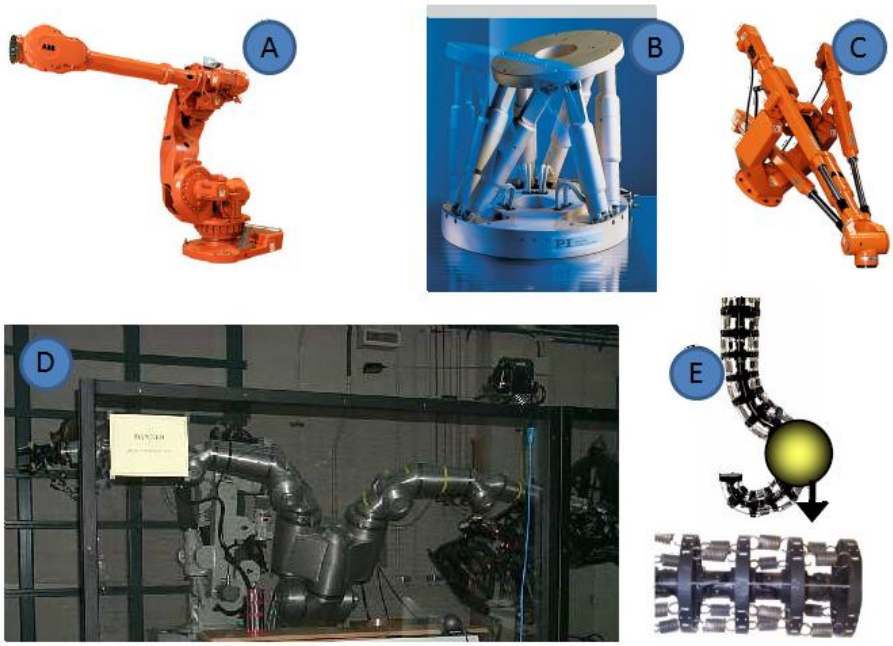


Figura 2.2: Tipos de robôs, segundo sua classificação cinemática **(Tartari Filho, 2006)**.

Outra classificação típica de manipuladores refere-se a sua estrutura mecânica. Algumas das formas mais utilizadas são (Asimo, 2009):

- *Robô Cartesiano*;
- *Robô Cilíndrico*;
- *Robô Esférico*;
- *Robô SCARA* (*Selective Compliance Assembly Robot Arm*);
- *Robô Articulado*;
- *Robô Paralelo*;

Tais classificações podem ser vistas com maior detalhe, na Figura 2.3.

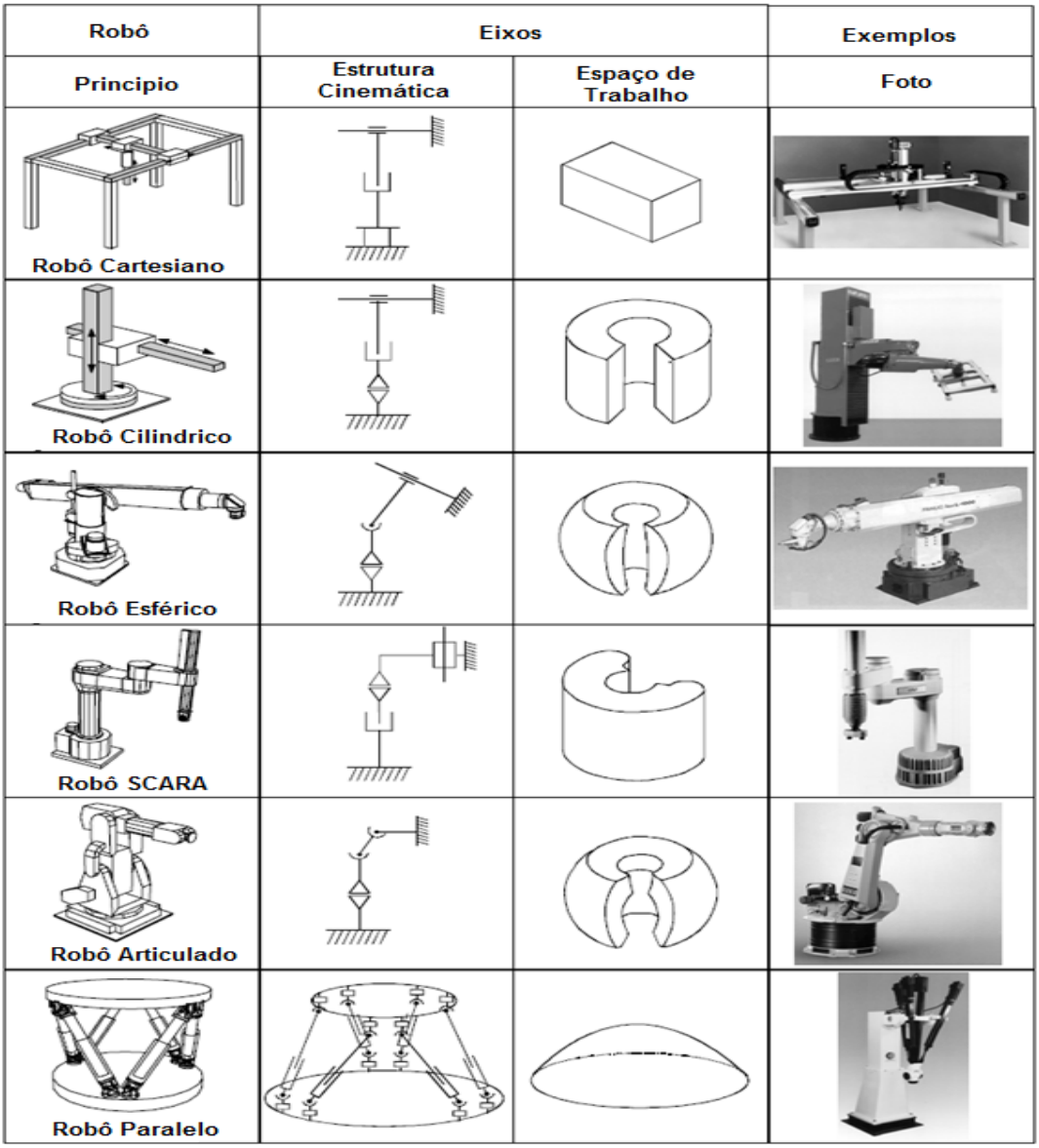


Figura 2.3: Classificação de robôs seriados quanto à estrutura mecânica **(Asimo, 2009)**.

Na Figura 2.3, pode ser vista também a estrutura cinemática de cada um, bem como o espaço de trabalho de cada um deles.

O manipulador descrito nesse trabalho se encaixa no perfil de um robô esférico.

Existe ainda a classificação dos robôs levando-se em consideração o seu nível de inteligência (LEAL, 2005):

- *Robôs de sequencias fixas:* realizam atividades pré-programadas e não alteráveis;
- *Robôs de sequencias variáveis*: realizam sequencias pré-programadas que podem ser alteradas facilmente;
- *Robôs Executores (playback)*: realizam uma tarefa fixa, guiados por um operador humano;
- *Robôs controlados numericamente*: realizam tarefas programadas por meio de linguagens especificas;
- *Robôs Inteligentes:* realizam suas atividades observando as alterações do ambiente e interagindo com elas.

O manipulador desenvolvido nesse trabalho pode ser encaixado em algumas das classificações citadas, podendo ser visto como um manipulador serial, esférico e inteligente – sendo que essa última classificação considera que o manipulador esteja completo, com módulo de visão e soldagem.

2.2 *HARDWARE* RECONFIGURÁVEL

As *FPGAs* representam o segmento que mais cresce no mercado de semicondutores. Isso porque projetos complexos podem ser implementados nelas, sem a necessidade de produção de um custoso *hardware*, específico para cada aplicação. Essa é uma das razões pelas quais esse tipo de dispositivo é uma solução apropriada para o desenvolvimento de sistemas embarcados.

O estudo de sistemas embarcados é um campo que vem despertando muita atenção no atual momento tecnológico, uma vez que mais e mais produtos vêm sendo desenvolvidos com o objetivo de agregar maior capacidade de processamento, robustez, flexibilidade e baixo consumo de energia, em sistemas de pequeno porte.

Atualmente, a área de sistemas embarcados é dominada pelo uso de *FPGAs* (Hartenstein R. , 2006). Segundo a lei de *Rammig*, aproximadamente 90% de todos os *softwares* desenvolvidos para aplicações embarcadas são desenvolvidas em *FPGAs* (Rammig) (vide Figura 2.4), onde problemas partição de *hardware*, *configware* e de *software* tem que ser resolvidos.

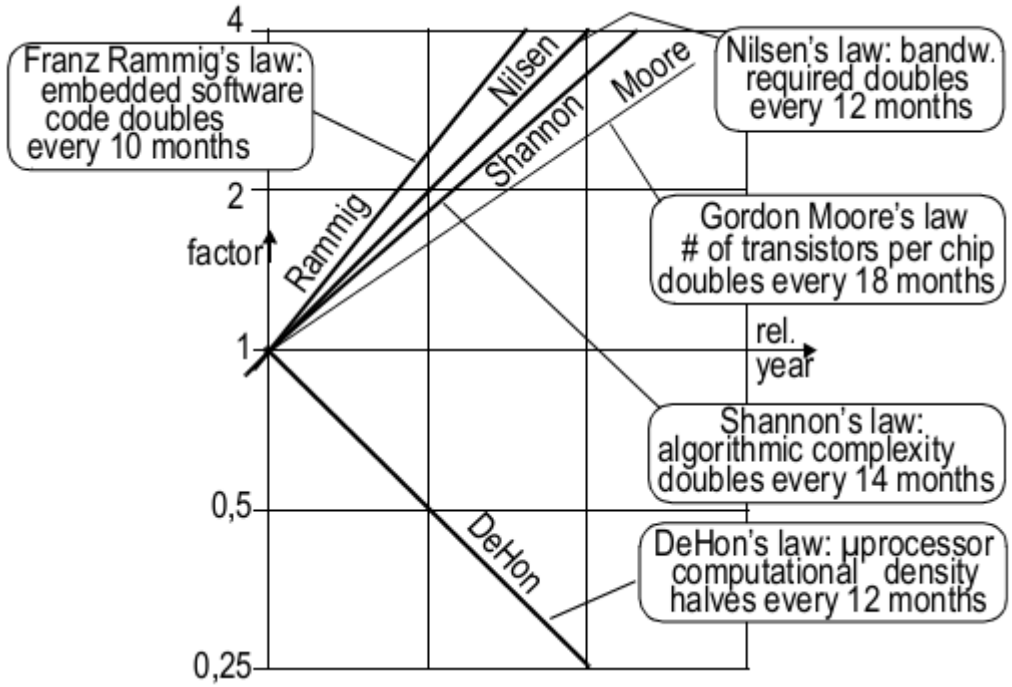


Figura 2.4: Lei de Rammig - Tendência de crescimento **(Hartenstein R. , 2006)**

O número crescente de projetos envolvendo as *FPGAs* retrata o fato de que podem ser utilizadas nas mais diversas aplicações da computação científica, onde alto desempenho é requerido. Alguns exemplos são aplicações médicas, físicas, químicas, matemáticas, dinâmica dos fluidos, astrofísica, biologia e outras.

Esses sistemas costumam apresentar requisitos de *hardware* e *software* diferenciados em relação aos sistemas comuns. Em geral, apresentando restrições de espaço, peso, consumo de energia, etc., devendo ter funções otimizadas de modo a garantir seu funcionamento de forma confiável, sob diversas condições e situações.

2.2.1 Microcontroladores

Os microcontroladores encontrados no mercado possuem uma arquitetura convencional tipo *von Neumann* (MUÑOZ, 2006). Essa arquitetura apresenta uma unidade lógica

aritmética (ULA) executando uma única instrução por vez, dentro de um conjunto de instruções possíveis. Ela armazena e lê dados em uma RAM, na qual um contador de programa controla o fluxo de instruções, como é mostrado na Figura 2.5. Este tipo de arquitetura foi formulada por John *von Neumann* nos anos 1960 e constituiu, durante anos, o dispositivo computacional mais flexível sobre o qual diversas aplicações tem sido realizadas (Harteinsten, 2000).

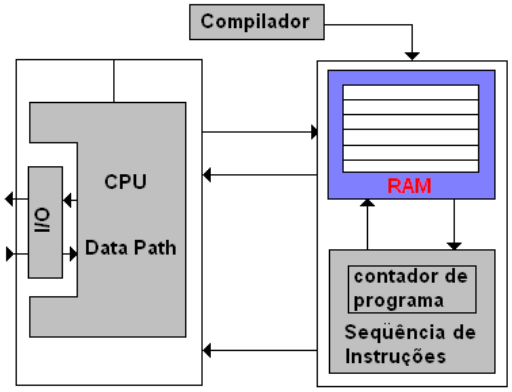


Figura 2.5: Arquitetura *von Neumann* **(MUÑOZ, 2006)**

A evolução de novas tecnologias nos processos de fabricação dos circuitos integrados permitiu que a complexidade e o desempenho aumentassem. No mercado, são encontrados microcontroladores de alto desempenho que permitem uma velocidade maior no processamento de dados, no cálculo de operações e na execução de instruções.

Embora, a evolução da indústria de memórias de acesso aleatório (RAM) tenha crescido consideravelmente em desempenho e capacidade de armazenamento, a velocidade de funcionamento das memórias, isto é, a velocidade com que são escritos ou lidos os dados na memória, não consegue acompanhar a velocidade com que são processados os dados na unidade central. Isso é conhecido como barreira de memória (*memory wall*) (Hartenstein R. , 2006).

Diversas alternativas podem ser estudadas e aplicadas para a solução dos problemas de gargalo de processamento, apresentados pela arquitetura de *von Neumann*. Uma dessas é a utilização de arquiteturas reconfiguráveis, utilizando *FPGAs* e desenvolvendo estratégias para mapear os algoritmos diretamente em *hardware*, gerando estruturas de fluxo de dados (tipo arranjo sistólico, por exemplo), *pipeline*s customizados e/ou estruturas que eliminem dependências de dados e/ou de controle (típicos do modelo de *von Neumann*).

2.2.2 *FPGA*S (*Field Programmable Gate Array*)

As *FPGA*s podem ser vistas como um conjunto de elementos lógicos reconfiguráveis embarcados em um *chip* de interconexões reconfiguráveis (Hartenstein R. , 2006). Seu *configware* – como é chamado o *software* de configuração do *hardware* da *FPGA* – é armazenado em uma memória hRAM – do Inglês *hidden* RAM -, alocada ao circuito da *FPGA*. Esse *configware* deve ser carregado para a *FPGA* cada vez que ela é alimentada, configurando os elementos lógicos da *FPGA* (Hartenstein R. , 2006).

Os elementos lógicos de uma *FPGA* contêm algumas entradas e uma saída e, dentro deles, são encontradas unidades formadas por um ou dois *flip-flops*, onde é possível armazenar valores de "0" ou "1". Os tipos de blocos lógicos mais comumente encontrados são baseados em LUT (*LookUp Table*).

Por meio do controle de um grupo de multiplexadores e portas, essas unidades, permitem o fluxo de dados desde as células de armazenamento até a saída do bloco lógico, implementando a função lógica desejada. O arranjo de células dentro da LUT é utilizado para armazenar a tabela verdade da função lógica.

Dentro de uma mesma *FPGA*, o circuito utilizado pela aplicação pode ser totalmente reconfigurado, permitindo assim que diferentes arquiteturas e funcionalidades possam ser implementadas e desenvolvidas, sem modificação do *hardware*. Desse modo, podem-se definir dois tipos básicos de *FPGAs* (Silva, 2010): as de granularidade fina (*fine-grain*), que executam operações de apenas um bit de largura, definindo circuitos no nível de portas lógicas, o que aumenta a flexibilidade de projeto, mas que torna o trabalho de reconfiguração do dispositivo bastante demorado. Já os dispositivos de granularidade grossa (*coarse-grain*), operam em nível de transferência entre registradores, tendo sido muito utilizados para implementação de aceleradores de *hardware* para diversas aplicações.

Como dito anteriormente, após alimentar a *FPGA*, o código de configuração é carregado na hRAM da *FPGA*. Contudo, esse *software* não é composto de sequencia de instruções, como no paradigma de Von Neuman (Hartenstein R. , 2006). As *FPGA*s trazem um novo paradigma para sistemas baseados em memória RAM: o *Kress-Kung Machine* (The Kress-Kung Machine Page, 2001). Nesse novo paradigma, ao invés de organizar uma lista de

instruções a executar, a compilação organiza os recursos de *hardware*, por meio de algoritmos de posicionamento (*planning*) e roteamento. Baseado no resultado da compilação, são definidas as listas de dados a serem distribuídos ao longo do *hardware* implementado. A esse *software* de configuração da *FPGA* é atribuído o nome de *configware*, visando não provocar confusões com o termo *software* (Hartenstein R. , 2006). A Figura **2.6** mostra uma comparação entre engenharia de *software* e engenharia de c*onfigware*.

Nesse novo contexto, existem três tipos de possíveis arquivos de configuração para um sistema. Na arquitetura tradicional (von Neumman), apenas o algoritmo é variável, uma vez que a plataforma utilizada é fixa. Desse modo, apenas um tipo de arquivo de configuração é necessário: o *software*. A partir desse arquivo, o compilador gera um código de máquina a ser carregado na memória de instruções (*RAM Instructions*).

Para a *FPGA*, no entanto, não somente o algoritmo, mas também os recursos são programáveis. Daí a necessidade de dois arquivos de configuração:

a) *Configware*: determina a estrutura do *hardware*, por meio do mapeamento, usando posicionamento e roteamento, ou métodos semelhantes de mapeamento (Configware, 2005);
b) *Flowware*: determina o fluxo de dados a ser passado como parâmetro ao *hardware* instanciado na *FPGA* (The Flowware Page, 2001).

Esses dois sistemas diferentes contrapõem as arquiteturas de *von Neumann* e a *Kress-Kung machine (FPGA)*. A grande vantagem da arquitetura *FPGA* é que os blocos lógicos podem ser configurados para permitir o fluxo de dados de forma independente e paralela, além de permitir a reconfiguração dos mesmos em tempo de execução. Isto possibilita a execução, em uma única pastilha, de diferentes processos concorrentes, ampliando o espectro de aplicações das *FPGA*s. Atualmente são utilizadas ferramentas de CAD para reconfigurar os elementos lógicos que constituem a *FPGA*, as quais envolvem aspectos clássicos de síntese de alto nível, síntese lógica e ferramentas de mapeamento tecnológico, posicionamento e roteamento global e local.

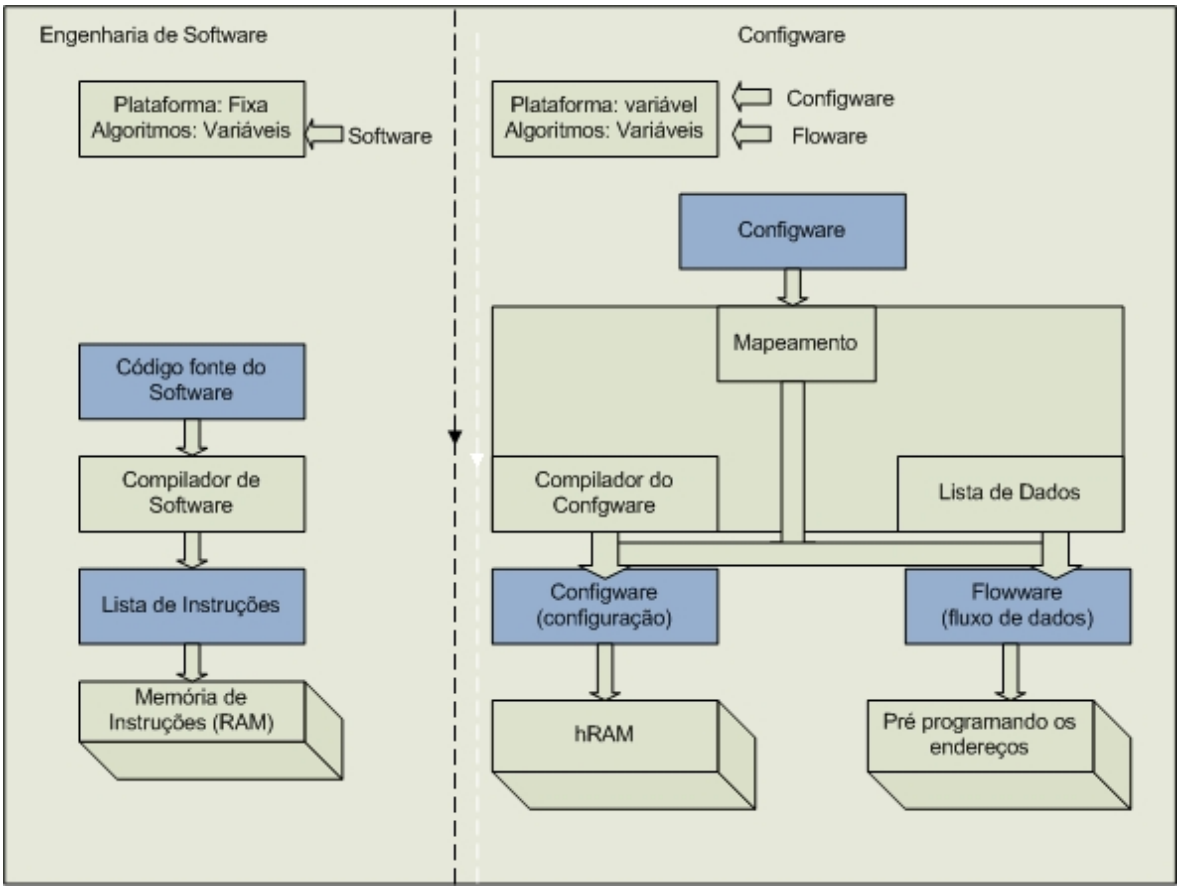


Figura 2.6: Engenharia de *Software* x *Configware* (Hartenstein R. , 2006, modificado)

Hoje em dia, as tendências tecnológicas preparam uma nova mudança, porém, alguns estudos afirmam que não haverá uma próxima transição e sim um estabelecimento lento e prolongado que considera tecnologias avançadas na área de sistemas embarcados (*Embedded Systems*) e computação reconfigurável (*Configware*), deixando de lado o tradicional *hardware* fixo (Harteinsten, 2000).

A Figura 2.7 mostra claramente a diferença do desempenho dos processadores em relação às memórias durante os últimos 20 anos. Esta diferença em desempenho, forma, hoje em dia, um dos maiores inconvenientes das arquiteturas convencionais, sendo conhecido como "gargalo de *von Neumann*" (Harteinsten, 2000). Em aplicações onde são necessários cálculos complexos, é comum encontrar um conjunto de processadores atuando de forma paralela com o objetivo de aumentar a eficiência e diminuir o tempo de cálculo.

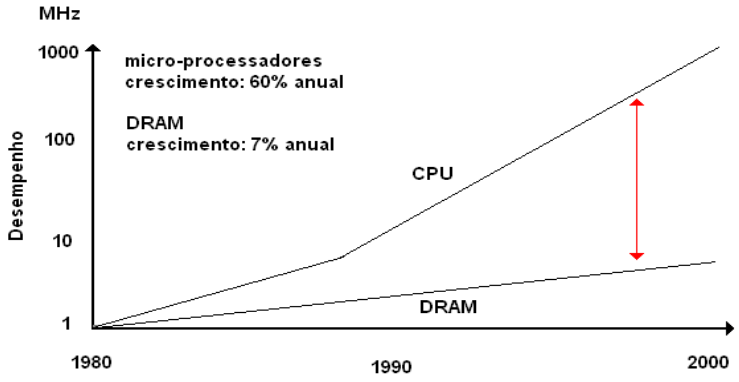


Figura 2.7: Paradigma de *von Neumann* (modificado – **(Harteinsten, 2000)**)

O mercado de fabricação de dispositivos *FPGA* é dominado pelas empresas Altera Corporation (Altera, 2009) e Xilinx Corporation (Xilinx), apesar de existirem outras empresas dividindo esse mercado. Desse modo, existe uma grande variedade de opções de dispositivos presentes no mercado, sendo necessário o conhecimento das principais características de qual tecnologia aplicar em determinado projeto. As principais características a serem analisadas são:

a) Tecnologia de programação utilizada;
b) Arquitetura interna dos blocos lógicos;
c) Arquitetura de roteamento.

No contexto do presente trabalho, as implementações foram realizadas na *FPGA* Stratix III, uma das famílias comercializadas pela *Altera*, pois as suas características de desempenho e capacidade são adequados para o tipo de aplicação desejada. As placas baseadas na Stratix (Altera, 2009) possuem as interfaces necessárias para a implementação da arquitetura de controle do manipulador proposto.

As empresas fabricantes de *FPGAs* possuem dispositivos para diferentes tipos de aplicações. Empresas como a *Altera* e *Xilinx* fornecem blocos a serem utilizados nos projetos, chamados de IP *cores*, e ferramentas de importante ajuda na elaboração de projetos. Por outro lado, os desenvolvimentos de aplicações mostram um aumento constante de publicações científicas (Hartenstein R. , 2006), indicando outras formas de programação utilizando *softwares* não proprietários. Os dispositivos *FPGA* fornecidos pelas empresas Altera e Xilinx são classificados em famílias, conforme as suas principais características, como é mostrado nas tabelas 2.1 e 2.2.

Tabela 2.1:: Resumo das famílias e características das *FPGA* da Altera

Família	Nome	LE	User I/O	Memory Bits	PLL	DSP
Cyclone II	EP2C35	33216	475	483840	4	-
Cyclone III	EP3C120	119088	532	3981312	4	-
Stratix	EP1S25	25660	707	1944576	6	10
Stratix II	EP2S60	48352	493	2544192	6	36
Stratix III	EP3SL150	113600	488	5630976	4	48
Stratix IV	EP4SGX230	171000	644	14625792	3	161

Tabela 2.2: Resumo das famílias e características das *FPGA* da Xilinx

Família	FPGA	Células Lógicas	BRAM (Kbits)	Processadores PowerPC	Multiplicadores 18x18 Bits
Virtex-II Pro	XC2VP2	3168	216	0	12
Virtex-II Pro	XC2VP30	2448	2448	2	136
Virtex-II Pro	XC2VP125	125136	10008	4	556
Spartan - 3	XC3S50	1728	72	-	4
Spartan - 3	XC3S200	4320	216	-	12
Spartan - 3	XC3S5000	74880	1872	-	104

Contudo, não são apenas os produtores/vendedores de *FPGA*s que produzem as ferramentas para confecção de *configwares*. Outras fontes comerciais também produzem suas ferramentas de configuração para esses equipamentos. Essas ferramentas são chamadas *EDA* (*Eletronic Design Automation*) e são necessárias, em virtude da grande complexidade do processo de criação para esses dispositivos.

2.3 ESTADO DA ARTE

Existem diversos trabalhos que tratam da modelagem e controle de manipuladores robóticos. Esses apresentam as mais diversas formas de controles aplicados à robótica.Por exemplo, existem controladores desenvolvidos utilizando apenas blocos de DSP, ou associando *DSP* e *ASIC*, ou mesmo utilizando uma *FPGA* para controles de mais alto nível, deixando as funções de cálculo mais pesadas para um ou mais blocos DSP

integrados, visando otimização (Sun & Mills, 2006). Outros controladores são ainda desenvolvidos utilizando microcontroladores ou PC's dedicados.

Sun & Mills (Sun & Mills, 2006) propõem uma arquitetura de controle, via *hardware*, para aumentar a performance de movimentação de um manipulador durante seus deslocamentos em alta velocidade. Para tanto, são passadas para a *FPGA* algumas funções anteriormente designadas para o DSP, como o fechamento da malha de controle dos servos-motores utilizados pelo manipulador. No mesmo trabalho, os autores propõem a divisão do controle do manipulador em uma parte linear, controle em malha fechada de velocidade e posição, e uma não linear, compensação dinâmica da parte linear, visando calcular movimentações mais complexas. Nessa arquitetura, a parte não linear é mantida no DSP, enquanto que a parte linear fica toda sob controle da *FPGA*. A arquitetura proposta, incluindo *FPGA* e DSP é mostrada na Figura 2.8.

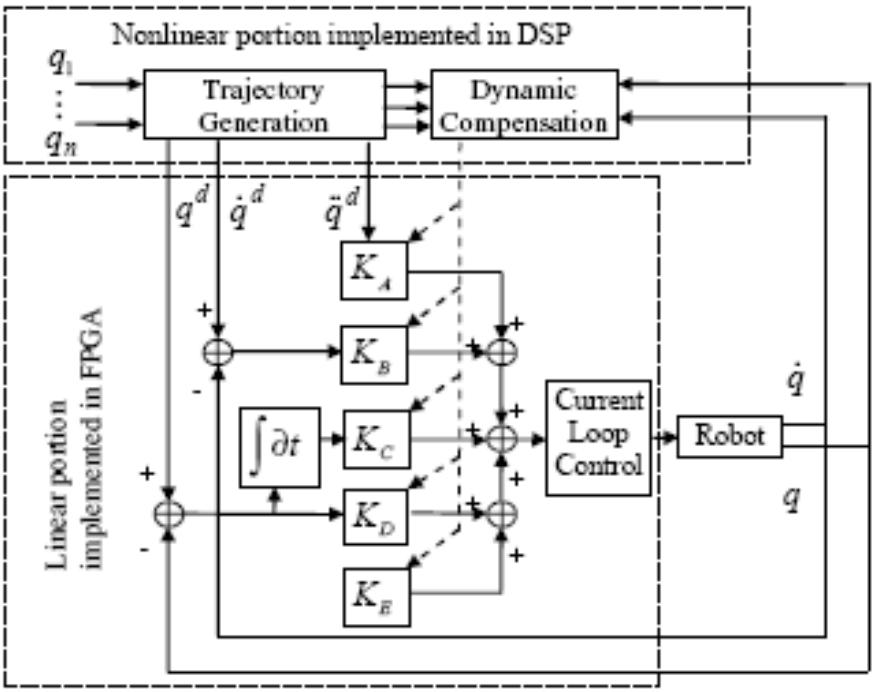


Figura 2.8: Arquitetura proposta por **(Sun & Mills, 2006)**.

Outra arquitetura é proposta por Yili, Hanxu, Qingxuan, & Guozhen, (2008). Nesse trabalho, os autores apresentam a elaboração do controle cinemático para um manipulador de 6 graus de liberdade, baseado num microcontrolador ARM e uma *FPGA* como co-processador. O ARM é responsável por calcular a trajetória a ser desempenhada pelo elemento terminal do manipulador. Já os cálculos de cinemática direta e inversa são desenvolvidos na *FPGA*, utilizando o CORDIC (*Coordinate Rotation Digital Computer*).

Segundo os autores, essa implementação permite reduzir o tempo de cálculo do controle da cinemática e otimizar o desempenho do manipulador em tempo real.

Kung e Shu (2005), trazem uma arquitetura bastante semelhante à desenvolvida neste trabalho. Eles apresentam o desenvolvimento do controlador de um manipulador robótico de 5 graus de liberdade baseado em uma plataforma *FPGA*. Nesse trabalho, os autores apresentam o desenvolvimento de uma arquitetura que utiliza o microcontrolador *NIOS* para realizar os cálculos de cinemática inversa do manipulador e para controlar a movimentação ponto a ponto do robô. Associado ao *NIOS*, foi desenvolvido um segundo módulo, baseado em *hardware*, para o cálculo da velocidade e posicionamento do manipulador. Ambos foram unidos utilizando a ferramenta SOPC Builder, da Altera. Para o controle de posição, foi embarcado também um controlador proporcional, utilizando *encoders* para fechar a malha de controle do manipulador. A arquitetura proposta por Kung e Shu pode ser vista na Figura 2.9.

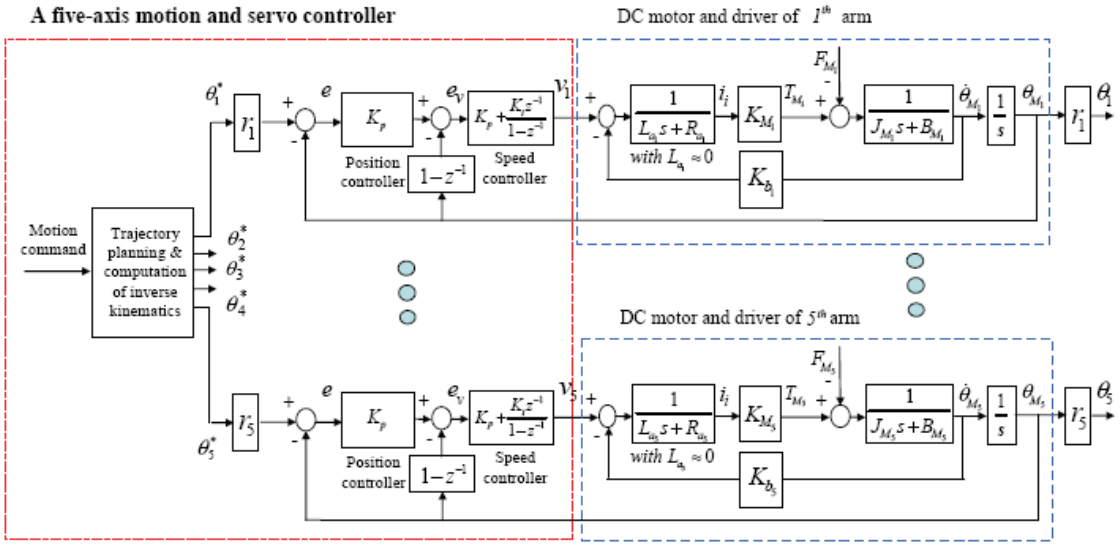


Figura 2.9: Diagrama de blocos mostrando a arquitetura proposta por (Kung & Shu, 2005)

O mesmo trabalho (Kung & Shu, 2005) traz ainda como destaque a não necessidade da utilização de DSPs para o controle dos motores. Desse modo, utiliza-se somente um *chip*, embarcado na *FPGA* para realizar todas as operações de controle do manipulador.

Outros trabalhos trazem algumas aplicações de arquiteturas utilizando *FPGAs* para o controle de outros tipos de robôs. É o caso, por exemplo, do trabalho apresentado por Jun,

Seok, Min, & Kyu (2006). Neste, é apresentada uma arquitetura para o controle de dois braços robóticos, com duas mãos, como elementos atuadores. Os autores destacam a capacidade da *FPGA* de comandar os motores, sem problemas de sincronização, uma vez que essas possuem uma série de portas I/Os que podem ser utilizados para esse fim. O mesmo trabalho apresenta, ainda, o *NIOS* como responsável pelo gerenciamento de alto nível do processo.

2.4 CONCLUSÕES DO CAPÍTULO

Neste capítulo, foram apresentados aspectos atuais referentes à utilização de robôs, além de algumas classificações a respeito dos mesmos. Foram discutidas algumas classificações de manipuladores, bem como suas definições.

Além disso, também foram discutidos aspectos referentes à tecnologia das *FPGAs* e seus conceitos. Foram abordados também aspectos a respeito das vantagens da utilização do *hardware* reconfigurável em relação ao uso de *hardware* de função fixa.

Foram levantados ainda, alguns projetos que serviram de base para o desenvolvimento desse trabalho.

3 MODELO CINEMÁTICO

Nesse capítulo, será descrito o processo de elaboração do modelo cinemático do manipulador objeto do presente trabalho. Nesse processo, será formulado um sistema de equações que relacionam as variáveis de junta do manipulador com a posição e orientação da ferramenta, em relação ao sistema de coordenadas fixado em sua base.

Apresenta-se também um modelo cinemático de velocidade para o manipulador. Para tanto, são desenvolvidas as equações que compõem o Jacobiano do robô. Tal modelo permite analisar as velocidades das juntas para garantir a velocidade do elemento terminal do robô.

O desenvolvimento desses modelos proporciona a observação das condições de movimentação e posicionamento do manipulador. Desse modo, é possível elaborar as estratégias de movimentação do robô.

3.1 MODELO

O robô a ser desenvolvido apresenta 5 graus de liberdade, tendo sua representação simplificada conforme apresentado na Figura 3.1. Sua topologia pode ser comparada à dos manipuladores Stanford (Spong, 2006) e ao Puma modificado(McKerrow, 1993), que possuem 6 graus de liberdade (ver figuras 3.2 e 3.3). Os cinco graus de liberdade são associados as 5 juntas do manipulador, sendo as três primeiras responsáveis pelo posicionamento do robô, enquanto as duas últimas tratam da sua orientação.

Pode-se observar que a Figura 3.1, que representa o manipulador, não apresenta a ferramenta a ser utilizada no processo de soldagem, isto é, as origens dos sistemas de coordenadas fixados nos elos 4 e 5 do manipulador – referentes ao "punho" do robô – foram consideradas coincidentes. Essa foi uma opção adotada na modelagem com o objetivo de se simplificar o modelo.

Tal simplificação baseia-se no fato de que a localização (posição e a orientação) do sistema de coordenadas fixado à tocha de soldagem[1] pode ser representada por uma transformação

[1] Localizado ao longo da direção longitudinal do arame, a uma distância da extremidade do tubo de contato correspondente ao chamado "stand-off" requerido pelo processo GMAW, com o eixo x alinhado com a direção e sentido de saída do arame.

homogênea fixa a partir do sistema de coordenadas fixado no elo 5 do manipulador. As justificativas matemáticas para essa escolha serão explicadas posteriormente no texto.

Tal escolha permite uma flexibilidade maior do sistema, uma vez que, determinado o modelo do manipulador, pode-se alterar a tocha, bastando, para isso, determinar os parâmetros da transformação homogênea correspondente à nova tocha e inseri-la na equação que rege o sistema.

A Figura 3.1 apresenta ainda os sistemas de coordenadas fixados a cada elo do robô. Esses elos foram baseados na convenção de Denavit-Hatenberg (McKerrow, 1993), a ser explicada posteriormente. A determinação desses elos possibilita a modelagem do manipulador.

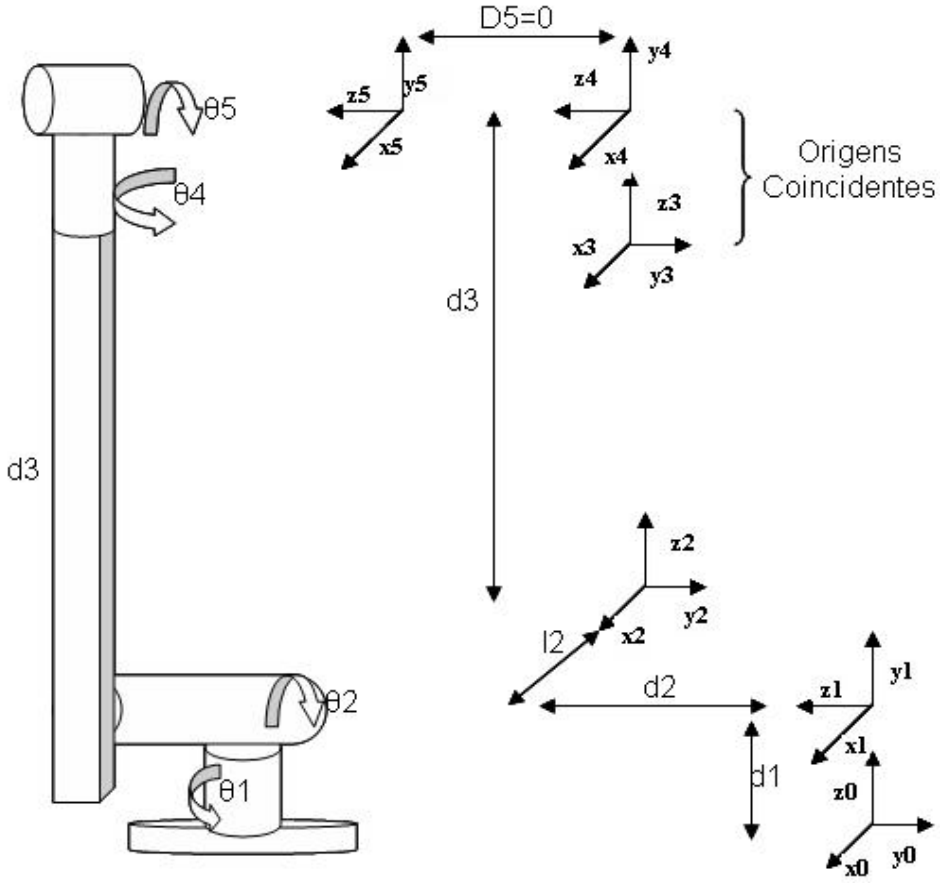


Figura 3.1: Visão do Modelo do Manipulador

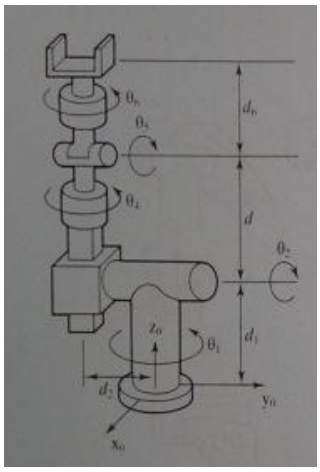


Figura 3.2: Manipulador Puma Modificado **(McKerrow, 1993)**

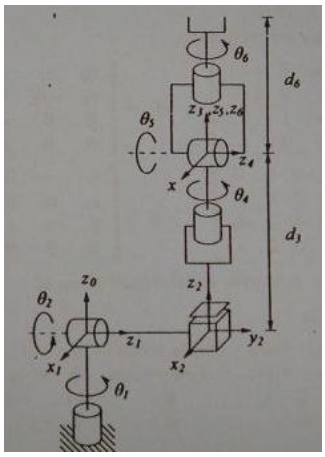


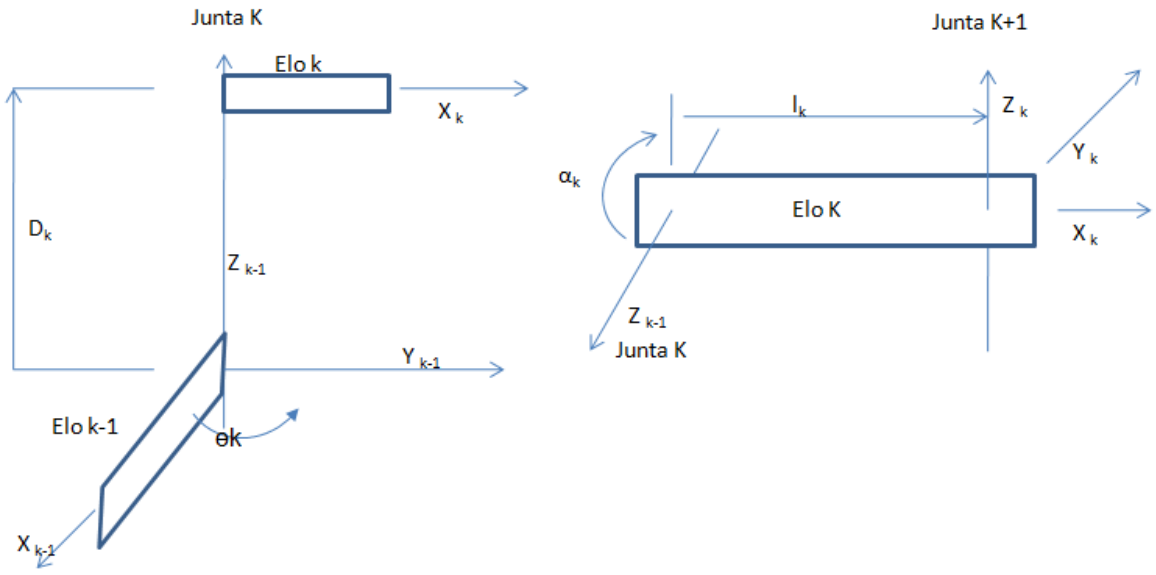
Figura 3.3: Manipulador Stanford **(Spong, 2006)**

No modelo adotado, considera-se que os eixos *z3* e *z4* (e também o eixo *z5*) interceptam-se em um ponto *Oc*. Para que essa consideração possa ser satisfeita faz-se necessário introduzir um deslocamento na direção do eixo x do sistema de coordenadas atribuído ao elo 1. Essa variável é apontada na Figura 3.1 como a distância *l2*.

A definição de *l2* como parâmetro variável do manipulador permite posicionar o sistema de coordenadas do seu eixo prismático de modo a satisfazer a condição de interceptação do ponto *Oc*, como desejado. Novamente, tal observação pode ser comprovada na Figura 3.1, onde *l2* desloca o eixo de coordenadas *(x2,y2,z2)*.

3.2 PARÂMETROS CINEMÁTICOS

Definem-se como parâmetros cinemáticos o conjunto de posições e orientações relativas entre um par de elos adjacentes conectados por uma junta prismática ou rotacional (Spong, 2006). Na Figura 3.4 (a), os elos *k* e *k*-1 são unidos pela junta *k*. Dessa forma, a distância entre os elos, ao longo do eixo z_{k-1} é dada por d_k. Já θ_k mede a variação angular do eixo *k* em torno de Z_{k-1}. θ_k também é a rotação necessária em torno de z_{k-1}, para que os eixos x_{k-1} e x_k se tornem paralelos. Na Figura 3.4 (b), existe um elo *k* entre as juntas *k* e *k*+1, assim l_k é a distância entre os dois sistemas de orientação. Ou seja, l_k é a translação ao longo de x_k, para fazer com que z_{k+1} e z_k se interceptem. O α_k é definido como a variação angular entre os eixos de junta *k*+1 e *k*, ou seja, é a rotação necessária em torno de x_k para que os eixos z_{k+1} e z_k fiquem paralelos.



(a) (b)

Figura 3.4: Parâmetros Cinemáticos

Outras definições são necessárias para facilitar o entendimento do desenvolvimento das equações seguintes. É o caso, por exemplo, da definição de matriz de transformação. Para descrever a movimentação de um objeto, é necessário relacionar a localização inicial de um sistema de coordenadas fixado ao objeto com sua localização final, após realizada a sua movimentação (McKerrow, 1993).

Além disso, é necessário descrever a sequencia de rotações e translações que levaram o objeto a partir de sua posição original à nova localização (posição e orientação). Isso pode ser feito com um conjunto de matrizes de transformação aplicadas sequencialmente a um sistema de referência, de modo a fazer com que aquele se movimente em direção a localização do novo sistema de coordenadas (McKerrow, 1993).

Essas matrizes 4x4 são definidas como matrizes de transformação homogênea (Forest, 1969). Em tais matrizes, chamadas de matrizes *T*, as três primeiras colunas (vetores 3x1) representam as direções dos eixos de um sistema de coordenadas projetados em um sistema de coordenadas de referência. A matriz 3x3 resultante no caso da robótica é uma matriz de rotação com determinante unitário. Já os três primeiros elementos (vetor 3x1) da última coluna da matriz 4x4 original representam as translações cartesianas da origem do sistema de coordenadas em relação ao sistema de referência. A última linha da matriz representa, por meio de seus três primeiros números, uma projeção do sistema, enquanto o último número indica uma escala.

A matriz de transformação homogênea geral pode ser vista na eq. 3.1.

$$T = \begin{bmatrix} Rotação\ (3x3) & Translação\ (3x1) \\ Projeção\ (1x3) & Escala\ (1x1) \end{bmatrix} \tag{3.1}$$

A matriz apontada na eq. (3.1) traz a transformação homogênea geral de um sistema de coordenadas genérico em relação a um sistema de referência qualquer. Contudo, faz-se necessário definir, a partir dela, matrizes homogêneas de translação e de rotação.

A equação geral de uma translação ao longo do eixo *x*, de uma distância *k*, é mostrada na eq. (3.2).

$$Trans(x,k) = \begin{bmatrix} 1 & 0 & 0 & k \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

A matriz geral de transformação, correspondente a uma translação pelo vetor $p_x\hat{\imath} + p_y\hat{\jmath} + p_z\hat{k}$ é vista na eq. (3.3) (McKerrow, 1993).

$$Trans(p_x, p_y, p_z) = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Já as matrizes de rotação são mais complexas, uma vez que é possível realizar rotações em torno de quaisquer dos eixos de coordenadas. Rotações em torno de um eixo genérico podem ser representadas utilizando uma correta marcação dos eixos das coordenadas do manipulador e decompondo aquelas em rotações em torno desses (Spong, 2006). Desse modo, existem três transformações de rotação correspondentes a rotações em torno dos eixos x, y e z, por um ângulo φ, conforme mostram as equações (3.4), (3.5) e (3.6).

$$Rot(x, \varphi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$Rot(y, \varphi) = \begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$Rot(z, \varphi) = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

3.2.1 Convenção Denavit-Hartenberg

Para realização da modelagem do robô, inicialmente, determinaram-se os sistemas de coordenadas para cada junta do manipulador. Tais sistemas de coordenadas foram atribuídos seguindo-se a notação proposta pelo algoritmo de D-H (Denavit-Hartenberg) para mecanismos (Spong, 2006). Essa notação é utilizada para descrever, de forma sistemática a posição e a orientação entre dois elos de um manipulador, por meio de coordenadas homogêneas. Desse modo, obtêm-se a posição e a orientação do elemento terminal do manipulador em função dos deslocamentos de todas as juntas do robô (Spong, 2006).

Nessa notação, cada transformação homogênea A_i é representada como um produto de 4 transformações básicas, apontadas nas equações (3.7). e (3.8), em sua forma matricial.

$$A = Rot(z, \theta) * Trans(z, D) * Trans(x, L) * Rot(x, \alpha) \tag{3.7}$$

$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta)\ \cos(\propto) & \sin(\theta)\ \sin(\propto) & L\cos(\theta) \\ \sin(\theta) & \cos(\propto)\ \sin(\propto) & -\cos(\theta)\ \sin(\propto) & L\sin(\theta) \\ 0 & \sin \propto & \cos\theta & D \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.8)$$

A Tabela 3.1 mostra os parâmetros da cinemática do sistema, obtidos a partir da análise da Figura 3.1, já utilizando a convenção de D-H. É importante ressaltar que, no modelo, o ângulo *θ2* é deslocado em 90º (*offset*) em relação à representação da Figura 3.1. Isso ocorre em função da posição inicial desejada para o robô e de suas limitações de movimentação, que visam proteger a ferramenta, fixada em seu órgão terminal. O modelo do manipulador, com o deslocamento de 90º, pode ser visto na Figura 3.5.

Tabela 3.1: Parâmetros D-H

Variável de Junta	θ(º) (Rz.)	α(º) (Rx)	D(mm) (Tz)	L(mm) (Tx)
θ_1	θ_1	90	d_1	0
θ_2	θ_2 + 90	-90	d_2	L_2
d_3	0	0	d_3	0
θ_4	θ_4	90	0	0
θ_5	θ_5	0	0	0

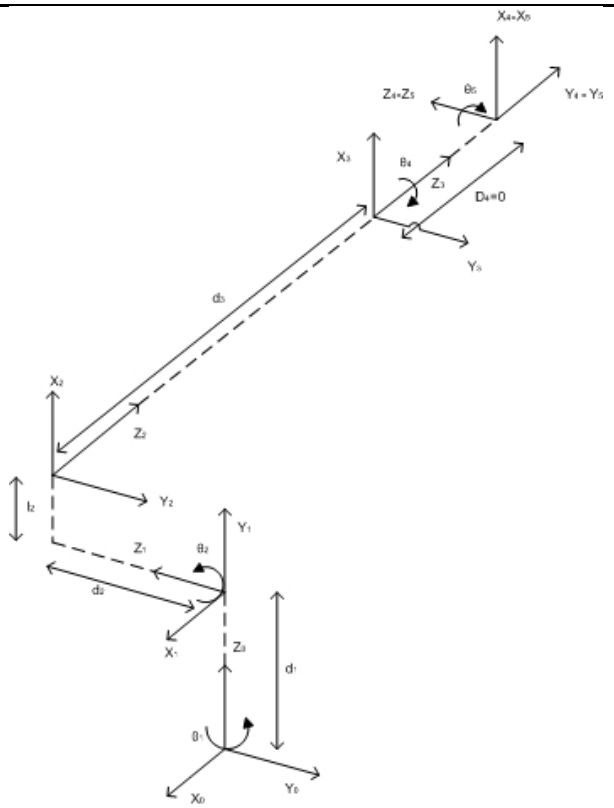


Figura 3.5: Diagrama de posicionamento do manipulador, com θ_2 deslocado em +90 º.

Os elos apontados na Tabela 3.1 podem ser vistos na Figura 3.6.

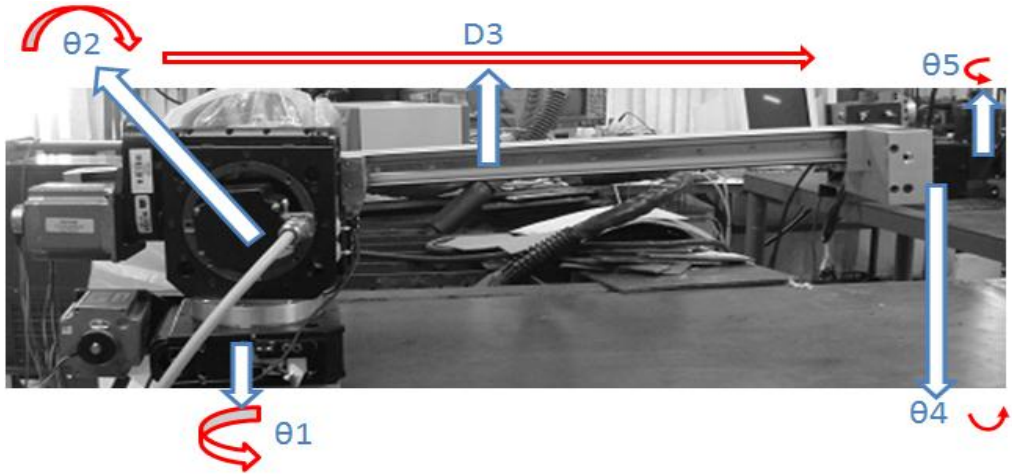


Figura 3.6: Manipulador e seus elos (*Pan Tilt* Antigo)

Importante ressaltar ainda que ao se definir o *offset* em +90°, deve-se observar que as coordenadas *x* da origem do sistema 5 em relação ao sistema 0 (fixado na base do manipulador) apresentam valor negativo para θ_1 situado entre -90º e + 90º e para θ_2 situado entre -45º e *+45º*. O modelo resultante poderá ser ajustado para o outro *offset*, bastando para isso, substituir nas equações literais resultantes da modelagem, o valor de θ_2 pelo novo valor adicionado de um novo *offset* (por exemplo, θ_2*-180º*, para se atingir a posição originalmente atingida se θ_2 fosse subtraído de 90º inicialmente).

Desse modo, a movimentação da ferramenta, no eixo x do manipulador acontecerá por meio de incrementos negativos, se o ponto final ficar à frente do manipulador. Essa foi uma escolha de desenvolvimento do projeto, por iniciativa do pesquisador.

As juntas do manipulador apresentam ainda, restrições de movimentação, determinadas pela geometria do manipulador. Essas restrições referem-se aos valores máximos e mínimos que cada junta pode assumir, durante a movimentação do manipulador, a fim de evitar colisões. A Tabela 3.2 mostra as restrições adotadas neste trabalho, para o manipulador escolhido.

Tabela 3.2: Restrições de movimentação das juntas do manipulador

Junta	Unidade	Posição mínima	Posição máxima
θ_1	Graus	-179	180
θ_2	Graus	-30	30
d_3	mm	150	750
θ_4	Graus	-45	45
θ_5	Graus	-45	45

3.2.2 Matriz de Transformação Homogênea

A partir da Tabela 3.1, foi possível calcular as matrizes de transformação homogênea de cada eixo do robô e, a partir delas, obter a matriz de transformação geral do manipulador (${}_R T^H$). Tais equações são mostradas nas eq. (3.9), (3.10), (3.11), (3.12) e (3.13). Por outro lado, a matriz de transformação homogênea completa entre a base e o elemento terminal do manipulador é mostrada na eq. (3.15). Nessas equações foram utilizadas as notações: $c_\theta = cos(\theta)$ e $s_\theta = sin(\theta)$.

Além disso, pode-se notar que todas as equações foram realizadas de forma literal. Essa foi uma escolha baseada no fato de existirem pequenas diferenças entre o modelo e o robô real. Ao se colocarem esses parâmetros de modo genérico, há maior dificuldade em se tratarem computacionalmente os dados do manipulador e resolver suas equações. Tal dificuldade é advinda da não possibilidade de simplificação de termos numéricos. Entretanto, essa é uma escolha que permite pequenos ajustes posteriores de variações em alguns parâmetros geométricos do manipulador.

A eq. (3.9) indica a matriz de transformação entre o sistema da base do manipulador e o sistema fixado em seu primeiro elo rotacional – junta 1. A eq. (3.10) indica a transformação entre os sistemas de coordenadas fixados no primeiro elo rotacional e no segundo - junta 2. A eq. (3.11) Indica a transformação entre os sistemas fixados no elo 2 e o sistema fixado no eixo prismático do manipulador. Da mesma forma, a eq. (3.12) indica a transformação entre o sistema do elo 3(junta prismática) e o sistema do elo 4 (junta rotacional). Para finalizar, a eq. (3.13) mostra a transformação entre os sistemas fixados nos elos 4 e 5.

A transformação homogênea que representa a localização do sistema da tocha em relação ao sistema atribuído ao elo 5 é mostrada na eq. (3.14). Essa matriz é colocada em termos

genéricos, uma vez que, nesse trabalho, busca-se elaborar um modelo que seja compatível com qualquer ferramenta.

$$ {}_0A^1 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9} $$

$$ {}_1A^2 = \begin{bmatrix} -s_2 & 0 & -c_2 & -l_2c_2 \\ c_2 & 0 & -s_2 & l_2s_2 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10} $$

$$ {}_2A^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11} $$

$$ {}_3A^4 = \begin{bmatrix} c_4 & 0 & s4 & 0 \\ s_4 & 0 & -c4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12} $$

$$ {}_4A^5 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.13} $$

$$ {}_5A^{Tool} = \begin{bmatrix} k_1 & k_4 & k_7 & k_{10} \\ k_2 & k_5 & k_8 & k_{11} \\ k_3 & k_6 & k_9 & k_{12} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14} $$

A matriz (${}_R\mathrm{T}^H$) pode ser vista na eq. (3.15).

$$ {}_RT^H = {}_0T^{Tool} = {}_0A^1 * {}_1A^2 * {}_2A^3 * {}_3A^4 * {}_4A^5 * {}_5A^{Tool} \tag{3.15} $$

Buscando a compatibilidade do modelo do manipulador com qualquer ferramenta, multiplicaram-se os lados direito e esquerdo da eq. (3.15) por $({}_5\mathrm{A}^{\mathrm{Tool}})^{-1}$. O resultado pode ser visto na eq. (3.16).

$$ {}_RT^{H^*} = {}_0A^5 = {}_0A^1 * {}_1A^2 * {}_2A^3 * {}_3A^4 * {}_4A^5 = \begin{bmatrix} a_1 & a_4 & a_7 & a_{10} \\ a_2 & a_5 & a_8 & a_{11} \\ a_3 & a_6 & a_9 & a_{12} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16} $$

(3.17)

$$a_1 = (\text{-}c_1 s_2 c_4 \text{-} s_1 s_4) c_5 \text{-} c_1 c_2 s_5$$

$$a_2 = (-s_1 s_2 c_4 + c_1 s_4) c_5 - s_1 c_2 s_5 \quad (3.18)$$

$$a_3 = c_2 c_4 c_5 - s_2 s_5 \quad (3.19)$$

$$a_4 = -(-c_1 s_2 c_4 - s_1 s_4) s_5 \text{-} c_1 c_2 c_5 \quad (3.20)$$

$$a_5 = -(-s_1 s_2 c_4 + c_1 s_4) s_5 - s_1 c_2 c_5 \quad (3.21)$$

$$a_6 = -c_2 c_4 s_5 - s_2 c_5 \quad (3.22)$$

$$a_7 = -c_1 s_2 s_4 + s_1 c_4 \quad (3.23)$$

$$a_8 = -s_1 s_2 s_4 - c_1 c_4 \quad (3.24)$$

$$a_9 = c_2 s_4 \quad (3.25)$$

$$a_{10} = -c_1 c_2 d_3 - c_1 l_2 s_2 + s_1 d_2 \quad (3.26)$$

$$a_{11} = -s_1 c_2 d_3 - s_1 l_2 s_2 - c_1 d_2 \quad (3.27)$$

$$a_{12} = -s_2 d_3 + l_2 c_2 + d_1 \quad (3.28)$$

Doravante, no escopo desse trabalho, será tratada a matriz ${}_R T^{H^*}$, uma vez que ela representa a transformação homogênea entre a base e o último elo do manipulador. Para transportar esse elo até a ponta de qualquer ferramenta, basta realizar a multiplicação entre essa matriz e a matriz ${}_5 A^{Tool}$.

Essa simplificação é válida uma vez que se assume que a geometria da ferramenta fixada ao elemento terminal do manipulador seja fixa, ou seja, a matriz ${}_5 A^{Tool}$ é constante.

3.3 CINEMÁTICA DIRETA-RPY

A partir da matriz de transformação é possível determinar a cinemática direta e inversa do manipulador (McKerrow, 1993). Vale lembrar que o processo de determinação da cinemática direta é relativamente mais fácil que o processo da cinemática inversa. A figura 3.7 apresenta as relações matemáticas da cinemática direta, ou seja, a partir da posição das juntas do manipulador – variáveis de juntas – é possível obter a posição do elemento terminal – variáveis cartesianas.

Figura 3.7: Cinemática Direta

A matriz de transformação geral é dada na eq. (3.29). Nela, R_θ (dimensão 3x3) indica a matriz de rotação, enquanto P (dimensão 3x1) traz o vetor de translação. Tal matriz permite observar de maneira direta o posicionamento do órgão terminal do manipulador no momento atual.

$$ {}_R T^{H^*} = \begin{bmatrix} R_\theta & P \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_x & y_x & z_x & p_x \\ x_y & y_y & z_y & p_y \\ x_z & y_z & z_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.29} $$

Busca-se, na cinemática direta, a orientação do objeto terminal com relação a um sistema de referência. Dessa forma, é natural que se especifique a orientação desse objeto como uma sequência de rotações em torno dos eixos ortogonais do sistema de coordenadas de referência. Há diversas maneiras de se representar orientação como sequencia de rotações, sendo que uma difere da outra em relação a quais eixos são realizadas as rotações para se atingir a orientação requerida (Spong, 2006). Dentre as várias formas de se representar orientação, destacam-se a representação baseada nos ângulos de Euler e a chamada RPY (Roll-Pitch-Yaw).

Neste trabalho, por questões práticas, escolheram-se os ângulos *RPY* (baseados em termos náuticos, originalmente em inglês: *Roll-Pitch-Yaw*). A matriz de orientação geral *RPY* é dada por uma sequência de três rotações: uma em torno do eixo *x* – *Yaw* –, uma no eixo *y* – *Pitch* – e outra no eixo *z* - *Roll*. Todas essas rotações seguem a chamada "regra da mão direita" (McKerrow, 1993).

A eq. (3.30) mostra a forma genérica dessa transformação.

$$RPY(\phi,\theta,\psi) = Rot(z,\phi) \times Rot(y,\theta) \times Rot(x,\psi)$$

$$= \begin{bmatrix} c_\phi c_\theta & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi & 0 \\ s_\phi c_\theta & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & 0 \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.30)$$

Pode-se observar que a eq.(3.30) não mostra valores referentes às translações do manipulador. Para que essas sejam incluídas, far-se-á uma multiplicação por uma matriz de translação genérica em relação aos eixos x, y e z do robô. A eq. (3.31) mostra a forma genérica da transformação RPY acrescida de translações genéricas nos eixos x, y e z.

$$RPY^C = Trans(T_x,T_y,T_z) \times Rot(z,\phi) \times Rot(y,\theta) \times Rot(x,\psi)$$

$$= \begin{bmatrix} c_\phi c_\theta & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi & T_x \\ s_\phi c_\theta & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & T_y \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.31)$$

Como ${}_R{T^H}^*$ é igual a *RPY*^{C}, tem-se, por meio de manipulações algébricas, as formas padrões dos valores dos ângulos *RPY*^{C} (ver eq. (3.32) e (3.33)). A partir deste ponto, utilizar-se-á a notação ϕ para ângulo "roll", θ, para ângulo "pitch" e ψ, para ângulo "yaw".

$$\phi = \arctan\left(\frac{x_y}{x_x}\right) \qquad \theta = \arctan\left(\frac{x_z}{\sqrt{{x_y}^2 + {x_x}^2}}\right) \qquad \psi = \arctan(\frac{y_z}{z_z}) \quad (3.32)$$

$$T_x = P_x \qquad T_y = P_y \qquad T_z = P_z \quad (3.33)$$

Por meio das eq. (3.32) e (3.33), é possível determinar a posição do elemento terminal do manipulador (a partir das variáveis de juntas). Entretanto, essas equações apresentam possíveis singularidades, as quais ocorrem em determinadas condições em que os valores x_y, x_x, y_z e z_z assumem o valor "0". Dessa forma, surgem indeterminações matemáticas (0/0), responsáveis por erros de cálculos.

Tal singularidade acontece quando x_z tem valor *1* ou *-1*, na eq. (3.29). Isso porque, nesse caso os quatro termos citados anteriormente assumem o valor zero, impossibilitando o cálculo dos valores corretos dos termos ϕ, θ e ψ.

Para solucionar esse problema, foi utilizada a metodologia proposta por (Slabaugh, 1999), modificada para *RPY*, definindo o valor de ϕ como zero e calculando os outros ângulos

sucessivamente. Neste caso, o valor de θ e de ψ é dado pelas eq. (3.34) ou eq. (3.35), em função do valor de x_z da matriz ${}_R T^H$ Caso o valor de x_z seja "1", a eq. (3.34) dá os valores de "θ" e "ψ". Por outro lado, se o valor de x_z for "-1", os valores são dados pela eq. (3.35) (Vasconcelos Filho et al, 2009).

$$\theta = -\frac{\pi}{2} \qquad \psi = \pi + \arctan(\frac{y_x}{z_x}) \tag{3.34}$$

$$\theta = \frac{\pi}{2} \qquad \psi = arctan(\frac{y_x}{z_x}) \tag{3.35}$$

A partir da definição dos parâmetros *RPY*, pode-se determinar que as variáveis de juntas sejam definidas pelo vetor: $Pn = [T_x \quad T_y \quad T_z \quad \phi \quad \theta \quad \psi]^T$.

Por outro lado, as variáveis de juntas são definidas por $\theta_n = [\theta_1 \quad \theta_2 \quad D_3 \quad \theta_4 \quad \theta_5]^T$, como mostrado na Tabela 3.1.

3.4 CINEMÁTICA INVERSA

A cinemática inversa determina as coordenadas de juntas do robô a partir da posição e da orientação final do sistema de coordenadas fixado no órgão terminal do manipulador (Figura 3.8). Esse cálculo é um processo complexo e pode ser desenvolvido de algumas maneiras, mas que deve levar em consideração alguns pontos, como:

a) *Existência de soluções*: Inicialmente, deve-se verificar se o ponto a ser alcançado pela ferramenta está dentro do espaço de trabalho do manipulador. Uma vez verificada essa condição, deve-se verificar se as restrições de movimentação das juntas são respeitadas, evitando danificar o robô e seu elemento terminal;
b) *Múltiplas soluções*: dada a natureza trigonométrica das equações que determinam as posições das juntas dos manipuladores, e sendo as mesmas funções pares, podem-se obter respostas redundantes, dentro da mesma faixa de operação da junta e dentro do espaço de trabalho. Desse modo, o controlador deve ter a capacidade de discernir entre essas opções.

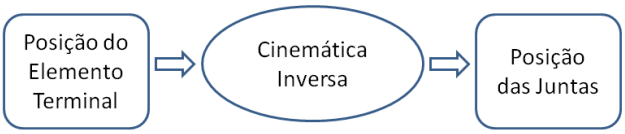


Figura 3.8: Cinemática Inversa

3.4.1 Solução analítica da cinemática inversa

Para se obterem as equações de cinemática inversa do manipulador, definem-se 16 equações[2], das quais 4 têm soluções triviais, reduzindo o número de equações a serem resolvidas para 12 (vide eq. (3.36) até (3.47)).

$$(\text{-}c_1 s_2 c_4 \text{-} s_1 s_4) c_5 \text{-} c_1 c_2 s_5 = x_x \tag{3.36}$$

$$-(-c_1 s_2 c_4 - s_1 s_4) s_5 \text{-} c_1 c_2 c_5 = y_x \tag{3.37}$$

$$-c_1 s_2 s_4 + s_1 c_4 = z_x \tag{3.38}$$

$$-c_1 c_2 d_3 - c_1 l_2 s_2 + s_1 d_2 = p_x \tag{3.39}$$

$$(-s_1 s_2 c_4 + c_1 s_4) c_5 - s_1 c_2 s_5 = x_y \tag{3.40}$$

$$-(-s_1 s_2 c_4 + c_1 s_4) s_5 - s_1 c_2 c_5 = y_y \tag{3.41}$$

$$-s_1 s_2 s_4 - c_1 c_4 = z_y \tag{3.42}$$

$$-s_1 c_2 d_3 - s_1 l_2 s_2 - c_1 d_2 = p_y \tag{3.43}$$

$$c_2 c_4 c_5 - s_2 s_5 = x_z \tag{3.44}$$

$$-c_2 c_4 s_5 - s_2 c_5 = y_z \tag{3.45}$$

$$c_2 s_4 = z_z \tag{3.46}$$

$$-s_2 d_3 + l_2 c_2 + d_1 = p_z \tag{3.47}$$

Pode-se observar que o sistema obtido para a cinemática inversa é bastante complexo, tendo em vista a necessidade de isolar as variáveis de juntas desejadas. Desta maneira, foram aplicados alguns algoritmos comuns na resolução de cinemática inversa (Spong, 2006).

[2] Igualando o lado direito da eq. (3.16) com o lado direito da eq. (3.29).

3.4.2 Isolando as variáveis do sistema

Como apontado na Tabela 3.1, as variáveis de juntas a serem determinados são θ_1, θ_2, d_3, θ_4 *e* θ_5. Dessa forma, dadas as equações (3.36) a (3.47), existem diversas maneiras de encontrar tais valores – isso porque, são 12 equações para 5 variáveis.

Por meio de observação, escolheu-se θ_2 como a primeira variável a ser isolada. Logo, as eq. (3.39) e (3.43) são elevadas ao quadrado e somadas, obtendo-se a eq. (3.48). Pode-se observar que a eq. (3.47) tem como variáveis apenas θ_2 e d_3. Desta maneira, através de manipulações simples, é possível obter d_3 em função de θ_2 a partir da eq. (3.47), como pode ser observado na eq. (3.49).

$$d_2{}^2 + c_2{}^2 d_3{}^2 + 2c_2 d_3 l_2 s_2 + l_2{}^2 s_2{}^2 = p_x{}^2 + p_y{}^2 \tag{3.48}$$

$$d_3 = -\frac{p_z - l_2 c_2 - d_1}{s_2} \tag{3.49}$$

Substituindo a eq. (3.49) na eq. (3.48) e realizando algumas operações matemáticas é possível obter c_2 e s_2. A partir destes valores é possível obter o valor de θ_2 (através de um arco-tangente):

$$c_2 = \frac{-k_2 \pm \sqrt{\Delta}}{2 * k_1} \tag{3.50}$$

$$s_2 = \frac{l_2 c_2 + (d_1 - p_z)}{\sqrt{p_x{}^2 + p_y{}^2 + (d_1 - p_z)^2 - d_2{}^2 - l_2{}^2}} = \frac{l_2 c_2 + (d_1 - p_z)}{\sqrt{k_1 - l_2{}^2}} \tag{3.51}$$

Onde:

$$k_1 = (p_x{}^2 + p_y{}^2 + (d_1 - p_z)^2 - d_2{}^2) \quad k3 = l_2{}^2 - p_x{}^2 - p_y{}^2 + d_2{}^2$$
$$k_2 = 2l_2(d_1 - p_z) \qquad \Delta = k_2{}^2 - 4k_1 k_3 \tag{3.52}$$

Substituindo a eq. (3.50) na eq. (3.49) é possível determinar o valor de d_3, caso s_2 seja diferente de zero. Caso s_2 seja igual a zero, novamente a partir da eq. (3.48), têm-se o valor segundo a eq. (3.53).

$$d_3 = \frac{\sqrt{{p_x}^2 + {p_y}^2 - {d_2}^2}}{c_2} \tag{3.53}$$

Pode-se observar que c_2 tem duas respostas, em função da raiz quadrada – vide a eq. (3.50). Nesse caso, como o valor máximo de θ_2 varia entre -30º e +30º (vide Tabela 3.2), não existe o risco de d_3 assumir um valor negativo, a partir da eq. (3.53). Tal afirmação é coerente com as restrições de movimentação desse eixo. Desse modo, o controlador é capaz de excluir a resposta fora do contexto.

Para determinar o valor de θ_1 são utilizadas, novamente, as eq. (3.39) e (3.43). Por meio de algumas manipulações matemáticas, obtêm-se os valores de c_1 e s_1 (vide eqs. (3.54) e (3.55)).

$$c_1 = \frac{-(p_y d_2 + c_2 d_3 p_x + l_2 s_2 p_x)}{(c_2 d_3 + l_2 s_2)^2 + {d_2}^2} \tag{3.54}$$

$$s_1 = \frac{p_x + c_1 c_2 d_3 + c_1 l_2 s_2}{d_2} \tag{3.55}$$

O valor de θ_4 é obtido a partir de c_4 e s_4 que são obtidos por meio de manipulações das eqs. (3.38), (3.42) e (3.46). As expressões resultantes são mostradas nas eqs. (3.56) e (3.57).

$$s_4 = -\frac{z_x c_1 + s_1 z_y}{s_2} \tag{3.56}$$

$$c_4 = -\frac{z_y + s_1 s_2 s_4}{c_1} \tag{3.57}$$

Novamente, existe o problema quando s_2 ou c_1 assumem o valor *0*. Caso s_2 seja zero, utiliza-se a eq. (3.58). Por outro lado, quando o valor de c_1 for zero, utilizar-se-á a eq. (3.59).

$$s_4 = \frac{z_z}{c_2} \tag{3.58}$$

$$c_4 = \frac{z_x}{s_1} \tag{3.59}$$

Aplicando-se a regra de *Cramer* nas equações (3.37) e (3.41), obtém-se s_5 e, a partir da eq. (3.45), obtém-se c_5. Os resultados são mostrados nas eqs. (3.60) e (3.61)

$$s_5 = \frac{s_1 y_x - c_1 y_y}{s_4} \tag{3.60}$$

$$c_5 = -\frac{(y_z + c_2 c_4 s_5)}{s_2} \tag{3.61}$$

Neste caso, para evitar as singularidades são utilizadas as eq. (3.62) e (3.63) respectivamente (tratando os problemas de indeterminação quando s_4 e s_2 assumem valor zero).

$$s_5 = -x_z s_2 - c_2 c_4 y_x \tag{3.62}$$

$$c_5 = -\frac{y_y + c_1 s_4 s_5}{s_1 c_2} \tag{3.63}$$

Entretanto, é fácil ver que a eq. (3.63) também apresenta singularidades. Para contorná-la, utiliza-se a eq. (3.64).

$$c_5 = -\frac{y_x}{c_1 c_2} \tag{3.64}$$

Através da função arco-tangente é possível determinar todos os ângulos, resolvendo assim a cinemática inversa do manipulador.

Obviamente, todos esses parâmetros e condições para uso de equações devem ser utilizados no controlador, de modo que este tenha condições de determinar corretamente a movimentação do manipulador. Na seção 6.2, apresentam-se os métodos computacionais que foram utilizados para validar os cálculos obtidos na solução analítica da cinemática inversa.

3.5 CINEMÁTICA DE VELOCIDADE – JACOBIANO

Considerando os aspectos de movimentação de um manipulador robótico, define-se como *caminho* a sequência de pontos no espaço pelos quais o seu elemento terminal passa, quando se desloca de um ponto A para um ponto B. Contudo, esse caminho, sem restrições, pode não descrever a *trajetória* desejada. Dessa forma, a trajetória do manipulador é um caminho percorrido, com restrições de tempo, incluindo parâmetros de velocidades e acelerações, além de posições ao longo do caminho (McKerrow, 1993).

Portanto, é necessário controlar parâmetros como velocidade e aceleração para garantir que a movimentação do elemento terminal seja correta, seguindo as restrições de tempo e respeitando a trajetória desejada. Nesse trabalho, a aceleração não será tratada, uma vez que as velocidades utilizadas são muito baixas (vide capítulo 2).

São comumente utilizadas duas formas de controle de movimento do manipulador (McKerrow, 1993):

- *Movimento interpolado das Juntas*: Busca-se que todas as juntas iniciem e finalizem sua movimentação ao mesmo tempo, através de mudanças nas variáveis de juntas, partindo de uma posição no espaço para outra. Apresenta como vantagem um movimento suave das juntas, bem como facilita o processo computacional de cálculos do manipulador. Entretanto, pode resultar em um caminho não previsto ou não desejado.
- *Movimento Cartesiano*: busca-se que o elemento terminal percorra um caminho descrito no espaço cartesiano, partindo de um ponto inicial e chegando a um ponto final com velocidade controlada. Para isso, há a necessidade de se interpolar pontos intermediários, pelos quais o elemento terminal deverá passar de modo a garantir o seguimento do caminho cartesiano. Para cada ponto é necessário calcular a cinemática inversa, de modo a produzir uma sequência de posições para cada variável de junta, o que torna o método muito mais complexo do ponto de vista computacional.

O caminho a ser percorrido pelo manipulador é obtido pela substituição da sequência de ângulos das juntas na cinemática direta do manipulador. Já a velocidade e a aceleração demandam a utilização de um modelo cinemático de velocidade e aceleração do manipulador.

3.5.1 Movimento Diferencial

No caso de um manipulador, mudanças infinitesimais nos elos do manipulador afetam de maneira infinitesimal na posição do elemento terminal do manipulador. Esse efeito é calculado por meio da chamada “matriz Jacobiana” do manipulador. Para tanto, é necessário entender inicialmente o conceito de transformação diferencial. A Figura 3.9 apresenta uma representação de um movimento infinitesimal translacional (a) e rotacional

(b). Importante ressaltar que as transformações diferenciais são definidas em relação a um sistema de coordenadas. Esses podem ser vistos matematicamente nas eq. (3.65) e (3.66).

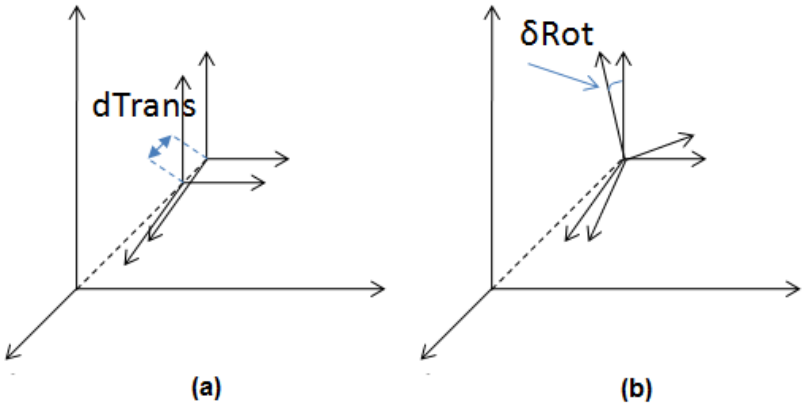


Figura 3.9: Movimento diferencial: (a) translacional; (b) rotacional

$$dTrans = trans(dx, dy, dz) \tag{3.65}$$

$$\delta Rot = Rot(\psi, \delta\psi).Rot(\theta, \delta\theta).Rot(\phi, \delta\phi) \tag{3.66}$$

Combinando os vetores obtidos nas eq. (3.65) e (3.66), tem-se uma matriz coluna, chamada Vetor de Movimento Diferencial **(McKerrow, 1993)**, representado na equação (3.67). Esse vetor representa um conjunto de movimentações infinitesimais, aplicadas em relação a cada um dos três eixos de um sistema de coordenadas cartesianas fixado em um manipulador.

$$D = [dx \quad dy \quad dz \quad \delta\psi \quad \delta\theta \quad \delta\phi]^T \tag{3.67}$$

De maneira análoga, o vetor de movimento diferencial, em termo das coordenadas das juntas para o manipulador desenvolvido nesse projeto é dado por:

$$D_q = [d\theta_1 \quad d\theta_2 \quad dd_3 \quad d\theta_4 \quad d\theta_5]^T \tag{3.68}$$

3.5.2 Jacobiano do Manipulador

As equações de cinemática direta definem uma função entre variáveis de posição e orientação no espaço Cartesiano e as variáveis do espaço de juntas. Já as relações de velocidade podem ser obtidas pela derivação das funções da cinemática direta, o que resulta, quando as derivadas são organizadas em forma matricial, na chamada matriz Jacobiana. O Jacobiano é uma função de valores matriciais e pode ser vista como uma versão vetorial de derivadas ordinárias de uma função escalar **(Spong, 2006)**. Já o

Jacobiano inverso relaciona a velocidade do órgão terminal no espaço cartesiano às velocidades das juntas. A Figura 3.10 exemplifica a função do jacobiano e do jacobiano inverso.

Pode-se observar pela Figura 3.10 que o Jacobiano Inverso realiza a transformação de diferenciais no sentido inverso, ou seja, dados valores diferenciais descritos no espaço cartesiano, os valores diferenciais correspondentes no espaço das juntas são obtidos por meio da aplicação do Jacobiano Inverso.

O Jacobiano aparece como uma ferramenta importante para o cálculo de diversos aspectos ligados ao processo de manipulação robótica: (a) planejamento de trajetória, (b) execução da trajetória de maneira suave e segura, (c) determinação de configurações singulares, (d) execução de movimentos coordenados e (e) derivação das equações dinâmicas de movimento **(Spong, 2006)**.

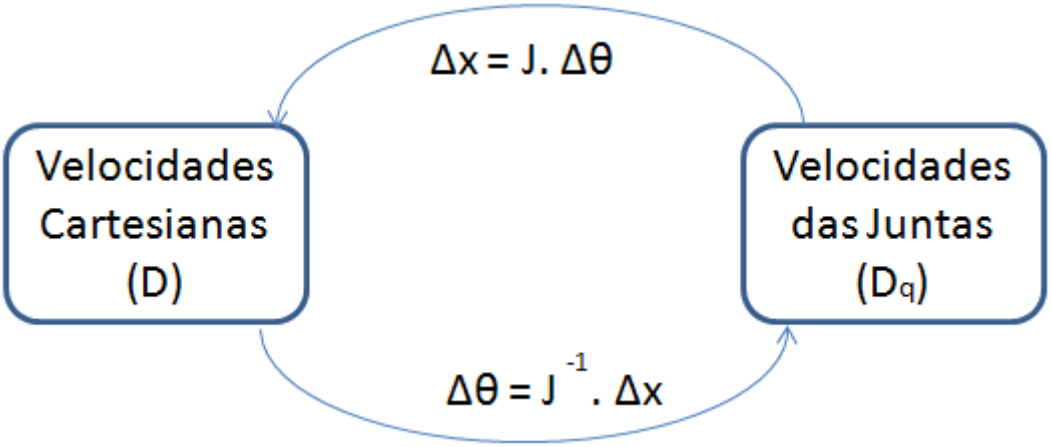


Figura 3.10: Aplicação do Jacobiano de um manipulador

A equação do Jacobiano (J), para um manipulador genérico de 6 graus de liberdade é mostrada na equação (3.69).

$$D = J.D_q = [J_1 \quad J_2 \quad J_3 \quad J_4 \quad J_5 \quad J_6].D_q \tag{3.69}$$

Em que J_i são matrizes com dimensão 6x1

Para calcular o jacobiano, pode-se optar pelo uso de diferenciação, vetores ou utilizar coordenadas homogêneas **(Paul, 1989)**.

Sabe-se que o elemento terminal do manipulador apresenta um componente de velocidade linear (nos três eixos) e um de velocidade angular (também nos três eixos). Dessa forma, através de manipulações algébricas, pode-se ver a forma comum do Jacobiano na eq. (3.70) **(Spong, 2006)**, onde J_v (velocidade linear do elemento terminal) e J_w (velocidade angular do elemento terminal) são matrizes *3x6*, sendo o número de colunas determinado pelo número de eixos atuados (graus de mobilidade) do manipulador. Desta maneira, pode ser verificado que o Jacobiano de um manipulador de 6 graus de mobilidade é uma matriz *6x6*.

$$J_{6x6} = \begin{bmatrix} J_{v\,3x6} \\ J_{w\,3x6} \end{bmatrix}_{6x6} \tag{3.70}$$

Cada coluna de J_v e de J_w possui uma regra para sua determinação. Essa regra varia com a posição e com o tipo da junta – rotacional ou prismática. Para um vetor J qualquer, de dimensões *6xN*, sendo N o número de graus de liberdade, têm-se as equações (3.71) e (3.72), indicando o vetor J_v genérico e suas condições de diferenciação **(McKerrow, 1993) (Spong, 2006)**.

$$J_v = [J_{v1} \quad J_{v2} \quad \cdots \quad J_{vN}] \tag{3.71}$$

$$J_{vi} = \begin{bmatrix} \dfrac{\partial x_i}{\partial q_i} \\ \dfrac{\partial y_i}{\partial q_i} \\ \dfrac{\partial z_i}{\partial q_i} \end{bmatrix} \tag{3.72}$$

Em que q_i , $i=1..N$ são as variáveis de juntas.

Cada linha expressa na eq. (3.72) pode ser vista como a velocidade linear do elemento terminal do manipulador caso todas as juntas estejam imóveis e a junta i com velocidade unitária **(McKerrow, 1993)**.

Já a equação (3.73) indica o vetor genérico J_w, com suas condições de determinação apresentadas na equação (3.74) **(Spong, 2006)**.

$$J_w = [J_{w1} \quad J_{w2} \quad \cdots \quad J_{wN}] \tag{3.73}$$

$$J_{wi} = [\rho_1 Z_0 \quad \rho_2 Z_1 \quad \cdots \quad \rho_N Z_{N-1}] \tag{3.74}$$

Importante ressaltar que o vetor Z_i é dado pela equação (3.75).

$$Z_{i-1} = {}^{0}R_{i-1}.\vec{K} \tag{3.75}$$

Na eq. (3.75), $\vec{\boldsymbol{K}}$ representa o vetor unitário do sistema de coordenadas *i-1*, expresso no sistema de coordenadas da base. Já $\boldsymbol{\rho_i}$ é igual a *1* se a junta *i* é rotacional, enquanto seu valor é 0 (zero), se a junta *i* for prismática.

A partir das eq.(3.72) e (3.74), infere-se que a equação que determina o Jacobiano do manipulador deste trabalho pode ser vista na eq. (3.76). Como o manipulador elaborado possui 5 graus de liberdade, observa-se que a matriz *J* é *6x5*. Outra observação é a de que as três últimas linhas da terceira coluna (Z_2) têm valor zero, em função da junta prismática.

Conforme comentado na seção Erro! Fonte de referência não encontrada., não foi calculado, nesse capitulo, a influência da ferramenta no calculo do Jacobiano do manipulador. Desse modo, considerou-se como elemento terminal do manipulador o centro do eixo 5..

$$J = \begin{bmatrix} \frac{\partial P_x}{\partial \theta_1} & \frac{\partial P_x}{\partial \theta_2} & \frac{\partial P_x}{\partial d_3} & \frac{\partial P_x}{\partial \theta_4} & \frac{\partial P_x}{\partial \theta_5} \\ \frac{\partial P_y}{\partial \theta_1} & \frac{\partial P_y}{\partial \theta_2} & \frac{\partial P_y}{\partial d_3} & \frac{\partial P_y}{\partial \theta_4} & \frac{\partial P_y}{\partial \theta_5} \\ \frac{\partial P_z}{\partial \theta_1} & \frac{\partial P_z}{\partial \theta_2} & \frac{\partial P_z}{\partial d_3} & \frac{\partial P_z}{\partial \theta_4} & \frac{\partial P_z}{\partial \theta_5} \\ Z_{0_1} & Z_{1_1} & 0 & Z_{3_1} & Z_{4_1} \\ Z_{0_2} & Z_{1_2} & 0 & Z_{3_2} & Z_{4_2} \\ Z_{0_3} & Z_{1_3} & 0 & Z_{3_3} & Z_{4_3} \end{bmatrix} \tag{3.76}$$

Em que:

$$\frac{\partial P_x}{\partial \theta_1} = s_1 c_2 d_3 + s_1 s_2 l_2 + c_1 d_2 \tag{3.77}$$

$$\frac{\partial P_y}{\partial \theta_1} = -c_1 c_2 d_3 - c_1 s_2 l_2 + s_1 d_2 \tag{3.78}$$

$$\frac{\partial P_z}{\partial \theta_1} = 0 \tag{3.79}$$

$$\frac{\partial P_x}{\partial \theta_2} = s_2 c_1 d_3 - c_1 c_2 l_2 \tag{3.80}$$

$$\frac{\partial P_y}{\partial \theta_2} = s_1 s_2 d_3 - s_1 c_2 l_2 \tag{3.81}$$

$$\frac{\partial P_z}{\partial \theta_2} = -c_2 d_3 - s_2 l_2 \tag{3.82}$$

$$\frac{\partial P_x}{\partial d_3} = -c_1 c_2 \tag{3.83}$$

$$\frac{\partial P_y}{\partial d_3} = -s_1 c_2 \tag{3.84}$$

$$\frac{\partial P_z}{\partial d_3} = -s_2 \tag{3.85}$$

$$\frac{\partial P_x}{\partial \theta_4} = \frac{\partial P_y}{\partial \theta_4} = \frac{\partial P_z}{\partial \theta_4} = 0 \tag{3.86}$$

$$\frac{\partial P_x}{\partial \theta_5} = \frac{\partial P_y}{\partial \theta_5} = \frac{\partial P_z}{\partial \theta_5} = 0 \tag{3.87}$$

$$Z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.88}$$

$$Z_1 = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix} \tag{3.89}$$

$$Z_3 = \begin{bmatrix} -c_1 c_2 \\ -s_1 c_2 \\ -s_2 \end{bmatrix} \tag{3.90}$$

$$Z_4 = \begin{bmatrix} -c_1 s_2 s_4 + s_1 c_4 \\ -s_1 s_2 s_4 - c_1 c_4 \\ c_2 s_4 \end{bmatrix} \tag{3.91}$$

3.5.3 Jacobiano Inverso do Manipulador

Como foi visto anteriormente, através do Jacobiano, é possível determinar o vetor de velocidades cartesianas do manipulador, utilizando as velocidades das juntas. Isso pode ser visto na eq. (3.92). Entretanto, como a velocidade do elemento terminal é conhecida, é necessário determinar a velocidade das juntas do manipulador para que ele seja controlável.

$$\dot{D} = J.\dot{D_q} \tag{3.92}$$

Para tanto, é necessário determinar uma forma de calcular a velocidade das juntas do manipulador, em função da velocidade do elemento terminal.

Existem diversas formas de se obter o Jacobiano Inverso, dentre as quais citam-se a derivação da cinemática inversa e a inversão da matriz do Jacobiano direto. A primeira opção torna-se bastante difícil, tendo em vista a complexidade da solução da cinemática inversa, a qual envolve funções transcendentais além de problemas com singularidades. A

segunda opção somente é possível diretamente quando a matriz do Jacobiano é quadrada, como no caso de um manipulador com 6 graus de liberdade.

Desse modo, aplicando-se a segunda metodologia, é necessário inverter a matriz do Jacobiano para obter-se o vetor de velocidades das juntas, como visto na eq. (3.93).

$$J^{-1}\dot{D} = \dot{D_q} \tag{3.93}$$

Entretanto, como citado, a matriz do Jacobiano do manipulador construído tem dimensão *6x5*. Dessa forma, ela não é *inversível*. Logo, utilizou-se uma ferramenta matemática chamada *pseudo-inversa* **(McKerrow, 1993)**.

Partindo-se da eq. (3.97), caso ambos os lados da equação sejam pré-multiplicados pela transposta da matriz do Jacobiano direto, obtém-se do lado esquerdo um novo vetor 5x1 e do lado direito uma nova matriz quadrada 5x5 multiplicando o vetor $\boldsymbol{\dot{D_q}}$. Pré-multiplicando-se agora ambos os lados da equação pela inversa da matriz quadrada 5x5, obtém-se a equação em que o vetor $\boldsymbol{\dot{D_q}}$ aparece como função de uma matriz 5x6 multiplicada pelo vetor $\boldsymbol{\dot{D}}$.

Esta é a chamada "pseudo-inversa à esquerda" e o desenvolvimento descrito acima é mostrado nas equações (3.94), (3.95) e (3.96).

$$[\dot{D}]_{6x1} = [J]_{6x5}[\dot{D_q}]_{5x1} \tag{3.94}$$

$$[J^T]_{5x6}\left[\dot{D}\right]_{6x1} = [[J^T]_{5x6}[J]_{6x5}]_{5x5}[\dot{D_q}]_{5x1} \tag{3.95}$$

$$[\dot{D_q}]_{5x1} = [[[J^T]_{5x6}[J]_{6x5}]_{5x5}]^{-1}[J^T]_{5x6}\left[\dot{D}\right]_{6x1} \tag{3.96}$$

Então, é possível obter a eq. (3.97), onde se pode calcular a velocidade das juntas a partir da velocidade cartesiana do manipulador.

$$\dot{D_q} = (J^T.J)^{-1}.J^T.\dot{D} \tag{3.97}$$

O vetor de velocidades das juntas foi obtido por meio de ferramentas matemáticas, como o Matlab e seus componentes são explicitados a seguir, entre as equações (3.98) e (3.102). Pode-se ver que essas equações resultantes são bastante complexas e de difícil implementação, em função do grande número de operações trigonométricas que devem ser realizadas.

$$\dot{\theta_1} = ((\dot{Px})\,(+c_1s_2(+d_2s_2+d_3c_4s_4c_2)+s_1l_2s_2+s_1d_3(+d_3(+d_3c_2+l_2s_2)+c_2)+c_4c_4(-c_1d_2-s_1c_2d_3+c_1c_2c_2d_2-s_1l_2s_2))+(\dot{Py})\,(+c_1d_3(-d_3l_2s_2-d_3d_3c_2-c_2)+s_1(+s_2(+d_2s_2+d_3c_4s_4c_2))-c_1l_2s_2+c_4c_4(-s_1d_2s_2s_2+c_1(+l_2s_2+c_2d_3)))+(\dot{Pz})\,(+c_2(+s_4(-d_2s_2s_4-c_2c_4d_3)))+(\dot{Rx})\,(-s_1d_2s_2d_3+c_4s_4d_3(-d_2c_1s_2s_2-c_2d_3s_1)+c_4c_4(+s_2d_3(-c_1c_2d_3+d_2s_1)))+(\dot{Ry})\,(+d_3(+c_4s_4(-d_2s_1s_2s_2+c_2d_3c_1)+c_1d_2s_2+c_4c_4s_2(-s_1c_2d_3-d_2c_1))+(\dot{Rz})\,(+d_3c_4c_2(+d_3c_2c_4+d_2s_2s_4)))/\text{divisor} \quad (3.98)$$

$$\dot{\theta_2} = (\dot{Px}\,(+c_4s_4c_2(+c_1d_2+c_2(+s_1d_3-c_1c_2d_2))+c_2s_2(+s_1(+c_4s_4l_2-d_2d_3d_3)+c_2d_3c_1(-l_2l_2+c_4c_4+d_3d_3))+d_3l_2s_2(+c_1(+2c_2s_2d_3+l_2)-d_2s_1s_2))+\dot{Py}\,(+s_1c_2c_2(+d_3(+s_2(-l_2l_2+d_3d_3+c_4c_4)-2c_2d_3l_2)-c_2d_2c_4s_4)+c_4s_4c_2(+s_1d_2+c_1(-c_2d_3-l_2s_2))+d_2d_3c_1(-l_2c_2c_2+s_2d_3c_2+l_2)+s_1d_3l_2(+l_2s_2+2c_2d_3))+\dot{Pz}\,(+c_2(+c_2(+c_2d_3(-d_3d_3+l_2l_2-c_4c_4)+s_2(-d_2c_4s_4-2d_3d_3l_2))-l_2l_2d_3))+\dot{Rx}\,(+c_4s_4(+c_2c_2c_1(-2c_2d_3l_2+s_2(-d_2d_2+d_3d_3-l_2l_2))+c_1(+s_2(+l_2l_2+d_2d_2)+2c_2d_3l_2)+c_2d_2s_2d_3s_1)+c_4c_4(+s_1(-l_2l_2-d_2d_2)+c_2(-c_2c_2c_2d_2d_3c_1+d_2d_3c_1+s_1(+c_2(+d_2d_2+l_2l_2-d_3d_3)-2d_3l_2s_2)))+s_1(+d_2d_2+l_2l_2+2c_2d_3l_2s_2+c_2c_2(-d_2d_2-l_2l_2+d_3d_3)))+\dot{Ry}\,(+c_2(+c_2(+c_1d_2d_2-c_1d_3d_3+c_1l_2l_2+c_4(+s_1s_2s_4(-d_2d_2+d_3d_3-l_2l_2)+s_1c_2(+d_3(-d_2c_4-2l_2s_4))+c_4c_1(+d_3d_3-l_2l_2)))+c_4(+c_1(-s_4d_2s_2d_3-c_2d_2d_2c_4+2c_4d_3l_2s_2)+d_3s_1(+2l_2s_4+d_2c_4))-2c_1d_3l_2s_2)+c_4(+s_1s_2s_4(+d_2d_2+l_2l_2)+c_4c_1(+l_2l_2+d_2d_2))+c_1(-d_2d_2-l_2l_2))+\dot{Rz}\,(+c_2(+c_2(+c_2c_4s_4(+l_2l_2+d_2d_2-d_3d_3)+c_4s_2d_3(-2s_4l_2-d_2c_4))+c_4s_4(-l_2l_2-d_2d_2)))))/\text{divisor}; \quad (3.99)$$

$$\dot{D_3} = -(\dot{Px}\,(+c_2(+c_2(+c_2c_1(-l_2l_2+d_3d_3d_3d_3+d_3d_3-d_2c_4s_4l_2+l_2l_2c_4c_4-3l_2l_2d_3d_3)+c_1s_2d_3(+l_2(+2+3d_3d_3-l_2l_2-c_4c_4)+d_2c_4s_4)+d_3(+d_2s_1(+l_2l_2-1-d_3d_3)+c_4s_4l_2s_1))+l_2(+c_4(+s_1s_2(+l_2s_4+d_2c_4)+c_1(+s_4d_2-l_2c_4))+d_3d_3(-2d_2s_2s_1+3c_1l_2)-d_2s_1s_2+c_1l_2)+c_2d_2c_4c_4s_1d_3)+l_2l_2d_3(-d_2s_1+c_1l_2s_2))+\dot{Py}\,(+c_2(+c_2(+d_3(+s_1(+d_3(+c_2(-3l_2l_2+1+d_3d_3)+3l_2s_2d_3)+d_2c_4s_4s_2+2l_2s_2)+c_1(+d_2(+1-l_2l_2-c_4c_4+d_3d_3)-c_4s_4l_2))+s_1l_2(+c_2(+c_4(+l_2c_4-d_2s_4)-l_2)+s_2d_3(-c_4c_4-l_2l_2)))+l_2(+s_1(+l_2(-c_4c_4+3d_3d_3+1)+c_4s_4d_2)+c_1s_2(+d_2(+s_4s_4+2d_3d_3)-c_4l_2s_4)))+l_2l_2d_3(+d_2c_1+s_1l_2s_2))+\dot{Pz}\,(+c_2(+c_2(+d_3(+d_3(+s_2(+1+d_3d_3-3l_2l_2)-3c_2d_3l_2)+c_2(+c_4(-d_2s_4+l_2c_4)+l_2(-2+l_2l_2)))+l_2s_2(-l_2+c_4(-d_2s_4+l_2c_4)))+d_3(+l_2(-l_2l_2-2c_4c_4+2d_3d_3+2)+2d_2c_4s_4))+l_2l_2s_2(+d_3d_3+s_4s_4)+s_2d_2d_2s_4s_4)+\dot{Rx}\,(+c_2(+c_4(+c_2(+d_2(+d_3(+d_3(+s_2c_1c_4+s_4s_1)-d_2c_2c_1s_4)-d_2l_2c_1s_2s_4+d_2c_4l_2s_1)+l_2(+l_2l_2c_4s_1-l_2l_2c_1s_2s_4+d_3c_1(+s_2d_3s_4-2l_2c_2s_4-c_2d_2c_4)))+d_3(+l_2(+c_4(+s_1(-2l_2s_2-c_2d_3)+d_2c_1)+s_4(+d_2s_2s_1+2l_2c_1))+d_2d_2(-c_4s_2s_1+c_1s_4)))+l_2s_1(+c_2(-l_2l_2+d_3d_3-d_2d_2)+2d_3l_2s_2)+s_1d_2d_2s_2d_3)+l_2(+c_4(+c_1s_2s_4(+d_2d_2+l_2l_2)+c_4s_1(-d_2d_2- \quad (3.100)$$

$l_2l_2))+s_1(+l_2l_2+d_2d_2)))+\dot{Ry}$ $(+c_2(+c_4(+s_4(+s_1(+c_2(+d_3(+c_2(-d_2d_3-2l_2l_2)+l_2s_2d_3)+s_2l_2(-l_2l_2-d_2d_2))+d_3(+d_2d_2+2l_2l_2))+d_3c_1d_2(-l_2s_2-c_2d_3))+c_4(+s_1(+d_3d_2(+c_2(-c_2l_2+s_2d_3)+l_2))+c_1(+s_2d_3(+2l_2l_2+d_2d_2)+c_2l_2(-l_2l_2-d_2d_2+d_3d_3))))+c_1(+l_2(-2d_3l_2s_2+c_2(+d_2d_2-d_3d_3+l_2l_2))-d_2d_2s_2d_3))+l_2(+c_1(+c_4c_4(+d_2d_2+l_2l_2)-l_2l_2-d_2d_2)+c_4s_4s_2s_1(+l_2l_2+d_2d_2)))+\dot{Rz}$ $(+c_2(+c_2c_4(+d_3(+d_2c_4(-l_2s_2-c_2d_3)+s_4s_2(-d_2d_2-2l_2l_2))+c_2s_4l_2(+l_2l_2-d_3d_3+d_2d_2))+c_4s_4l_2(-l_2l_2-d_2d_2))))/divisor$

$\dot{\theta_4}$=-($\dot{Px}$ $(+s_2c_4c_4s_1c_2d_3-s_2c_1d_2-d_3d_3s_1l_2-s_2c_1c_2c_2d_2c_4c_4+s_1l_2c_2c_2-s_2s_1c_2d_3-s_2d_3d_3d_3s_1c_2+c_4c_4s_1l_2+s_2c_1c_2c_2d_2+s_2c_4c_4c_1d_2-s_1l_2+c_1d_3c_4c_2c_2c_2s_4+d_3d_3s_1l_2c_2c_2-c_1d_3c_4c_2s_4-c_4c_4s_1l_2c_2c_2)+$ $\dot{Py}$ $(+s_2c_4c_4s_1d_2+d_3d_3c_1l_2-c_4c_4c_1l_2-s_2c_4c_4c_1c_2d_3-s_2s_1d_2-s_2s_1c_2c_2d_2c_4c_4+s_2s_1c_2c_2d_2-d_3d_3c_1l_2c_2c_2+s2c_1c_2d_3+s_2d_3d_3d_3c_1c_2+c_1l_2-s_1d_3c_4c_2s_4-c_1l_2c_2c_2+s_1d_3c_4c_2c_2c_2s_4+c_4c_4c_1l_2c_2c_2)+$ $\dot{Pz}$ $(-c_2d_2c_4c_4+c_2c_2c_2d_2c_4c_4+c_2d_2-c_2c_2c_2d_2+s_2c_2c_2c_4s_4d_3)+$ $\dot{Rx}$ $(-c_1c_2c_2c_2d_3d_3c_4c_4+s_2c_2s_4d_3d_3s_1c_4-c_1c_2d_2d_2c_4c_4+c_1c_2l_2l_2d_3d_3+s_1d_2d_3-2c_1c_2c_2d_3l_2s_2c_4c_4+d_2c_4c_4d_3s_1c_2c_2+c_1c_2c_2c_2d_3d_3+2c_1c_2c_2d_3l_2s_2-c_1c_2c_2c_2l_2l_2-c_1c_2c_2c_2d_2d_2+c_1l_2l_2c_2+c_1c_2d_2d_2+c_1c_2c_2c_2d_3d_3d_3d_3-c_1c_2l_2l_2c_4c_4+c_1c_2c_2c_2d_2d_2c_4c_4-s_1d_2d_3c_2c_2-d_2c_4c_4d_3s_1+c_1c_2d_3d_3c_4c_4+2c_1l_2c_2c_2d_3d_3d_3s_2+c_1c_2c_2d_2c_4s_4s_2d_3-c_1l_2l_2c_2c_2c_2d_3d_3+c_1c_2c_2c_2l_2l_2c_4c_4+d_2c_4s_2d_3c_1s_4)+$ $\dot{Ry}$ $(-c_1d_2d_3-s_2c_2s_4d_3d_3c_1c_4-2s_1c_2c_2d_3l_2s_2c_4c_4+s_1c_2c_2d_2c_4s_4s_2d_3+s_1c_2c_2c_2d_2d_2c_4c_4+s_1c_2c_2c_2l_2l_2c_4c_4+2s_1l_2c_2c_2d_3d_3d_3s_2-s_1l_2l_2c_2c_2c_2d_3d_3-s_1c_2l_2l_2c_4c_4+s_1c_2l_2l_2d_3d_3+2s_1c_2c_2d_3l_2s_2+s_1c_2d_3d_3c_4c_4-d_2c_4c_4d_3c_1c_2c_2+s_1c_2c_2c_2d_3d_3d_3d_3+s_1l_2l_2c_2-s_1c_2d_2d_2c_4c_4-s_1c_2c_2c_2l_2l_2-s_1c_2c_2c_2d_3d_3c_4c_4+d_2c_4c_4d_3c_1-s_1c_2c_2c_2d_2d_2+c_1d_2d_3c_2c_2+s_1c_2c_2c_2d_3d_3+d_2c_4s_2d_3s_1s_4+s_1c_2d_2d_2)+$ $\dot{Rz}$ $(+s_2d_2d_2+s_2l_2l_2-s_2l_2l_2c_4c_4-s_2d_2d_2c_4c_4-s_2l_2l_2c_2c_2+s_2l_2l_2c_2c_2c_4c_4+s_2c_2c_2d_3d_3+2c_2d_3d_3d_3l_2+2c_2d_3l_2-2c_2c_2c_2d_3d_3d_3l_2-2c_2c_2c_2d_3l_2+2c_2c_2c_2d_3l_2c_4c_4-s_2c_2c_2d_2d_2-s_2c_2c_2d_3d_3c_4c_4-s_2l_2l_2c_2c_2d_3d_3+s_2c_2c_2d_3d_3d_3d_3+s_2l_2l_2d_3d_3-2c_2d_3l_2c_4c_4+s_2c_2c_2d_2d_2c_4c_4+d_2d_3c_4c_2s_4-d_2d_3c_4c_2c_2c_2s_4))/divisor$

$\dot{\theta_5}$=-($\dot{Px}$ $(+c_2c_2s_4s_1d_3-2c_1c_4c_2c_2c_2d_3d_3l_2+2c_1c_4c_2d_3d_3l_2+d_2c_4d_3s_1l_2c_2c_2-d_2c_4d_3s_1l_2+c_2s_4s_1l_2s_2+c_1c_4l_2l_2s_2d_3+c_1c_4c_2c_2d_3d_3d_3s_2-d_2c_4s_2d_3d_3s_1c_2+c_2c_2s_4d_3d_3d_3s_1+c_2s_4c_1d_2-c_1c_4c_2c_2d_3l_2l_2s_2+c_2s_4d_3d_3s_1l_2s_2-c_1c_2c_2c_2s_4d_2+c_1c_4c_2c_2d_3s_2)+$ $\dot{Py}$ $(-c_2c_2s_4c_1d_3+d_2c_4s_2d_3d_3c_1c_2+2s_1c_4c_2d_3d_3l_2-c_2c_2s_4d_3d_3d_3c_1-2s_1c_4c_2c_2c_2d_3d_3l_2-s_1c_2c_2c_2s_4d_2+s_1c_4c_2c_2d_3d_3d_3s_2+s_1c_4c_2c_2d_3s_2+d_2c_4d_3c_1l_2-s_1c_4c_2c_2d_3l_2l_2s_2-$

(3.101)

$$d_2c_4d_3c_1l_2c_2c_2\text{-}c_2s_4d_3d_3c_1l_2s_2\text{+}s_1c_4l_2l_2s_2d_3\text{-}c_2s_4c_1l_2s_2\text{+}c_2s_4s_1d_2)\text{+}\dot{P}z\ (\text{-}c_2c_2c_2c_4d_3\text{-}$$
$$c_2c_4l_2l_2d_3\text{-}2c_2c_2c_4d_3d_3l_2s_2\text{-}c_2c_2s_2s_4d_2\text{-}c_2c_2c_2c_4d_3d_3d_3\text{+}c_2c_2c_2c_4l_2l_2d_3)\text{+}\dot{R}x$$
$$(c_2c_2d_3d_3c_1s_2s_4\text{-}l_2l_2c_2c_2d_3d_3c_1s_2s_4\text{+}c_2c_2d_3d_3d_3d_3c_1s_2s_4\text{+}c_1c_2d_2c_4d_3\text{-}$$
$$c_2c_2d_3d_3s_1c_4\text{+}2c_2d_3d_3d_3l_2c_1s_4\text{-}c_1c_2c_2c_2d_2c_4d_3\text{-}s_1c_2s_4d_2s_2d_3\text{+}d_2d_2c_1s_2s_4\text{+}2c_2d_3l_2c_1s_4\text{-}$$
$$l_2l_2c_2c_2c_1s_2s_4\text{-}2c_2c_2c_2d_3l_2c_1s_4\text{+}l_2l_2d_3d_3c_1s_2s_4\text{-}l_2l_2d_3d_3s_1c_4\text{+}l_2l_2c_1s_2s_4\text{-}$$
$$c_2c_2d_2d_2c_1s_2s_4\text{+}l_2l_2c_2c_2d_3d_3s_1c_4\text{-}c_2c_2d_3d_3d_3d_3s_1c_4\text{+}c_1c_2d_2c_4d_3\text{-}$$
$$c_2c_2d_3d_3s_1c_4\text{+}2c_2d_3d_3d_3l_2c_1s_4\text{-}c_1c_2c_2c_2d_2c_4d_3\text{-}s_1c_2s_4d_2s_2d_3\text{+}d_2d_2c_1s_2s_4\text{-}$$
$$2c_2c_2c_2d_3d_3d_3l_2c_1s_4\text{-}2c_2d_3d_3d_3l_2s_2s_1c_4)\text{+}\dot{R}y\ (\text{+}l_2l_2d_3d_3s_1s_2s_4\text{+}s_1c_2d_2c_4d_3\text{+}l_2l_2s_1s_2s_4\text{-}$$
$$l_2l_2c_2c_2d_3d_3s_1s_2s_4\text{-}l_2l_2c_2c_2d_3d_3c_1c_4\text{+}2c_2d_3d_3d_3l_2s_2c_1c_4\text{-}l_2l_2c_2c_2s_1s_2s_4\text{-}$$
$$2c_2c_2c_2d_3d_3d_3l_2s_1s_4\text{+}c_2c_2d_3d_3s_1s_2s_4\text{+}2c_2d_3d_3d_3l_2s_1s_4\text{-}$$
$$c_2c_2d_2d_2s_1s_2s_4\text{+}2c_2d_3l_2s_1s_4\text{+}c_2c_2d_3d_3d_3s_1s_2s_4\text{-}2c_2c_2c_2d_3l_2s_1s_4\text{+}c_1c_2s_4d_2s_2d_3\text{-}$$
$$s_1c_2c_2c_2d_2c_4d_3\text{+}c_2c_2d_3d_3d_3d_3c_1c_4\text{+}l_2l_2d_3d_3c_1c_4\text{+}c_2c_2d_3d_3c_1c_4\text{+}d_2d_2s_1s_2s_4)\text{+}\dot{R}z$$
$$(\text{+}c_2c_2c_2s_4l_2l_2\text{+}c_2c_2c_2s_4d_2d_2\text{-}c_2s_4l_2l_2d_3d_3\text{-}c_2c_2c_2s_4d_3d_3\text{-}c_2c_2c_2s_4d_3d_3d_3d_3\text{-}d_2c_4s_2d_3c_2c_2\text{-}$$
$$c_2s_4d_2d_2\text{+}c_2c_2c_2s_4l_2l_2d_3d_3\text{-}c_2s_4l_2l_2\text{-}2c_2c_2s_4d_3l_2s_2\text{-}2c_2c_2s_4d_3d_3d_3l_2s_2))/\text{divisor}$$
$$\text{divisor}=(\text{+}c_2(\text{+}d_3(\text{+}2s_2c_4(\text{-}l_2c_4\text{+}d_2s_4)\text{+}2l_2s_2\text{+}d_3(\text{+}c_2(\text{+}1\text{+}d_3d_3\text{-}l_2l_2)$$
$$\text{+}2d_3l_2s_2))\text{+}c_4c_4c_2(\text{+}l_2l_2\text{+}d_2d_2)\text{+}c_2(\text{-}l_2l_2\text{-}d_2d_2))\text{+}l_2l_2(\text{+}d_3d_3\text{+}1)\text{+}d_2d_2\text{+}c_4c_4(\text{-}l_2l_2\text{-}d_2d_2)) \quad (3.102)$$

3.6 CONCLUSÕES DO CAPÍTULO

Esse capítulo trouxe a modelagem cinemática do manipulador elaborado para esse projeto. Foi explicitada toda a teoria envolvida na modelagem do manipulador. Desse modo, foram desenvolvidas as cinemáticas direta e inversa do manipulador, permitindo a determinação da posição do manipulador em função de seus eixos ou de duas juntas, em função da posição final da ferramenta.

Além disso, foi determinado também o modelo de velocidade do manipulador, visto na elaboração do Jacobiano do robô, bem como o Jacobiano inverso, responsável pelo cálculo das velocidades das juntas do manipulador.

Tais ferramentas matemáticas indicam as equações utilizadas para o controle do manipulador, melhor vista no capítulo 5).

Além disso, no capítulo 6), são apontados programas que verificam a exatidão das equações aqui mostradas.

4 PLATAFORMA

4.1 ASPECTOS GERAIS

Esse capítulo aborda as plataformas e as interfaces utilizadas na elaboração desse projeto. A plataforma de *hardware* utilizada é constituída pela *FPGA* utilizada – *Development and Educational Board 3 (DE3)* –, associada aos *drivers* responsáveis pela atuação sobre os motores do manipulador. Para este trabalho, as interfaces foram definidas em função do tipo de comunicação de cada driver/controlador.

Por outro lado, também são apresentadas as ferramentas para configuração do *hardware* e do *software* embarcado na *FPGA*. Essas ferramentas estão integradas ao *software Quartus II*, da Altera, e são, basicamente, ferramentas de desenho e automação eletrônica (EDA – *Electronic Design Automation*), para desenvolvimento de sistemas digitais, programados em *hardware*.

Um diagrama de blocos geral do sistema implementado pode ser visto na Figura 4.1. Nesta, é possível observar que o sistema projetado comunica-se via uma interface RS-232 com os módulos externos. O sistema foi embarcado em uma *FPGA* objetivando o controle da movimentação do robô, disponibilizando uma comunicação com os controladores dos motores do manipulador. Para esta comunicação foi usada uma interface RS-232 para o envio de comandos, assim como comandos específicos do tipo pulso/frequência.

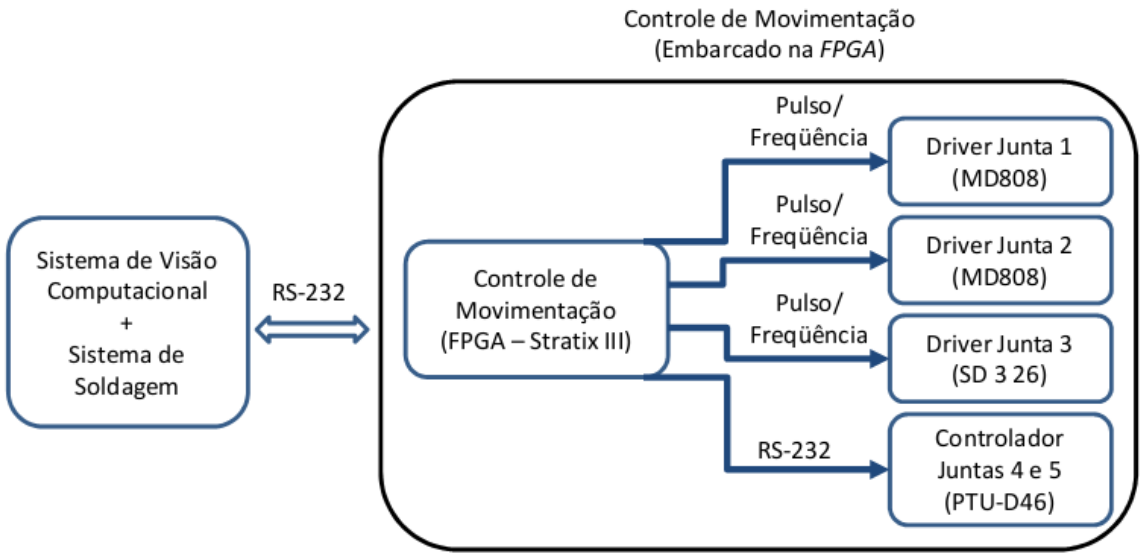


Figura 4.1: Visão geral do *hardware* implementado

4.2 DESENVOLVIMENTO DO *HARDWARE* PARA A *FPGA*

A seguir são explicadas algumas características da placa DE3 e do pacote *Quartus II*. É importante frisar que essa placa foi especificada para ser usada no presente trabalho utilizando o *chip Stratix III* em virtude de ser esse um dos *chip*s da Altera de maior memória interna, para trabalhar com sistemas embarcados de *hardware* e *software*.

Essas características da placa (explicitadas anteriormente) foram consideradas importantes, visando à possibilidade de desenvolver o *software* a ser embarcado, com todas as suas características de complexidade computacional. É importante salientar que o *software* a ser embarcado deve incluir todo o modelo cinemático do robô, leitura de sinais dos sensores, assim como gerar os sinais de controle pertinentes para o funcionamento do manipulador. Além disso, tal *software* deve ter a capacidade de tratar com operações matemáticas complexas no menor tempo possível, visando garantir o desempenho do controle do manipulador.

4.2.1 Características da placa *DE3 – Development and Educational Board 3*

A placa "*DE3*", utilizada nesse trabalho é fabricada pela *Terasic* (Terasic, terasic, 2012). Essa possui um *chip FPGA* desenvolvido pela Altera, de boa capacidade de processamento e de memória, tendo em conta o estado atual da tecnologia. O *chip* usado é o *Stratix III* (EP3SL150F1152C2N), apresentando pouco mais de 142.000 elementos lógicos (LEs). A Tabela 4.1 resume a capacidade deste dispositivo (Altera, Stratix 3 Handbook, 2011).

Tabela 4.1: Capacidade do dispositivo

	Dispositivo	**Total LEs**	**Total de Pinos (I/O)**	**Total de Bits de Memória**	**Total DSP Blocks**	**Total PLL**
Capacidade	EP3SL150F1152 C2N	142.500	744	5.630.976	48	8

Pelas características do FPGA da placa, é possível embarcar um microprocessador com diferentes configurações de recursos de processamento e de memória (nesse caso, foi usado o *NIOS II*, da Altera).

Por outro lado, a placa escolhida conta com 64 Mbits de memória Flash, acessíveis diretamente por outro *chip FPGA* (EPCS64), não volátil (Terasic, DE3 User Manual,

2009). Esse componente é responsável por carregar uma configuração padrão para o *chip Stratix III*. Desta forma, ao fornecer alimentação para a placa, o *chip* de configuração (não volátil), tem a função de carregar uma configuração válida no *chip Stratix III* (volátil).

Esta configuração permanecerá válida enquanto a placa estiver alimentada ou não houver um pedido de *RESET*, por meio de um botão na placa, ou não for carregada uma nova configuração por meio de uma interface *JTAG*.

A placa apresenta também as seguintes características adicionais (Terasic, DE3 User Manual, 2009):

- I/O
    - 8 *leds RGB* de uso geral
    - 4 switches;
    - 1 DIP switch de 8 posições;
    - 2 displays de 7 segmentos;
- Interfaces de expansão
    - 8 conectores do tipo *HSTC*;
    - 2 conectores *Header* de 40 pinos;
- Interfaces de memória
    - Um socket *DDR2 SO-DIMM*;
    - Um socket para cartão SD;
- Clock
    - Oscilador de 50 MHz;
    - 2 conectores *SMA* para entrada de clock externo e para saída de *PLL*;
- Outras Interfaces
    - 1 controlador USB mestre/escravo (1 controlador de 3 saídas USB mestre/dispositivo);
    - 1 *chip* sensor de temperatura;

A Figura 4.2 destaca os principais componentes presentes na DE3.

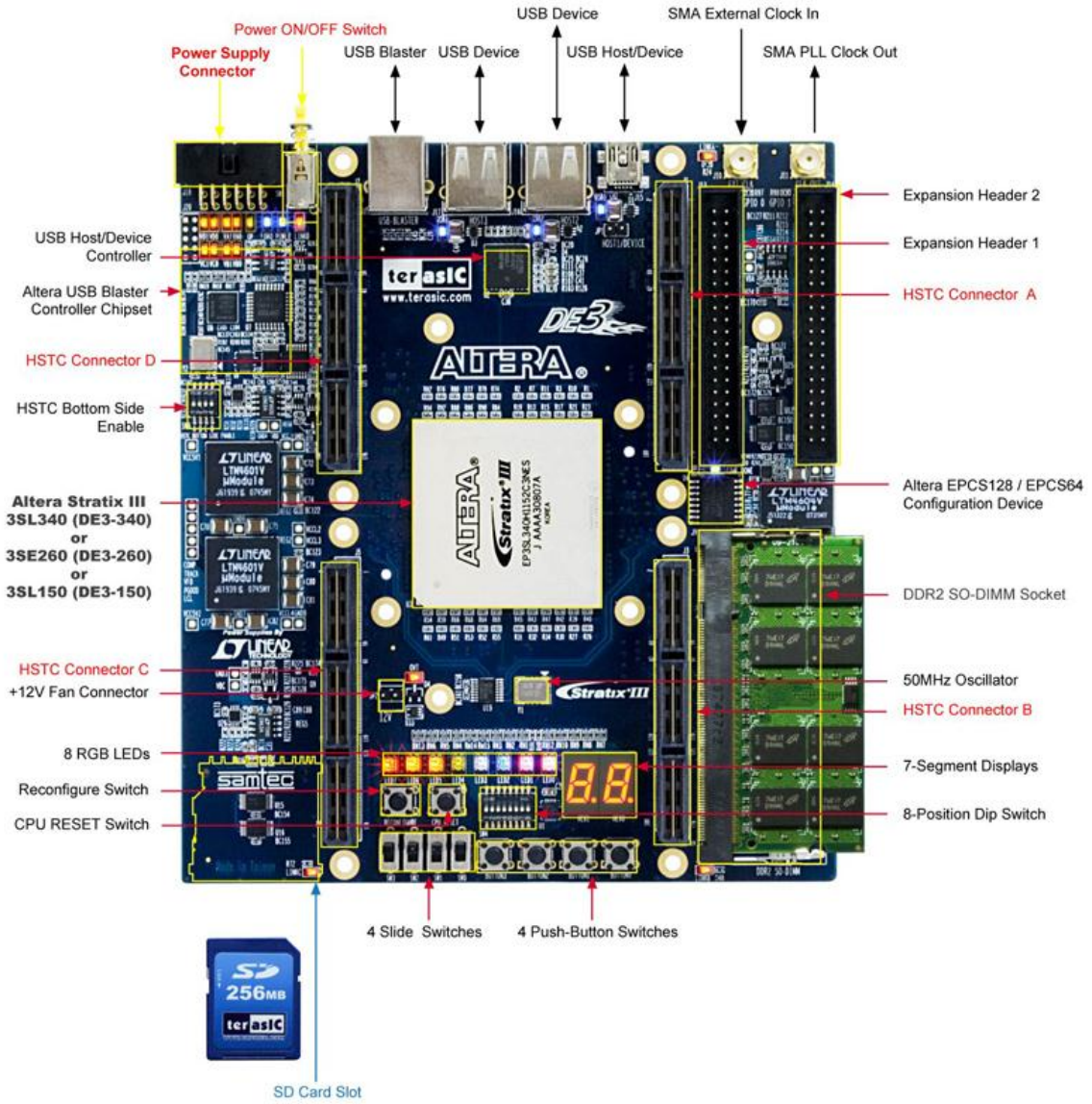


Figura 4.2: Placa DE3

4.2.2 Software para programação em *hardware*

Para realizar construções em *FPGA*s, se faz necessário o uso de ambientes de desenvolvimento dedicados a *dispositivos* reconfiguráveis. Nesse âmbito, encontra-se o *Quartus II*, desenvolvida e comercializada pela Altera. Esta ferramenta possui diversas funcionalidades para auxiliar o desenvolvimento de sistemas complexos dentro de uma *FPGA*.

Dentro do *Quartus II*, é feito todo o desenvolvimento do *hardware*, incluindo todas as fases do projeto. Neste contexto, podem ser feitas simulações para verificação do sistema

projetado. Assim que o sistema possuir uma construção funcional adequada, pode ser feita a programação para o *chip*, para que seja observado o funcionamento do sistema projetado.

Durante a compilação, é gerado um arquivo de configuração, de extensão "*pof*", que pode ser gravado na memória flash da *FPGA*. Também é gerado um arquivo "*sof*", que pode ser carregado diretamente para a *FPGA*, mas que permanecerá apenas enquanto a alimentação for mantida.

O *software* possui todas as facilidades de uma ferramenta EDA, pois a partir de uma linguagem de descrição de *hardware* (Ex. *VHLD* ou *Verilog*) é gerada toda a estrutura de interconexões, tabelas verdade e roteamento dentro do *chip* específico que se está a utilizar. Isso deixa o projetista livre para a criação da funcionalidade do sistema. Essas variações no projeto de sistemas embarcados são ainda facilitadas pela possibilidade de união entre as duas linguagens, caso necessário, ou desejado.

As principais etapas do processo de projeto de *hardware* são descritas a seguir:

- **Descrição do projeto em linguagem de *hardware*;**

Nesta etapa existem várias ferramentas que auxiliam o projeto, tais como uso de blocos lógicos já existentes, disponibilizados como IP, *Intelectual Property*, ou como funções da própria biblioteca do *Quartus*. Tais blocos são descrições de *hardware* que realizam funções de uso comum.

Como exemplo, pode-se citar: Controladores RS-232, controladores para interface JTAG, controladores de memória DDR, controladores de *ethernet* MAC/PHY, ou menos sofisticados como registradores, Flip-Flops, blocos de memória (*on-chip*), etc.

- **Mapeamento das entradas e saídas.**

Aqui são definidos os endereços, em termos de pinos da *FPGA*, para onde serão direcionados os sinais gerados pela lógica interna e em quais pinos serão recebidos os sinais externos. Também existem os chamados *scripts*, que facilitam a tarefa de realizar o mapeamento. Esses scripts são fornecidos, em geral, pelo fabricante da placa, garantindo a minimização de erros de alocação de pinos – erro bastante comum e que pode causar danos à placa.

- **Síntese do projeto.**

O trabalho agora é feito pelo *software*, que deve compilar a linguagem de *hardware*, calcular os recursos utilizados e sua interligação. Assim, estabelece o roteamento dos sinais dentro do *chip* e, por último, calcula se os requisitos de tempo foram atingidos, gerando assim um resultado onde é mostrada a utilização de recursos bem como os parâmetros de tempo obtidos. Qualquer erro, em qualquer dessas etapas, é apontado durante a compilação. As etapas de síntese englobam tarefas de síntese lógica, posicionamento, roteamento, mapeamento tecnológico e, finalmente, a geração do *bitstream* para programação da placa.

- **Simulação**

Pode ser feita uma simulação do funcionamento do *hardware*, dentro do *Quartus II* ou utilizando-se um *software* à parte, o *ModelSim*, também da Altera.

- **Programação do dispositivo**

Na etapa de síntese do projeto foi gerado o arquivo com o *bitstream* para programar o dispositivo. Este é programado através de um cabo JTAG, conectado ao PC através da de uma interface USB.

Uma ferramenta bastante interessante, presente no *Quartus II* é o SOPC Builder (*System On a programmable Chip*), sendo uma aplicação que permite criar sistemas complexos por meio de interligações de blocos menores, utilizando um barramento específico, o *Avalon* (Altera, 2005).

O *SOPC Builder* permite ao projetista adicionar funcionalidades ao sistema sem demandar preocupações com a interconexão dos diversos componentes constituintes. Tal ferramenta facilita o desenvolvimento de todo o sistema de *hardware*, composto por diversos componentes, discutidos em detalhes no capítulo 5.

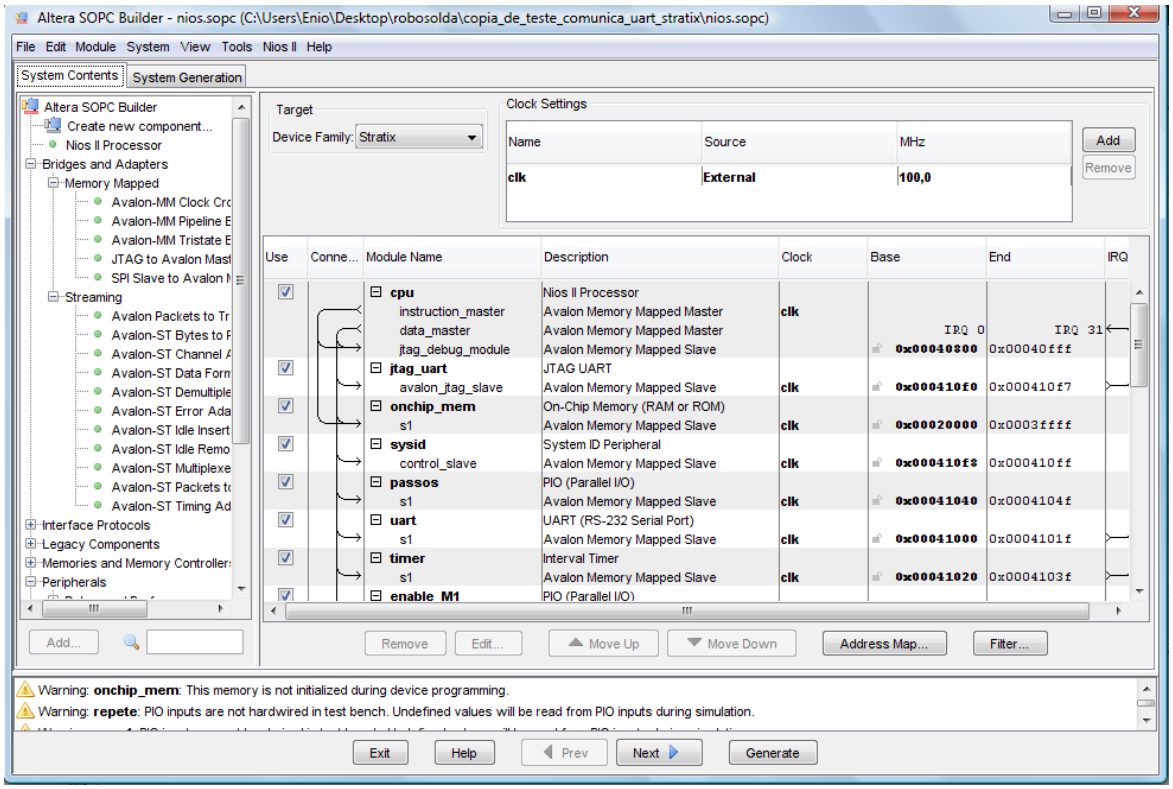


Figura 4.3: SOPC Builder

Com a utilização da ferramenta *SOPC Builder*, mostrada na Figura 4.3, foi gerado um único bloco que representa o microprocessador embarcado na Altera, *NIOS II*, associado aos periféricos desejados. Dessa forma, mostra-se também a flexibilidade desse tipo de arquitetura, onde é possível associar quaisquer periféricos ao microprocessador, sem necessidade de modificação do *hardware* também embarcado na *FPGA*. A Figura 4.4 mostra uma representação do bloco obtido após a geração do sistema dentro do ambiente do *SOPC Builder*.

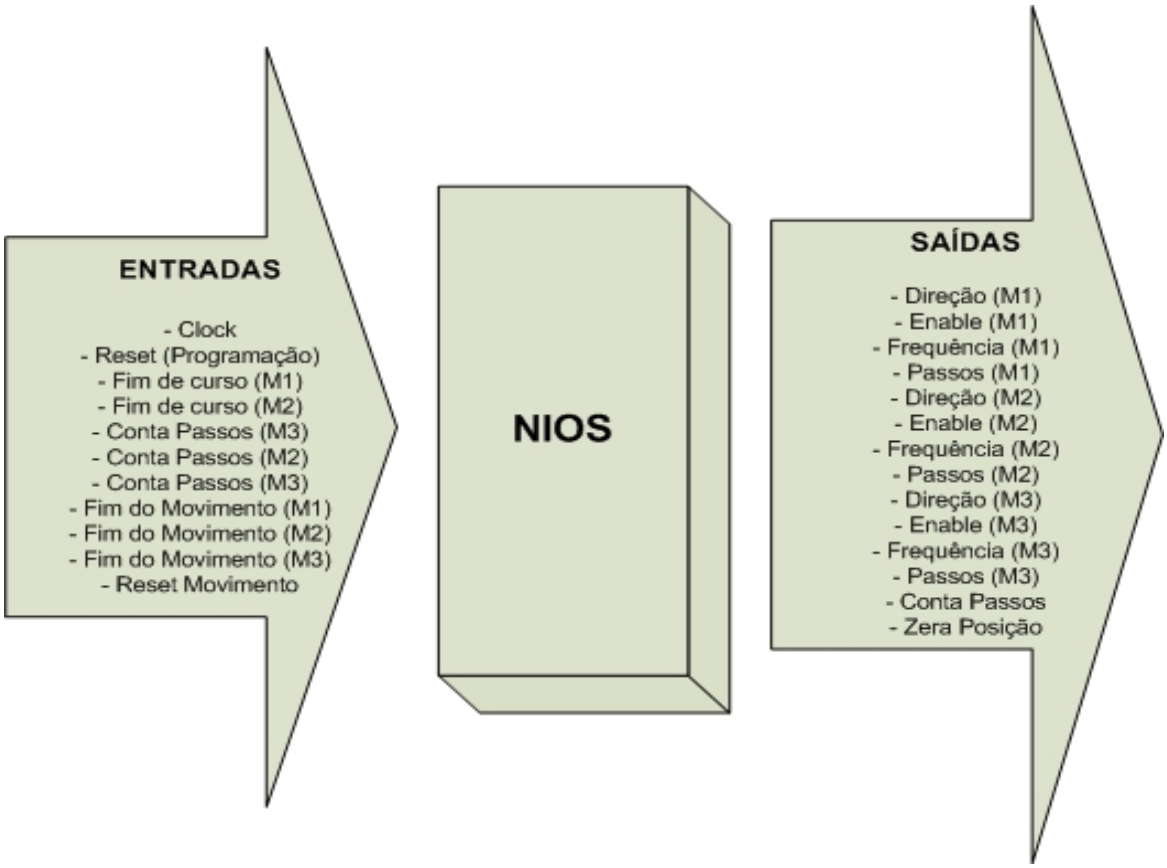


Figura 4.4: Representação em bloco do *NIOS*, associado aos seus periféricos, produzido no *SOPC Builder*

4.3 CRIAÇÃO DE *SOFTWARE* PARA O *NIOS II*

Junto com o pacote de instalação do *NIOS II* encontra-se a *NIOS II IDE*. Essa é uma plataforma baseada no projeto *Eclipse* (Eclipse, 2009), à qual são adicionadas algumas funcionalidades de modo a facilitar o trabalho com o microprocessador *NIOS II*.

Uma das mais notáveis funcionalidades é a integração com o ambiente de desenvolvimento de *hardware*, o *Quartus II*, por meio de um único arquivo, criado pelo *Quartus II* após a síntese do projeto. Esse arquivo é lido pela *NIOS II IDE*, que passa a ter toda a descrição dos periféricos presentes no recém criado e customizado controlador. Dessa forma, é capaz de gerar uma biblioteca especifica para este, de modo a ter acesso a todas as funcionalidades presentes no microprocessador.

Ainda dentro da *IDE* é possível compilar e baixar o código para a placa da *FPGA*, através de um cabo *JTAG/USB*, e executar o código. Também é oferecido um console para que

sejam transmitidos dados de/para o microprocessador, utilizando-se a mesma interface utilizada para a programação.

O projeto a ser implementado no *NIOS* envolve uma série de operações matemáticas. Entre elas, a multiplicação e inversão de matrizes, bem como o calculo de funções trigonométricas que exigem precisão, para determinar a correta movimentação do manipulador. Além da complexidade matemática associada a essas operações, é importante que essas sejam executadas no menor tempo possível, visando garantir a trajetória do manipulador, de modo a satisfazer as restrições de velocidade e tempo de execução da soldagem. Desse modo, a implementação e uso de otimizações no ambiente de desenvolvimento podem ajudar a aumentar o desempenho dos cálculos e manipulações matemáticas.

No contexto desse projeto, foram modificados alguns parâmetros *default* do sistema do *NIOS IDE*. Normalmente, o *NIOS* utiliza a opção *None(-O0)* para o campo *Optimization Levels*. Neste trabalho, foi utilizada a opção *Optimize Most(-O3)*. Essa opção minimiza o tamanho do programa a ser embarcado na *FPGA*, enquanto maximiza a velocidade de execução do projeto, ao otimizar funções. A Figura 4.5 apresenta a tela modificada do *software* do *NIOS IDE*.

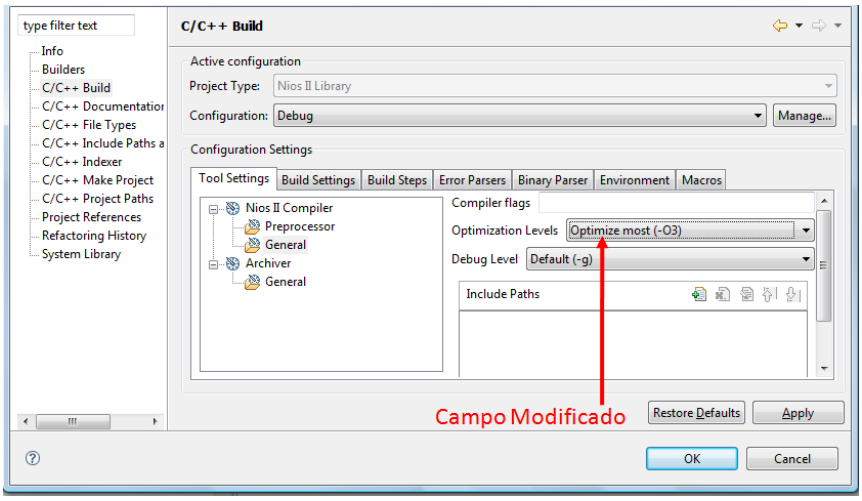


Figura 4.5: Otimização via configuração no *NIOS* IDE

4.4 INTERFACE *FPGA* – DRIVERS DAS JUNTAS 1, 2 e 3

Os *drivers* que controlam os eixos 1, 2 e 3 do manipulador, são comandados através de instruções do tipo *pulso/direção*. Para prover esse tipo de comunicação neste trabalho, dois sistemas foram desenvolvidos.

O primeiro refere-se à questão de acoplamento de tensão, uma vez que os *drivers* dessas juntas seguem o padrão TTL (5V como valor de nível alto), enquanto a *FPGA* segue o padrão CMOS (3.3V como valor de nível alto). E, o segundo refere-se ao *hardware* embarcado na *FPGA*, responsável por fazer o acionamento lógico desse circuito.

4.4.1 Acoplamento de tensão

O *chip* da *FPGA* trabalha numa faixa de tensão de 3,3V, enquanto os *drivers* de acionamento dos motores das juntas 1, 2 e 3 do manipulador são acionados por um sinal de 5V. Dessa forma, para prover comunicação entre as duas plataformas, é necessário desenvolver um acoplamento entre elas.

Nesse projeto, optou-se pelo uso de optoacopladores para fazer essa interface, uma vez que constituem uma tecnologia sólida e de simples aplicação. Além disso, a utilização desse tipo de montagem isola eletricamente os dois lados do circuito, impedindo, por exemplo, que um lado drene mais corrente do que o outro lado pode fornecer. Desse modo, caso um dos *drivers* apresente problema de tensão ou corrente, dificilmente irá danificar a *FPGA* e vice-versa.

Os três *drivers* apresentam estruturas semelhantes nesse ponto, uma vez que ambos têm comandos referentes à *Pulso*, *Direção* e *Enable*. E, como todos trabalham com padrão de tensão TTL, uma única fonte, com corrente suficiente, é necessária para permitir a alimentação dos optoacopladores e prover a fonte para o acionamento dos *drivers*. Dessa forma, foi montado um esquema semelhante ao diagrama lógico apresentado na Figura 4.6. Tal figura apresenta um dos três módulos de comunicação *FPGA-drivers* implementados na *FPGA*, indicando os comandos de saída da *FPGA* e entrada nos *drivers*.

O opto acoplador escolhido para o projeto foi o 6N137, cujo *datasheet* pode ser visto no Anexo B.

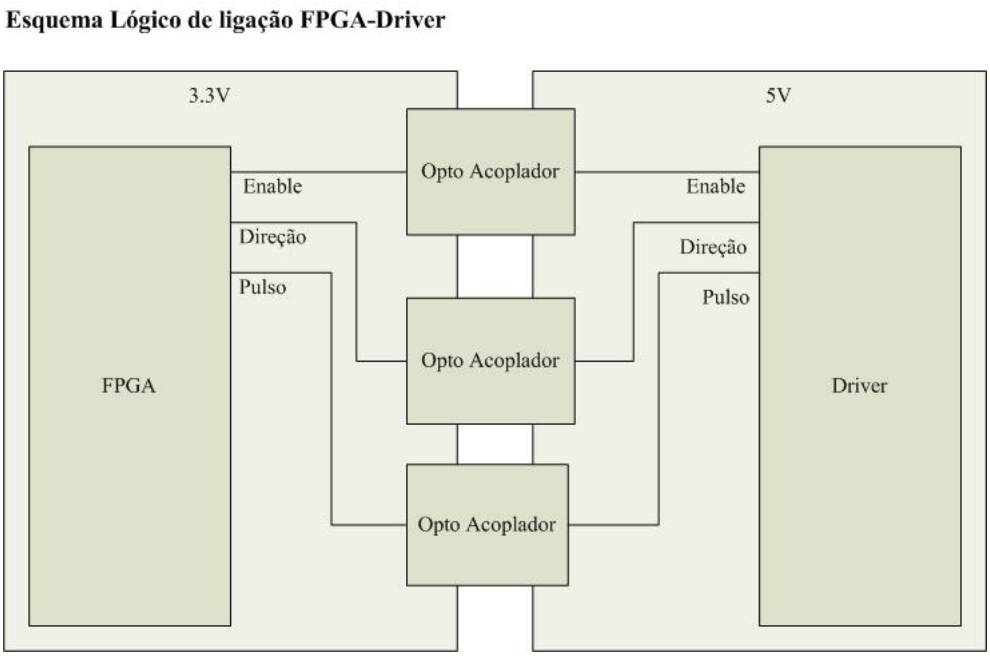


Figura 4.6: Esquema Lógico de ligação *FPGA* - Driver

Esse esquema apresentado é repetido para a conexão de cada um dos *drivers* das juntas 1, 2 e 3 à *FPGA*. Dessa forma, criou-se um padrão modular, e de fácil replicação.

A partir desse projeto, foi desenvolvida, na UnB, uma placa de acoplamento (PA), para fazer a ligação entre a *FPGA* e os *drivers*, de maneira elegante e robusta. Essa placa é apresentada na Figura 4.7. Ela possui uma saída para cada uma das três primeiras juntas do manipulador e uma para os dois últimos atuadores.

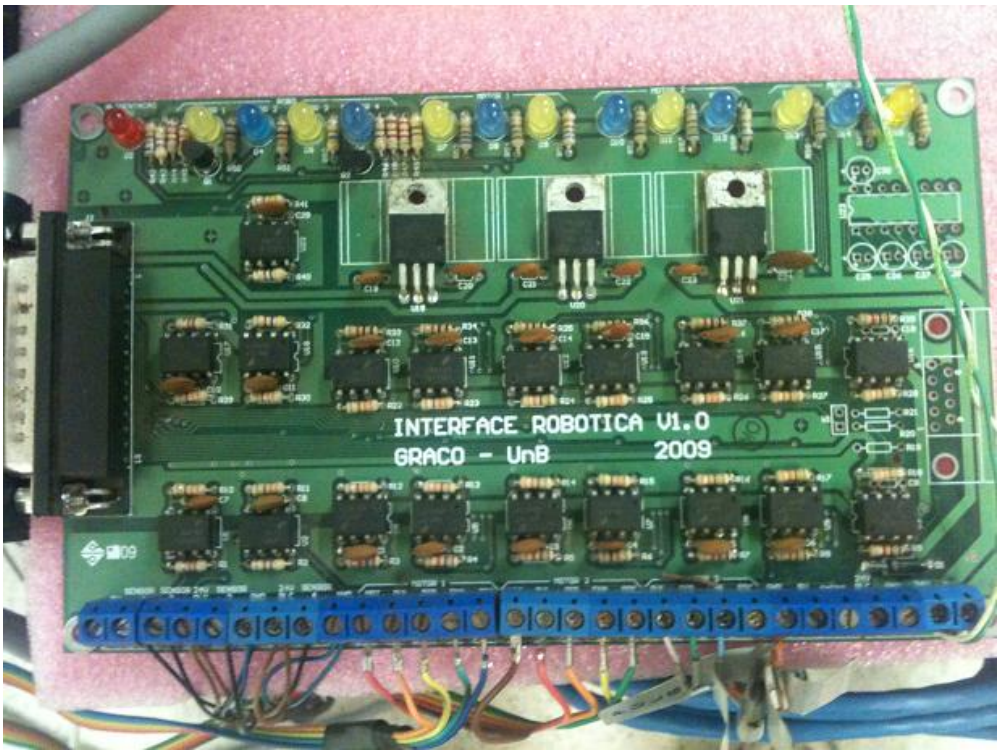


Figura 4.7: Placa Opto Acoplada UnB (PA)

Além disso, a placa tem um conector DB25 (do tipo "fêmea"), para comunicação com a *FPGA*. Desse modo, utilizou-se, para realizar a comunicação entre a *DE3* e a PA, um cabo de duas pontas diferentes. A conexão na PA era um DB 25 (do tipo "macho"), enquanto a outra ponta do cabo era um *header fêmea* de 40 Pinos a ser ligado na *DE3*.

4.4.2 *Hardware* embarcado na *FPGA*: comunicação *FPGA*-DRIVERS

O sistema apresentado nesse trabalho refere-se à união entre um microprocessador embarcado numa *FPGA* juntamente com outras estruturas de *hardware* também embarcadas nessa *FPGA*, visando controlar um manipulador robótico. Desse modo, algumas funções foram implementadas diretamente no *hardware*, visando aumentar a eficiência do processo de controle e movimentação.

Essas funções tratam, principalmente, das ações sobre os atuadores do manipulador. Desse modo, o *NIOS* funciona como um centralizador de comandos, que delega tarefas de acionamento e deixa que esses "blocos" de *hardware* garantam a execução dos comandos (blocos esses apresentados nas Figura 4.9 e Figura **4.8**).

Tais blocos atuam sobre os *drivers* das juntas 1, 2 e 3 do manipulador, que são controlados a partir de três comandos principais: *Pulso*, *Direção* e *Enable*. Dada essa padronização, foi desenvolvido um sistema digital na *FPGA*, responsável por acionar esses *drivers*, com a capacidade de controlar a freqüência de acionamento do motor, bem como de controlar o tempo e a direção desse movimento. Essas funções são utilizadas em dois momentos distintos:

(a) Posicionamento inicial das juntas do manipulador;

(b) Controle "step-by-step" das iterações do algoritmo de calculo de trajetória;

Em ambos os casos, cabe ao *software* embarcado no *NIOS* determinar o inicio do movimento, mas cabe ao *hardware*, fora do *NIOS*, garantir essa movimentação e indicar o seu termino.

Tendo em conta que ambas as funções tratam de movimentação do manipulador e atuação sobre os *drivers*, poder-se-ia argumentar que com apenas um bloco de *hardware* as duas funções poderiam ser executadas. Contudo, algumas especificidades de cada uma das funções mostram que cada uma delas é otimizada em relação à outra, para a atividade a ser realizada.

4.4.2.1 Posicionamento inicial das juntas do Manipulador

Quando o manipulador é acionado inicialmente, supõe-se que a posição de cada uma de suas juntas não é conhecida, uma vez que não se pode afirmar nada sobre a posição em que o manipulador foi desligado. E, considerando que toda a movimentação do manipulador é realizada em malha aberta, saber sua posição inicial é necessário para garantir uma movimentação correta. Desse modo, ao ser ligado, o manipulador deve deslocar cada uma de suas juntas para uma posição conhecida, que servirá de referência para os movimentos posteriores.

A Figura 4.8 apresenta o bloco base utilizado para o acionamento das juntas 1, 2 e 3 no processo de movimentação para a posição zero.

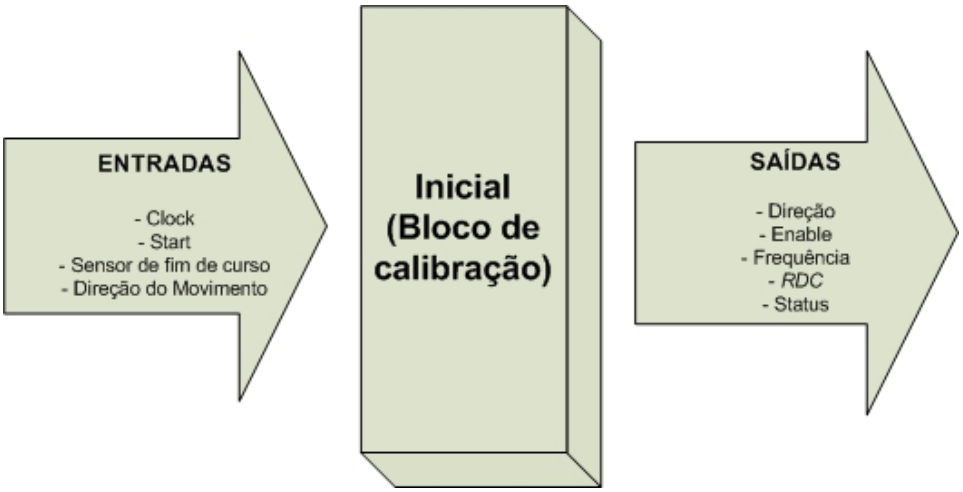


Figura 4.8: Bloco de *Hardware* utilizado para o posicionamento inicial dos eixos do manipulador robótico

A seguir, é feita a explicitação de cada um dos I/O's do bloco apontado na Figura 4.8.

a) *Clock (Entrada)*: essa é a entrada do *clock* do Bloco. Como esse sistema funciona independentemente do *NIOS*, seu valor é diferente. Nesse caso, foi utilizada a freqência de 50 MHz;

b) *Start (Entrada)*: recebe um comando do *NIOS* para começar a movimentação para a posição inicial;

c) *Sensor (Entrada)*: recebe informação do sensor de fim de curso de cada eixo, visando indicar o fim da movimentação;

d) *Dir_in (Entrada)*: indica a direção do movimento, onde 1 é o sentido horário de rotação do motor;

e) *Dir_out (Saída)*: indica a direção do movimento para o *driver*;

f) *Enable (Saída)*: indica se a movimentação do motor está ou não habilitada;

g) *Pulse (Saída)*: freqüência do pulso de acionamento do *driver*;

h) *Rdc (Saída)*: indica se o *driver* deve ser acionado com 100% ou 50% da corrente disponível;

i) *Status (Saída)*: Indica para o *NIOS* se a movimentação da junta foi finalizada, ao acionar o sensor de fim de curso;

O processo de posicionamento inicial do manipulador começa logo após o *NIOS* ser inicializado. Esse envia um comando através da saída *out_port_from_the_zera_pos* (a ser melhor comentada no capítulo 5), conectada a porta *start* do bloco ligado a cada uma das 3 primeiras juntas do manipulador.

Entretanto, neste trabalho, definiu-se uma participação do usuário nesse processo de posicionamento inicial. Visando a segurança do manipulador e prevendo a possibilidade de alguém estar próximo ao manipulador, essa movimentação só começa se o *Switch0* da DE3 estiver na posição 1. Desse modo, evitam-se acidentes.

Além disso, em função de algumas peculiaridades associadas a construção do manipulador, as juntas 1 e 2, possuem apenas um sensor de fim de curso. Foi definido, então, que o usuário também é responsável por indicar o lado para o qual esses eixos devem movimentar-se até chegar na posição inicial, através dos Switches 1 e 2 da DE3.

4.4.2.2 Controle das iterações do algoritmo de calculo de trajetória

A Figura 4.9 indica o bloco de acionamento dos *drivers* dos motores, chamado de *freq_div_new*, embarcado na *FPGA*. Esse bloco apresenta algumas entradas e saídas, explicadas abaixo:

a) *Clock_100MHz (Entrada)*: essa é a entrada do *clock* da *FPGA*. Ele serve como referência para as operações do sistema, sendo o mesmo tanto para o bloco onde está inserido o *NIOS* quanto para o *freq_div_new*;

b) *Div_freq (Entrada)*: é um vetor responsável por apontar freqüência necessária ao movimento;

c) *Steps (Entrada)*: é um vetor responsável por indicar o número de passos a serem executados pelo motor durante o movimento;

d) *Dir_in (Entrada)*: indica a direção do movimento, onde 1 é o sentido horário de rotação do motor;

e) *Enable_d (Entrada)*: indica se o bloco deve ou não iniciar/continuar a movimentação;

f) *Passo*: controla o tipo de movimentação. Caso assuma o valor *1*, indica que o controle do movimento será através do número de pulsos dados. Já se assumir o valor *0*, indica que o movimento continuará enquanto o valor do *enable_d* seja *1*.

g) *Dir_out (Saída)*: indica a direção do movimento para o *driver*;

h) *Enable_m (Saída)*: indica se a movimentação do motor está ou não habilitada;

i) *Pulse_Out (Saída)*: freqüência do pulso de acionamento do driver;

j) *Fim (Saída)*: indica o fim do comando de movimentação;

k) *Steps_atual (saída)*: vetor responsável por dar um informar ao *NIOS* a quantidade de passos dado até o momento;

Como se pode observar através da explicação do comando *passo*, existem dois tipos de movimentação providos pelo *aciona_mot*. O primeiro modo demanda que o *NIOS* determine o número de pulsos a serem enviados ao *driver* do motor.

Depois de iniciado o comando de movimentação, o *freq_div_new* conta quantos pulsos já enviou ao *driver*. Como são utilizados motores de passo, o fim da contagem desses pulsos indica o fim da movimentação desejada. Esse é o modo utilizado para a maior parte das movimentações do manipulador, onde os sinais são todos internos à *FPGA*. Desse modo é provido o controle em malha aberta.

Nesse ponto é importante ressaltar que a garantia da correta movimentação do manipulador só será alcançada se os motores não perderem passos. Ou seja, seu dimensionamento deve estar correto, para garantir a qualidade da movimentação em malha aberta.

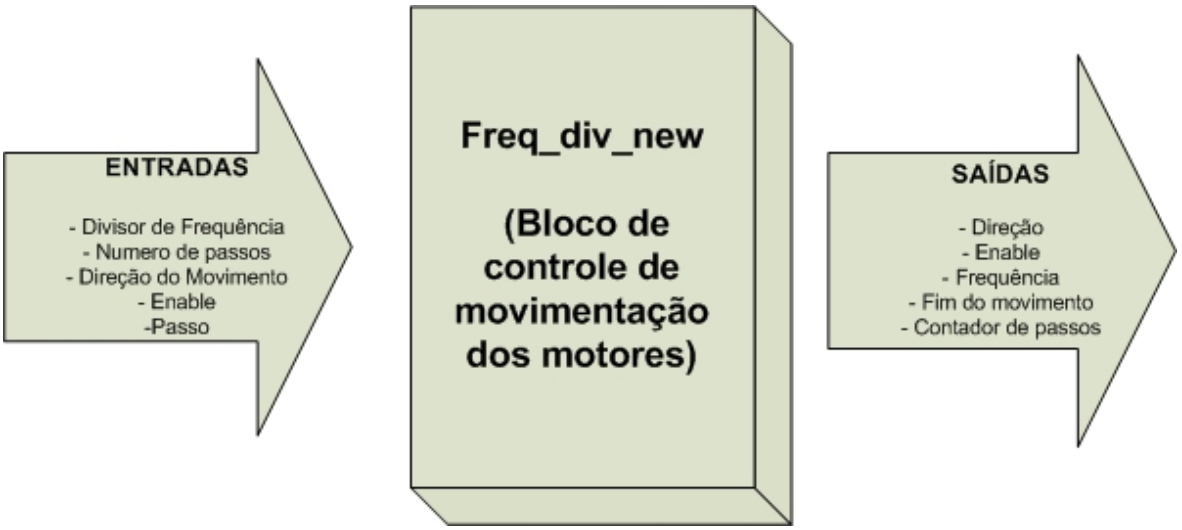


Figura 4.9: Bloco responsável pelo controle das iterações do algoritmo de calculo de trajetória

Novamente, buscou-se simplificar o projeto do *hardware*. E, do mesmo modo que o acoplamento de tensão, os blocos *freq_div_new* foi replicado para os três *drivers*.

4.5 INTERFACE *FPGA* – CONTROLADOR DAS JUNTAS 4 e 5 (PAN-TILT)

As juntas 4 e 5, referentes ao punho do manipulador são controladas juntamente, através de comunicação RS232. Para se implementar tal comunicação, foi utilizado um periférico acoplado ao *NIOS II*. Esse periférico é provido por um *MegaCore Uart Rs-232* e é um recurso oferecido pelo *SOPC Builder*, visando facilitar o trabalho de desenvolvimento do *software* (Altera, 2009).

A utilização desse recurso diminui o tempo de criação do projeto de *hardware*, além de facilitar o trabalho do desenvolvimento do *software* a ser embarcado no *NIOS II*. Isso porque as bibliotecas e primitivas de comunicação com esse periférico já estão implementadas. Outra vantagem associada ao uso do *Core* é a utilização do barramento *Avalon*. Como os dados de controle do *Pan-Tilt* são gerados pelo *NIOS*, como será visto no capítulo 5 e podem ser enviados diretamente para a porta serial, sem a ocorrência de maiores atrasos ou erros de comunicação.

A Figura 4.10 apresenta a interface de configuração da *UART*.

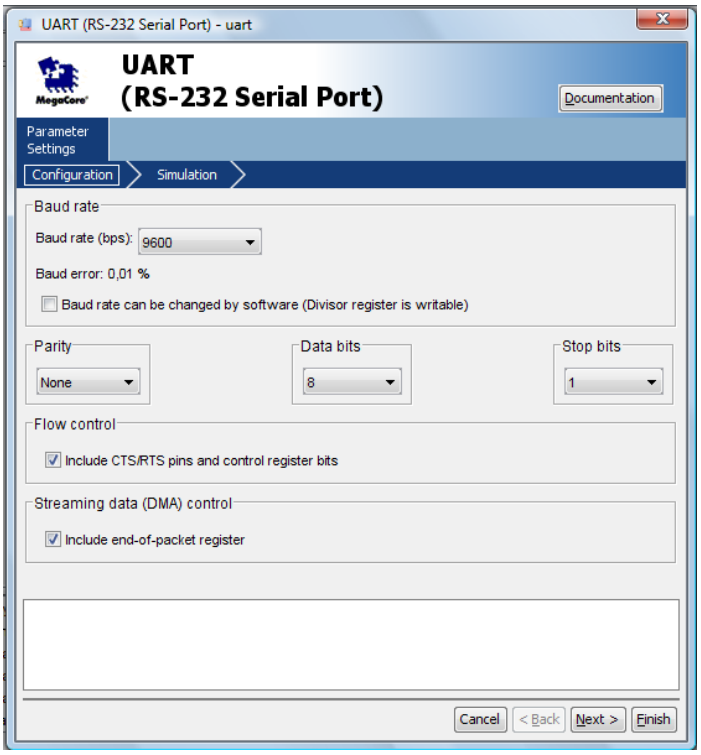


Figura 4.10: Interface de configuração da UART RS-232

Contudo, a placa utilizada nesse projeto não apresenta interface serial RS232 própria. Foi então necessária a utilização de um circuito integrado externo à *FPGA* para implementar essa comunicação.

Optou-se pela utilização de pinos da *FPGA*, associados ao uso de um *Max-232* (ver *datasheet* no anexo B). Novamente, a diferença de padrões de tensão foi um fator determinante para o uso desse dispositivo. A Figura 4.11 mostra a representação lógica desse esquema de comunicação.

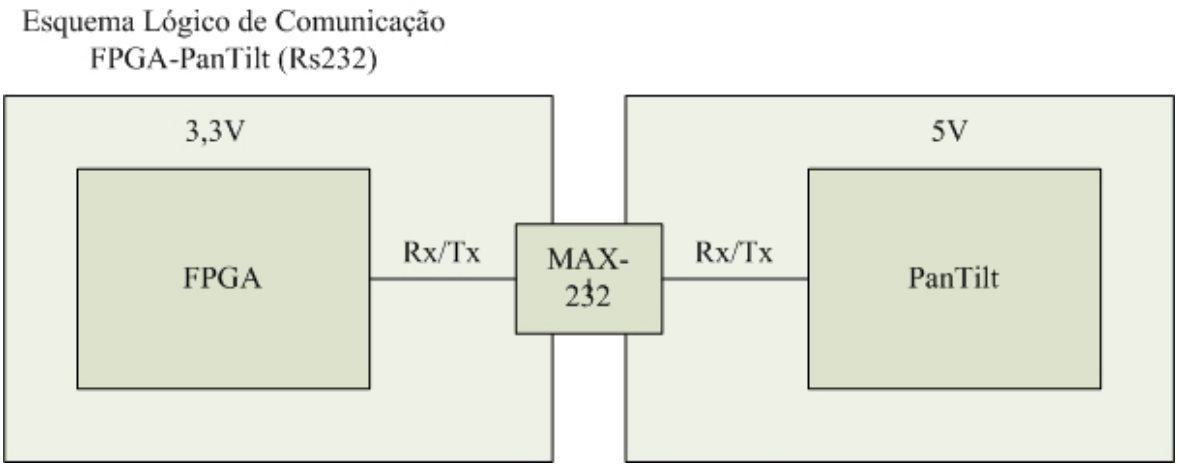


Figura 4.11: Diagrama Lógico de conexão PanTilt-*FPGA*

4.6 SISTEMA COMPLETO

Unindo os diversos blocos descritos ao logo desse capitulo, foi montada a plataforma de controle do manipulador desenvolvido nesse projeto. A integração entre esses elementos pode ser vista na Figura 4.12, de maneira resumida.

Nela, podem ser vistos os blocos de entrada de dados para o microprocessador *NIOS II*, compostos por:

- Entrada de frequência (*PLL*);
- Status dos sensores de fim de curso – responsáveis por indicar o fim da movimentação para a posição zero do manipulador (retorno dos blocos *inicial*);
- *Feed backs* dos módulos de comando – responsáveis por indicar o fim das movimentações com trajetórias definidas (retorno do bloco *frq_div_new*);

Também podem ser observadas as saídas do *NIOS*, responsáveis por acionar a movimentação dos motores do manipulador:

- *Freq_div_new* (m1, m2 e m3);
- *Inicial* (m1, m2 e m3);
- Comandos para controle dos motores 4 e 5 do manipulador;

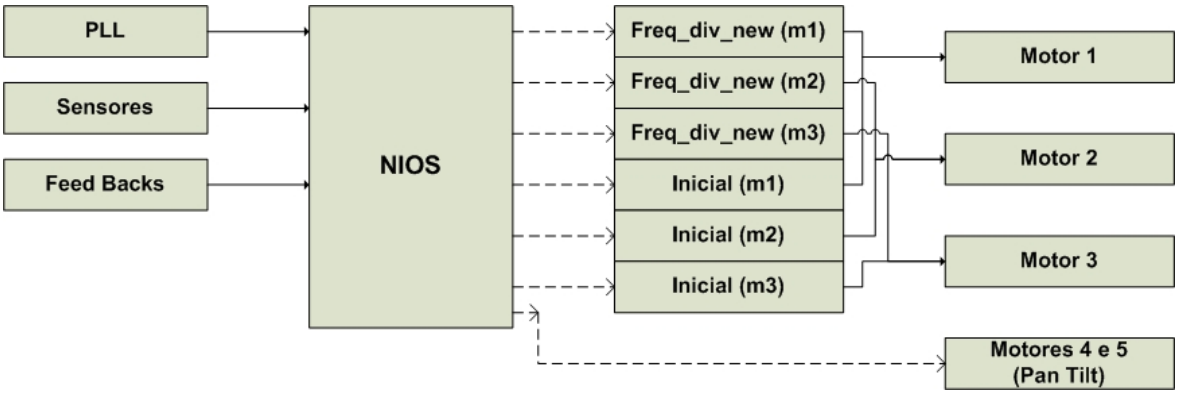


Figura 4.12: Sistema completo embarcado na *FPGA*

Os componentes da arquitetura que compõem o bloco do processador serão explicados no capítulo 5.

4.7 INTERFACE *FPGA*-MÓDULOS EXTERNOS

Esse trabalho faz parte de um projeto maior, sendo que, no projeto, existem dois módulos que devem comunicar-se com a solução aqui proposta. Esses dois módulos são referentes à visão computacional e ao gerenciamento de soldagem. Esses módulos também demandam uma comunicação com a *FPGA*.

Na primeira etapa do projeto, esses módulos estão localizados num PC, externos à *FPGA*. Dessa forma, é interessante a utilização de um padrão de comunicação simples e funcional, de rápida implementação. Desse modo, novamente, planejou-se a utilização de uma comunicação utilizando serial do tipo RS-232, utilizando-se, novamente, do *Core* fornecido pela Altera.

Entretanto, como esses blocos ainda não foram concluídos, optou-se por não implementar essas funções nesse projeto. Desse modo, as entradas de dados para testes e simulações foram inseridos no código do projeto.

4.8 MULTI-PROCESSAMENTO

Uma alternativa a plataforma implementada nesse capitulo é baseada na utilização de mais de um núcleo de processamento, através do uso de mais de um microprocessador. A ideia de implementar mais de um núcleo é bastante válida na busca pelo aumento de desempenho dos cálculos matemáticos e operações matemáticas de alta complexidade.

A plataforma *NIOS* é bastante robusta quanto a isso, permitindo a fácil integração entre dois ou mais *NIOS*, dentro da *FPGA*, através do barramento *Avalon* – a ser melhor explicado no capitulo 5.

Entretanto, o algoritmo de calculo da cinemática inversa, desenvolvido no capitulo 3 desse trabalho, mostra uma série de cálculos sendo realizados de forma serial, a serem implementados no *NIOS*, para determinação e controle da movimentação do manipulador. Como pôde ser visto, o calculo de θ_1, por exemplo, depende do calculo de θ_2 e de D_3, dificultando, assim, o calculo em paralelo dessas variáveis.

Desse modo, optou-se por manter a utilização de apenas um núcleo de processamento, facilitando a implementação do algoritmo escolhido.

4.9 CONCLUSÕES DO CAPÍTULO

Esse capítulo mostrou as interfaces entre os diversos módulos de *hardware* envolvidos nesse projeto. Mostrou também algumas das razões a respeito das escolhas dos padrões de comunicação entre eles. Além disso, mostrou as plataformas de *software* utilizadas para o desenvolvimento do *hardware* utilizado no processo.

5 ARQUITETURA

5.1 ASPECTOS GERAIS

Esse capítulo aborda a arquitetura de sistema proposta para esse trabalho. Tal arquitetura é composta pelo projeto de *hardware* associado ao micro-processador *NIOS II*, instanciado na *FPGA*, e pelo *software* embarcado nele, responsável pelo calculo das equações de controle do manipulador.

O grande objetivo desse trabalho é prover as condições para que o robô execute sua movimentação seguindo uma trajetória desejada. Dentro dessa perspectiva, foram explicitadas as condições físicas utilizadas para isso, como os atuadores, motores, interfaces e *hardware*. Contudo, o controle desse conjunto fica a cargo do *hardware* embarcado na *FPGA*. E, esse, tem como módulo central o *chip* instanciado nela, uma vez que ele é que possui as diretivas de comandos dos *drivers*, interpreta as informações de entrada, calcula as novas posições e gera as saídas para os motores.

5.2 DESCRIÇÃO DA ARQUITETURA DO *HARDWARE*

No capitulo 4, na Figura 4.1, foram explicitadas as estruturas de *hardware* que compõem este trabalho. Nesta figura, foi apresentado o bloco de controle de movimentação, que está inserido na *FPGA* e é composto pela associação entre o bloco de periféricos integrados ao micro-processador *NIOS II* (Figura 4.4) – e os blocos de acionamento dos motores (Figura 4.8 e Figura 4.9). A união desses elementos constitui a Arquitetura do *hardware* implementado nesse trabalho.

A Figura 5.1 mostra com mais detalhes a arquitetura do *hardware* proposto, exibindo as conexões entre os blocos *freq_div_new* e *inicio*, descritos no capítulo anterior, com o *NIOS* e seus periféricos.

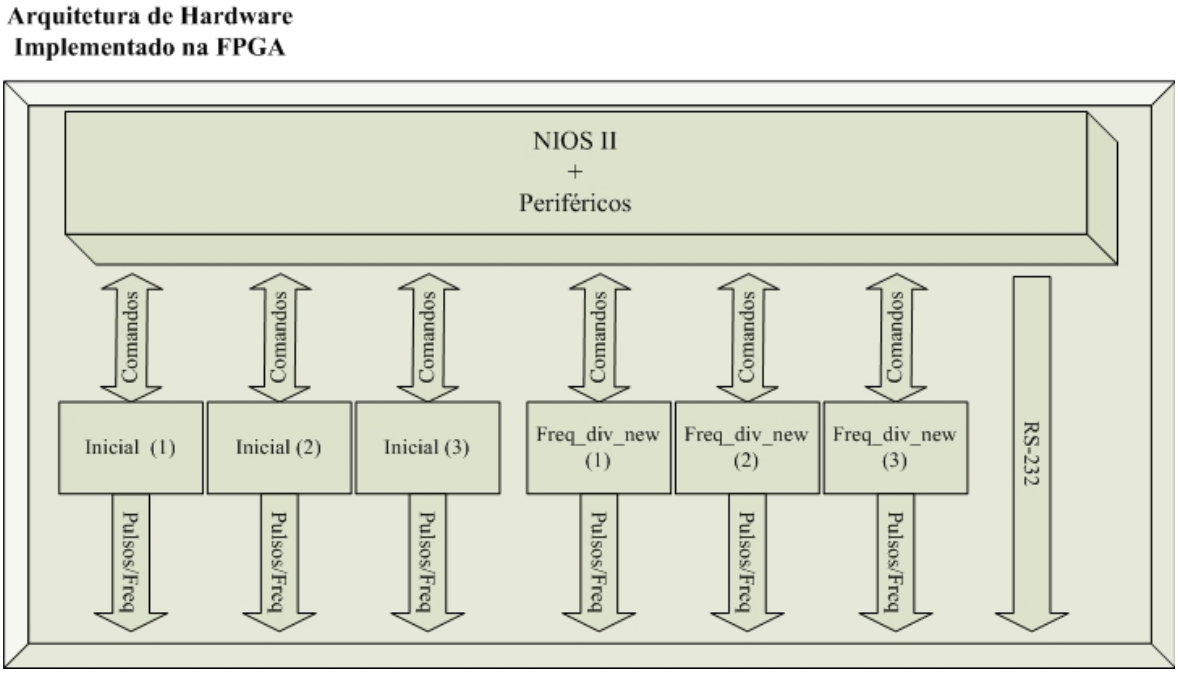


Figura 5.1: Arquitetura de *Hardware* implementado na *FPGA*

A arquitetura do bloco que une o *NIOS II* aos periféricos deve ser explicada em duas partes:

a) Descrição dos componentes e integração entre eles;

b) Descrição do *software* embarcado e suas funcionalidades.

5.3 COMPONTES DO *HARDWARE* ASSOCIADO AO *NIOS II*

Para o projeto do *hardware* embarcado, utilizou-se o *SOPC builder* para instanciar um *chip* embarcado na *FPGA*, que permitiu embarcar a lógica de controle de movimentação do manipulador. O projeto contém os seguintes componentes, os quais são apresentados na Figura 5.2:

a) *Processador NIOS II*: é o gerente do sistema, sendo responsável por tratar as equações de cinemática – direta, inversa e do Jacobiano;

b) *Bloco de memória de 128 Kb*: Armazena os dados do programa, bem como as respostas das equações e armazena os dados inseridos pelos blocos externos;

c) *Driver JTAG*: responsável pela comunicação entre a *FPGA* e o console, para *debug* e configuração;

d) *Timer*: Provê as condições de temporização das ações do JTAG;

e) *SysId*: cria um número identificador para a *FPGA*;

f) *I/O*: Conjunto de entradas e saídas do *chip*, responsáveis pela comunicação RS-232 e pela comunicação com os blocos *Aciona_mot*;

g) *UART (RS-232 Serial Port)*: responsável pela comunicação entre o *chip* e os eixos 4 e 5 do manipulador e com os módulos externos ao sistema de controle de movimentação.

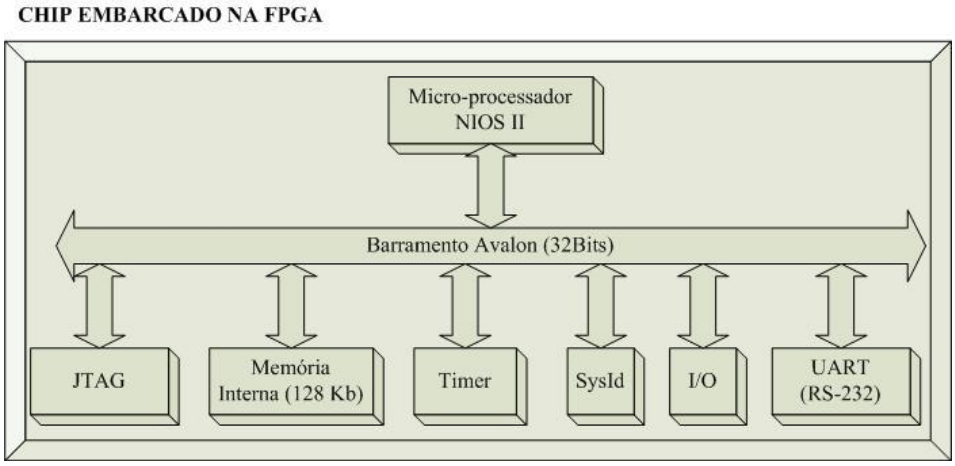


Figura 5.2: Visão simplificada do *chip* embarcado na *FPGA*, com seus componentes

Todos os componentes inseridos no *chip* comunicam-se através do barramento *Avalon*. Como citado anteriormente, o mesmo permite uma comunicação quase transparente entre os periféricos, facilitando assim o trabalho de programação do usuário/desenvolvedor.

A vantagem do *Avalon* consiste em que, depois de instanciados os periféricos, a conexão de comunicação é automática, podendo ser modificada em caso de demandas especificas. Por exemplo, podem-se instanciar dois dispositivos de memória e determinar quais periféricos terão acesso a cada memória.

Contudo, mesmo que essas conexões sejam modificadas, dentro da programação do microprocessador, basta acessar as bibliotecas referentes a cada periférico e enviar e receber dados deles, sem necessidade de conhecimentos específicos a respeito do protocolo utilizado no barramento de comunicação.

Poder-se-ia utilizar, nesse projeto, a memória externa ao *chip* da *FPGA*, presente na placa *DE3*. Entretanto, essa opção não foi utilizada em função do tempo de acesso a ela. Como a latência do acesso á memória externa é aproximadamente 8 vezes maior que a de acesso a

memória do *chip* – medida experimentalmente –, e a necessidade de acesso é constante, optou-se pelo uso da memória interna à *FPGA*.

5.4 CONFIGURANDO OS PERIFÉRICOS DO *CHIP*

Todos os componentes comentados no item 5.3 devem ser configurados, através do *SOPC Builder*, em função das especificidades do projeto. Os mais importantes aspectos dessas configurações serão descritos a seguir:

- *Processador NIOS II*: dentro do *SOPC builder*, escolhe-se o componente *NIOS II Processor*. Esse componente permite instanciar o processador embarcado na *FPGA* onde será possível manter um controle de alto nível, com programação em linguagem C/C++. O processador apresenta três barramentos, depois de instanciado: *instruction_master*, *data_master* e *jtag_debug_module*. O primeiro e o segundo são responsáveis, respectivamente, pelos conjuntos de instruções e de dados do processador. Já o terceiro refere-se à interface de debug do sistema, associada ao JTAG, que também permite a comunicação com o console do PC.

  A tela de configuração do *NIOS* é apresentada na Figura 5.3. Dentro dessa configuração, é possível escolher e configurar o processador de acordo com as necessidades de projeto. Inicialmente, é possível configurá-lo em relação o *core* utilizado. Dentre esses, são três as variações, que alternam entre capacidades, funções, unidades lógicas utilizadas e utilização de memória.

  Além disso, também é possível configurar o *Cache* de instruções e de dados, sinais de solicitação de reset, o tipo de debug a ser utilizado e até mesmo customizar instruções a serem utilizadas. Contudo, as escolhas do *core* limitam essas opções.

  Dentro desse projeto, o processador *NIOS* é o único elemento *mestre* do barramento Avalon. Isso quer dizer que é ele quem inicia e solicita operações ao barramento. Todos os outros periféricos são considerados *escravos* do barramento.

- *Memória do Chip*: O componente *On Chip Memory (Ram or Rom)* é responsável pela configuração desse periférico. Nele é possível determinar o tipo de blocos de memória que serão instanciados, sendo do tipo MRAM, M4K ou M512. Como a memória do *Chip* é fixa, a escolha de um ou de outro tipo de memória indica a

quantidade de blocos que podem ser utilizados. Importante frisar que essa quantidade de memória e os tipos de blocos dependem do tipo de *FPGA* escolhido. Para esse projeto, foram utilizados 256 Kbytes, com blocos de 32 bits, do tipo MRAM.

Além disso, é possível configurar esse bloco como RAM (de escrita possível) ou ROM (somente leitura). Nesse projeto, foi escolhido o tipo RAM.

- *Driver JTAG*: como comentado anteriormente, esse periférico permite a comunicação serial entre a *FPGA* e o console do *NIOS II* IDE. Desse modo é possível ainda realizar o debug do projeto e de sua programação;

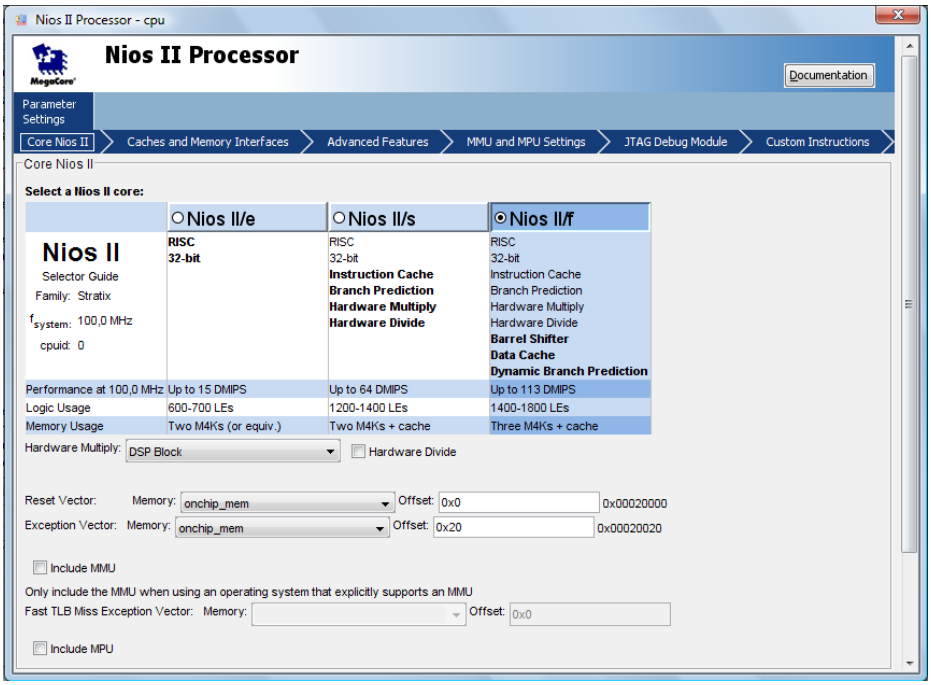


Figura 5.3: Tela de configuração do *NIOS II*

- *SysId*: periférico de configuração de identificação do projeto. Essa é uma ferramenta de segurança que impede que seja feito um download de um projeto desenvolvido para um determinado *chip* em outro *chip* que não seja compatível com o projeto;

- *Timer*: periférico recomendado pela fabricante, uma vez que é responsável por garantir o funcionamento correto do driver do JTAG, temporizando suas ações. Além disso, esse periférico permite a utilização de funções de tempo dentro do projeto do processador;

- *UART (RS-232)*: Outro periférico de comunicação. Esse componente foi comentado na seção 4.5 do capitulo 4. Esse periférico permite a comunicação serial RS-232. Para esse projeto, foram utilizadas duas instâncias desse periférico.

  A primeira é utilizada para a comunicação com os eixos 4 e 5 do manipulador, sendo configurada em velocidade de 9600 bps. Já a segunda é responsável pela comunicação com os módulos externos ao processo de controle de movimentação. Para tanto, está configurado em 115600 bps.

  Esse periférico tem saída própria do *chip*, não demandando I/Os extras para realizar comunicação com exterior do *chip*;

- *PIO (Parallel I/O)*: esse periférico instancia os elementos de entrada/saída do sistema. É responsável por levar para fora do *chip* os dados responsáveis por acionar os blocos *div_freq_new* e *inicia*. As instâncias de I/O utilizadas são descritas a seguir:

  - *Passos (saída)*: indica o tipo de movimentação, comentada no capitulo 4;
  - *Enable M1, M2 e M3 (saída)*: habilitam a movimentação dos motores das juntas 1, 2 e 3;
  - *Dir M1, M2 e M3 (saída)*: indicam a direção de movimentação das juntas 1, 2 e 3;
  - *Fim M1, M2 e M3 (entrada)*: indicam o fim do movimento dos eixos 1, 2 e 3, durante o calculo de trajetória;
  - *Freq M1, M2 e M3 (saída)*: indicam a freqüência desejada para a movimentação das juntas 1, 2 e 3;
  - *Passos_a M1, M2 e M3 (entrada)*: indicam o numero de passos dados até o momento pelos motores das juntas 1, 2 e 3;
  - *Steps M1, M2 e M3 (saída)*: indicam o número de passos a serem dados pelos motores das juntas 1, 2 e 3;

- *Repete (entrada)*: indicam para o processador que o movimento deve ser reiniciado;

- *Pos M1, M2 e M3 (entrada)*: indicam para o processador que foi acionado o fim de curso do eixo;

- *Zera_pos*: Indica que o movimento de zerar a posição do manipulador deve ser iniciado;

5.5 COMPONENTES DO *SOFTWARE* EMBARCADO NO *NIOS II*

Uma vez descrita à arquitetura do *hardware* do *chip* onde está instanciado o processador do projeto, discute-se a arquitetura do *software* embarcado no mesmo. É nesse *software* onde estão presentes os comandos de acionamento dos motores, bem como os parâmetros de cinemática direta, inversa e do Jacobiano. É nesse *software* também que estão inseridos os módulos de comunicação com os *drivers* dos motores das juntas 4 e 5, através da interface RS-232.

O sistema de controle foi produzido em linguagem C, dentro do ambiente de desenvolvimento *NIOS II* IDE. Inicialmente, foi usada a linguagem C++, com orientação a objetos. Contudo, dentro de um ambiente de microprocessador, esse uso não é o mais recomendado, uma vez que utiliza mais memória que os métodos estruturados. Portanto, optou-se pela linguagem estruturada em C, mais leve e compacta, apesar de menos reutilizável. Apesar disso, a utilização da linguagem C ainda mantém o projeto bastante simples, uma vez que é uma linguagem bastante utilizada, de conhecimento bastante difundido.

A Figura 5.4 apresenta a arquitetura implementada no *software*. São apresentados os blocos principais desse *software*, que serão descritos com maior propriedade nas sessões seguintes. Além disso, são apresentadas 3 divisões entre as funções utilizadas.

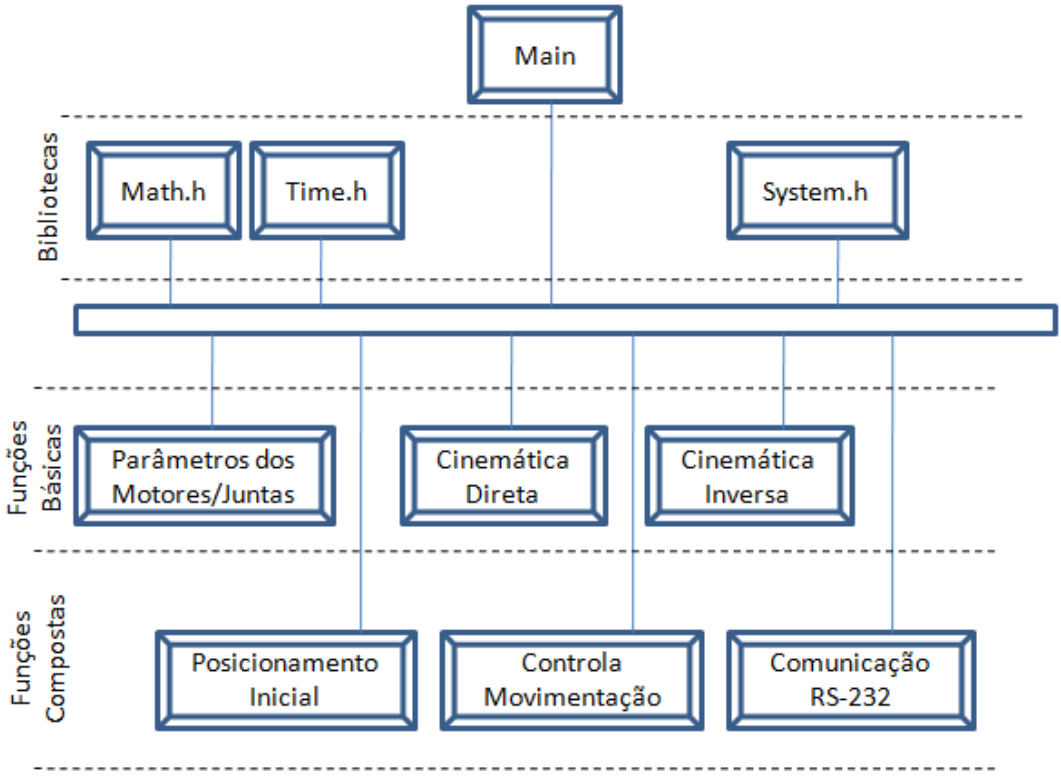


Figura 5.4: Arquitetura do *software* implementado na *FPGA*

5.6 BIBLIOTECAS DE FUNÇÕES

As funções chamadas de bibliotecas são aquelas que dão as bases ao funcionamento do *software*. Na Figura 5.4 são apresentadas as bibliotecas mais relevantes ao projeto, uma vez que outras também foram utilizadas.

Dentre as bibliotecas destacadas, encontram-se as bibliotecas padrão da linguagem C/C++: *Time.h* e *Math.h*. A primeira habilita uma série de componentes de tempo, que foram utilizados para mensurar parâmetros de desempenho e validar alguns tempos de resposta. Já a segunda permitiu que fossem calculadas uma série de equações matemáticas referentes aos modelos de cinemática direta e inversa. Dessa forma, não foi necessário re-implementar as funções que seriam utilizadas, como seno e cosseno, por exemplo.

Importante ressaltar aqui que as bibliotecas utilizadas no *NIOS* e em outros compiladores de linguagem C são bastante próximos, não só pelas funções utilizadas, mas pelos resultados obtidos.

Outra biblioteca utilizada foi a *system.h*. Essa é uma biblioteca nativa do *NIOS*, e é responsável pela geração de parâmetros de inicialização e de acesso aos periféricos do

sistema. Nessa biblioteca são encontrados os endereços de acesso aos periféricos. Desse modo, é possível até que o desenvolvimento de *hardware* de um determinado projeto seja feito por uma pessoa e a parte de *software* embarcado seja feita por outra, uma vez que esses parâmetros encontram-se discriminados nessa biblioteca, diminuindo a probabilidade de erros de programação.

5.7 FUNÇÕES BÁSICAS

As funções básicas são responsáveis pela estrutura de movimentação do manipulador. São elas que fornecem os subsídios matemáticos e do modelo para prover a movimentação. Esse bloco é composto por um conjunto de funções que trata de operações especificas, e que servem como base para a movimentação, apesar de não serem provedoras de movimentação (Figura 5.4).

As principais dessas funções serão explicitadas abaixo:

a) *Parâmetros dos Motores/Juntas*: nessas funções, são definidas as constantes do processo de movimentação do manipulador. É aqui que são determinados os parâmetros das juntas, bem como os parâmetros dos motores, como a resolução de cada um deles. Esse conjunto foi criado para facilitar as adaptações do projeto, em caso de necessidade;

b) *Cinemática Direta*: bloco que implementa os cálculos de cinemática direta, descrito no capítulo 4. Recebe como entrada um vetor de coordenadas de juntas – {θ1, θ2, d3, θ4, θ5}. Retorna um vetor de coordenadas cartesianas – {Tx, Ty, Tz, Rz, Ry, Rx};

c) *Cinemática Inversa*: bloco que implementa os cálculos de cinemática inversa, descrito no capítulo 4. Recebe como entrada um vetor de coordenadas cartesianas – {Tx, Ty, Tz, Rz, Ry, Rx}, seguindo a notação RPY. Retorna um vetor de coordenadas de juntas – {θ1, θ2, d3, θ4, θ5};

5.8 FUNÇÕES COMPOSTAS

As funções compostas são responsáveis por permitir e produzir as entradas e saídas concretas da *FPGA*. Dessa forma, é o bloco das funções responsáveis pela movimentação

do manipulador – acionando os 5 eixos. Esse bloco utiliza as bibliotecas e as funções básicas para produzirem suas ações (Figura 5.4).

5.8.1 Posicionamento inicial

O posicionamento inicial é um desses blocos compostos. Como comentado no capitulo quatro, essa é uma função delegada ao *hardware* embarcado na *FPGA*. Contudo, ela é iniciada pelo *NIOS*, após sua energização. O microprocessador indica ao *hardware* que as juntas do manipulador devem voltar para a posição inicial, através do pino *zera_pos*. Em seguida, fica verificando as entradas *Pos_M1*, *Pos_M2* e *Pos_M3*, até que essas indiquem que os sensores de fim de curso foram ativados.

5.8.2 Controla movimentação

O padrão de movimentação buscado por esse trabalho refere-se à movimentação em linha reta, visando à soldagem em cordões, para reparar os danos causados pela cavitação a uma pá de turbina. Para tanto, o controle da movimentação é peça fundamental desse processo.

O sistema de controle de movimentação também é considerado um dos blocos compostos. Nesse caso, é responsável pela movimentação a partir do ponto zero do manipulador até o ponto final desejado pelo usuário. E, como comentado no capítulo 1, até essa etapa do projeto, são passados para o controle do manipulador um ponto inicial e um ponto final, ambos em coordenadas cartesianas – {Tx, Ty, Tz, Rz, Ry, Rx}.

Para realizar tal movimentação, seu controle é dividido em duas etapas: *Movimentação 1* e *Movimentação 2*:

- *Movimentação 1*: é a movimentação entre o ponto zero do manipulador e o ponto inicial desejado pelo usuário. Tal movimentação é relativamente simples, não demandando um controle preciso de trajetória, ou de velocidade. Sua única restrição refere-se a não determinar movimentos que o manipulador não esteja habilitado a fazer;

O algoritmo da Movimentação 1, entre o ponto zero do manipulador e o ponto inicial do movimento desejado pelo usuário é descrito abaixo:

a) Transformam-se as coordenadas cartesianas do vetor do ponto inicial da movimentação desejada pelo usuário (*PIn*) em coordenadas de juntas, através da função de *cinemática inversa* (Q_{In});

b) Calcula-se a distância entre a posição angular atual desejada (Q_{In}) e a posição angular atual (Q_{at});

c) Calcula-se o número de passo a serem dados por cada um dos motores;

d) Determina-se a velocidade de movimentação de cada eixo;

e) É habilitada a movimentação dos 3 eixos;

f) À medida que os sinais de conclusão da movimentação de cada eixo vão chegando ao microprocessador, através dos pinos *passos_a_M1*, *passos_a_M2* e *passos_a_M3* (contagem de pulsos), suas respectivas movimentações vão sendo desabilitadas;

Já o algoritmo da *Movimentação 2* é mais complexo, uma vez que deseja-se seguir uma trajetória determinada – no caso, uma linha reta – e respeitar as restrições de velocidade associadas ao movimento de soldagem.

Para realizar o processo de movimentação, seguindo de uma linha reta, optou-se pela estratégia de diminuir ao máximo o tempo de atualização do manipulador. Para tanto, existe uma variável, também determinada nos *parâmetros dos motores*, que indica a velocidade desejada para a atualização da frequência de movimentação dos motores. Atualmente, utiliza-se o valor de 0,05 segundos por atualização.

Importante frisar que a diminuição desse tempo acarreta em aumento da precisão do movimento, em compensação aumenta o número de iterações do algoritmo. Tal diminuição é mais complexa de realizar-se, em função dos tempos máximos de atualização das frequências dos *drivers* dos motores.

Desse modo, busca-se aproximar os pontos dentro da reta a ser desenvolvida, de modo a garantir que o manipulador percorra um caminho que seja o mais próximo de uma reta. No capítulo 6, são apresentados os resultados obtidos no *Matlab* a respeito desse modelo, bem como os resultados práticos obtidos pelo manipulador.

Dada a complexidade observada na *movimentação 2*, bem como as dificuldades computacionais em realizar cálculos tão pesados, foram desenvolvidos dois algoritmos para implementá-la:

a) *Real Time Algorithm - RTA*: Algoritmo que determina incrementos de posição em tempo real com a movimentação das juntas do manipulador. Para cada pequena movimentação do manipulador, são recalculados todos os parâmetros de velocidade e posição de cada junta. A Figura 5.5 mostra esse algoritmo;

b) *List Algorithm – LA:* Algoritmo calcula todos os incrementos de movimentação, calculando velocidades e direções, de maneira seqüencial e definindo vetores com essas informações. Só então é iniciado o movimento das juntas do manipulador. É melhor dividido em duas partes:

   - *Determinação de passos*: é a etapa onde o microprocessador calcula a trajetória e os passos a serem dados por cada junta do manipulador. São determinados vetores de posição, direção e velocidade que serão responsáveis pela movimentação do manipulador;

   - *Atuação*: é a movimentação entre os pontos Inicial (PIn) e final (PFn), solicitados pelo usuário, seguindo os vetores de direção e velocidade obtidos na *determinação de passos*.

O segundo algoritmo foi desenvolvido em função da experimentação do primeiro sobre o manipulador. Ao testar o *RTA*, observou-se que a movimentação das juntas 1 e 2 do manipulador não acontecia de maneira suave, gerando perda de passos dos motores e, por consequência, fazendo com que a ferramenta do manipulador não alcançasse a posição final desejada. Tais problemas eram ocasionados pela quantidade de cálculos que eram feitos entre cada iteração das juntas do manipulador, dentro do *NIOS*.

Desse modo, optou-se por uma segunda abordagem, onde os cálculos são todos feitos na sequencia, e só depois são enviados aos atuadores das juntas. Mais uma vez é importante citar que essa abordagem só funciona nesse manipulador porque seus atuadores são motores de passo que, em teoria, não devem perder passos. Então, com o movimento mais suave, observou-se uma trajetória mais correta, a ser melhor explicada no capitulo 6.

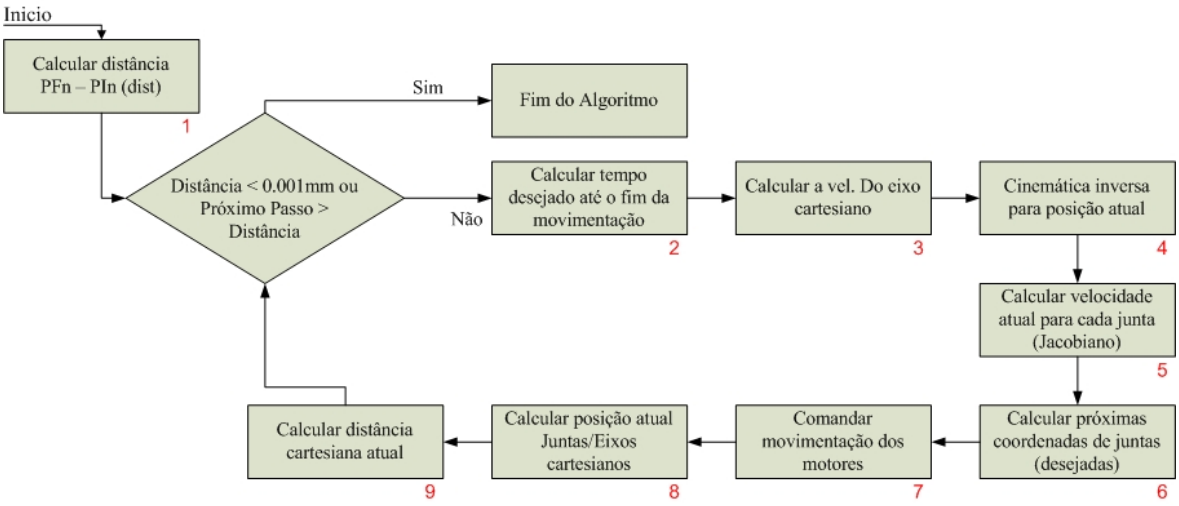


Figura 5.5: Algoritmo de movimentação *RTA*

A seguir, uma descrição as etapas do algoritmo *RTA*:

1) Inicialmente, calcula-se a distância entre os pontos a serem seguidos no processo de movimentação (dist).

   Se a distância entre o ponto inicial (atual) e o ponto desejado para o final do movimento for menor que 0.001 mm, ou se o próximo incremento de posição a ser dado nos eixos ultrapassar o ponto desejado, o algoritmo chega ao fim. Caso contrário, prossegue abaixo;

2) Dada a velocidade de soldagem, previamente cadastrada, no módulo de parâmetros dos motores, determina-se o tempo desejado até o final da movimentação;

3) Dado o tempo desejado para o fim da movimentação do manipulador, é possível determinar a velocidade cartesiana para cada eixo do manipulador, considerando a trajetória em linha reta;

4) Determina-se a posição atual das juntas;

5) Através do Jacobiano, determina-se a velocidade de cada junta do manipulador. Essa é uma operação bastante pesada computacionalmente, uma vez que as equações do Jacobiano desse manipulador são bastante extensas.

Visando aumentar a velocidade desse processo, optou-se por simplificar as equações do Jacobiano, calculando-se previamente equações repetidas e também simplificando os termos que o compõem.

Outro importante aspecto desse passo é a necessidade de que os movimentos das juntas sejam muito pequenos, uma vez que, como citado no capítulo 4, o Jacobiano é uma função de derivadas. Dessa forma, sua correta aplicação e validade dependem de variações muito pequenas de cada parâmetro, de cada vez;

6) Calculam-se as variáveis de juntas desejadas, em função do tempo de atualização;

7) Comando de movimentação dos motores. Esse módulo é bem parecido com o módulo apresentado para o sistema de *Movimentação 1*. A Figura 5.6 apresenta o algoritmo desse módulo;

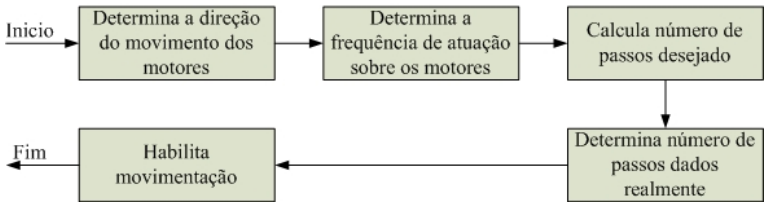


Figura 5.6: Algoritmo de comando dos motores (RTA)

Nesse ponto é importante observar que o número de passos dado é diferente do valor dos passos desejados. Isso porque o divisor de freqüência trabalha apenas com números inteiros. Desse modo, é necessário corrigir essa possível perda e recolocá-la nos cálculos;

8) Determina a posição do manipulador em termos cartesianos e calcula a posição real das juntas;

9) Calcula a distância atual entre a posição do manipulador e a posição desejada em P_{Fn}.

O diagrama do algoritmo *LA* pode ser visto na Figura 5.7. É fácil observar, comparando as figuras **Figura 5.5** e Figura 5.7, que muitos blocos são os mesmos. As diferenças ficam por conta do momento de atuação sobre os motores. Enquanto no *RTA*, o passo 7 do algoritmo é a atuação sobre o manipulador, no *LA*, nesse passo, grava-se mais uma posição na lista de

posições e velocidades que servirão de guia para a movimentação das juntas, que só acontece no passo 10. Já o passo 10 pode ser melhor visto na Figura 5.8.

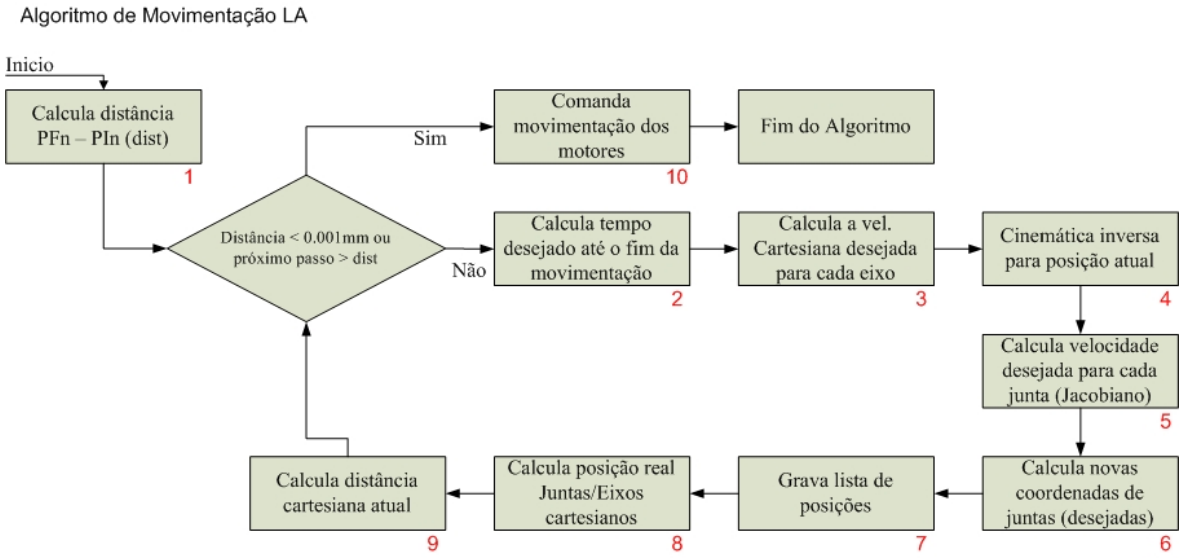


Figura 5.7: Algoritmo de movimentação LA

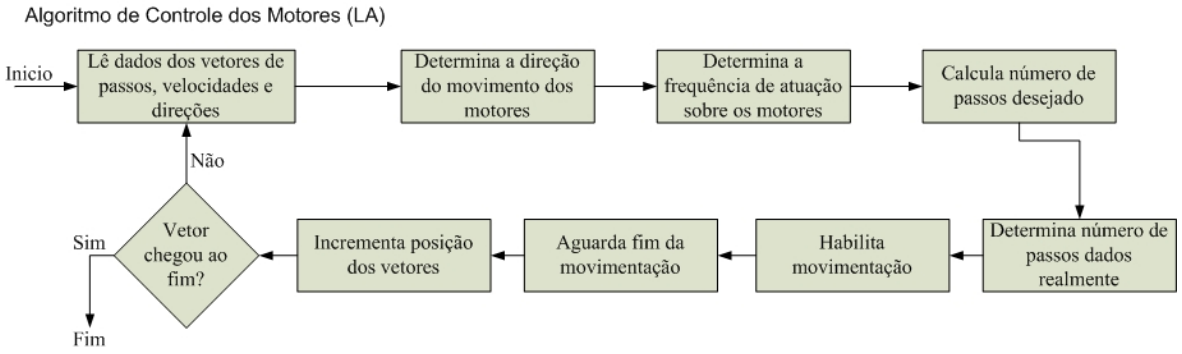


Figura 5.8: Algoritmo de controle dos motores (LA)

5.8.3 Comunicação RS-232

Esse é o último dos blocos das funções compostas. É baseada em boa parte nas bibliotecas instanciadas pelo *NIOS*. E, é responsável por permitir a comunicação com os motores do PanTilt – eixos 4 e 5 do manipulador. A seguir, seguem as funções mais importantes no trato com a porta serial dentro do *NIOS*:

- *IORD_ALTERA_AVALON_UART_RXDATA* (*Endereço_Base*): Lê os dados provenientes do buffer da porta serial, a partir de um endereço inicial;
- *IOWR_ALTERA_AVALON_PIO_TXDATA* (*Endereço_base*,*Valor*): Função que escreve dados na serial, a partir de um endereço ;

Essa comunicação é feita através de um protocolo proprietário da Directed Perception. Esse protocolo foi implementado na *FPGA*, de modo a comunicar-se com esses atuadores. Tal protocolo pode ser visto em maiores detalhes no manual do fabricante (Directed Perception).

Esses atuadores apresentam o fator complicador ao ter um protocolo de comunicação. Isso porque, como o comando sobre o motor não é imediato, há uma dificuldade em sincronizar as movimentações dos dois últimos eixos com os três primeiros.

Esse equipamento apresenta ainda uma característica especial: quando ele está em comunicação, não permite que outrem utilize o barramento para comunicações com outros periféricos.

5.9 MAIN

O bloco da *main* pode ser considerado com a espinha dorsal do projeto de *software*. É nele que fica possível visualizar com maior clareza a arquitetura do *software* desenvolvido para esse projeto. Nesse bloco, são unidas as diversas estruturas citadas anteriormente e coordenadas em suas atividades, gerando um roteiro para o *software* de controle da movimentação.

O algoritmo descrito pela *main* pode ser visto na Figura 5.9, onde são mostrados os diversos blocos do projeto, unidos e integrados.

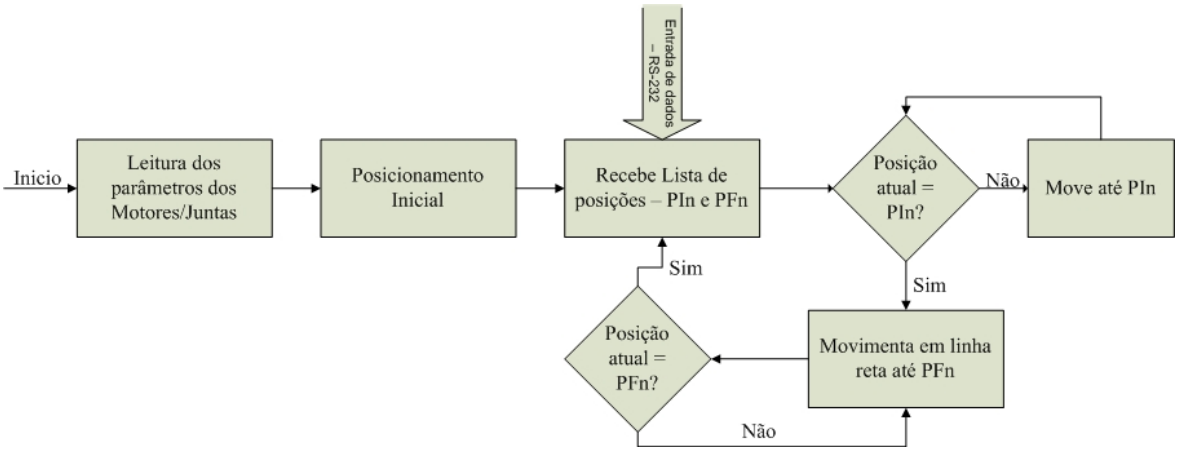


Figura 5.9: Algoritmo da função *Main*

5.10 CONCLUSÕES DO CAPÍTULO

Nesse capítulo, foi apresentada a arquitetura do projeto. Dentre os itens apresentados, foram vistas as estruturas do *chip* implementado na *FPGA* que é responsável pelo controle dos cálculos de movimentação do manipulador.

Por outro lado, foram descritas também as estratégias referentes a movimentação do manipulador e seus algoritmos de controle, tratando o passo a passo de sua movimentação, desde o ponto zero até a conclusão de um movimento.

Adicionalmente, foram descritos também os dois algoritmos implementados para a realização da movimentação, seguindo a trajetória desejada, através de cálculos em tempo real e através de lista de posições.

6 RESULTADOS E TESTES

6.1 ASPECTOS GERAIS

Neste capitulo serão discutidos alguns dos resultados obtidos ao logo da elaboração e implementação desse trabalho, bem como os testes que validaram as metodologias e *software*s utilizados.

Serão abordados também os testes de validação das equações de cinemática direta e inversa e a implementação da metodologia para movimentação do manipulador. Além disso, será analisada a simulação da trajetória seguida pelo manipulador, através do *software* MatLab.

Outro ponto a ser discutido nesse capitulo é a trajetória efetivamente realizada pelo manipulador, comparando os algoritmos *RTA* e *LA*, descritos no capitulo 5.

Além disso, serão analisados aspectos da implementação do *hardware*, como a ocupação da *FPGA*, o tempo de resposta para algumas das operações e aspectos sobre o funcionamento do manipulador.

O primeiro aspecto de resultado a ser comentado é a própria arquitetura de *hardware* e *software* implementada nesse projeto. A Figura 6.1 mostra uma versão simplificada desse conjunto.

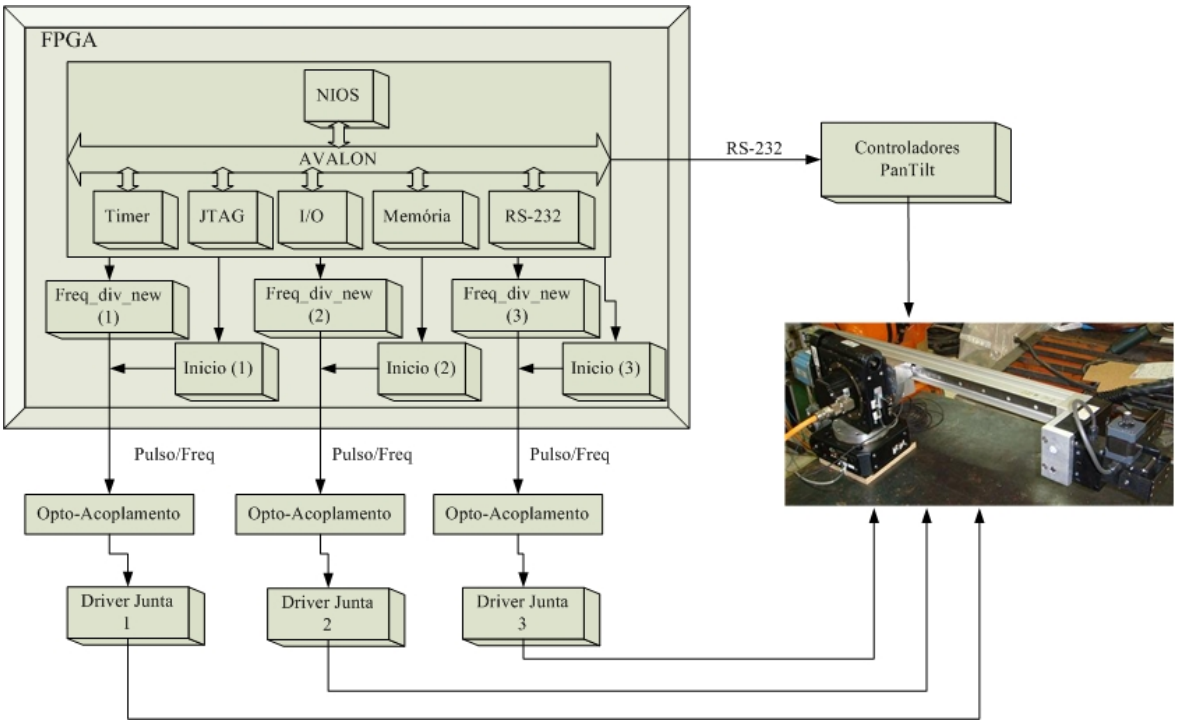


Figura 6.1: Arquitetura completa do projeto

A Figura 6.1 dá destaque a alguns pontos mais importantes do projeto, como a estrutura embarcada no microprocessador, os blocos de *hardware* implementados na *FPGA* e as estruturas externas a *FPGA*, responsáveis pelo acionamento das juntas do manipulador.

Já a Figura 6.2 mostra o manipulador implementado.

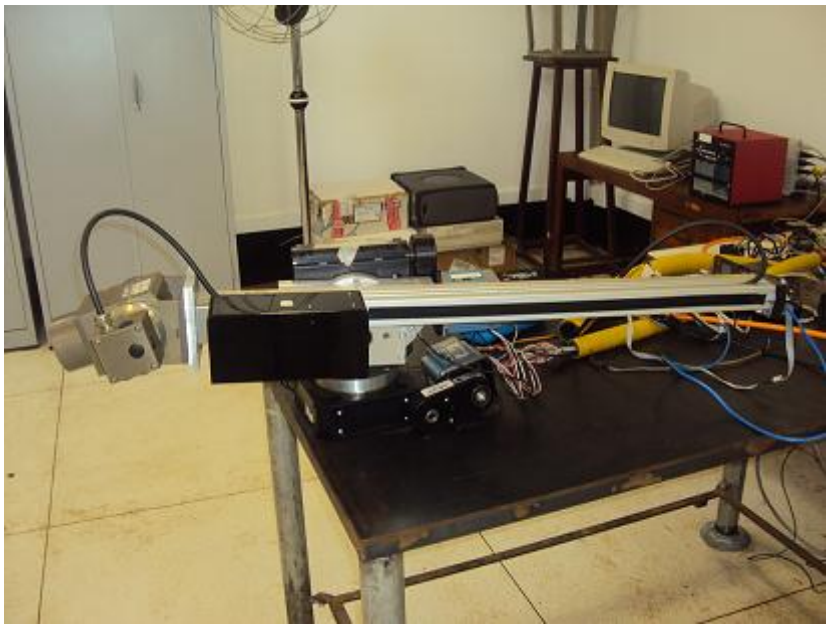


Figura 6.2: Manipulador montado no GRACO

6.2 VALIDAÇÃO DA CINEMÁTICA DIRETA E INVERSA

No capítulo 3, foram discutidas e elaboradas as equações do modelo cinemático direto e inverso do manipulador. Entretanto, dada à complexidade desses cálculos, principalmente os da cinemática inversa, a utilização de um método de validação mostra-se bastante necessário, uma vez que erros podem ser cometidos.

Tal validação busca confirmar a validade das equações encontradas ao longo da modelagem do manipulador. E, também se busca determinar as singularidades do processo de calculo computacional uma vez que o processador utilizado pode trabalhar de maneiras diferentes em alguns aspectos – principalmente em regiões próximas ao valor zero ou em algumas aproximações decimais.

Em função desses aspectos, optou-se pela utilização de dois *software*s distintos. O primeiro apresenta um simulador gráfico, enquanto o segundo apresenta um simulador numérico de maior precisão. Em ambos os casos, buscou-se:

a) Obter a posição da ferramenta a partir das variáveis de juntas e,

b) Realizar cálculos de cinemática inversa e obter as mesmas variáveis de juntas que iniciaram o processo (usando a resposta obtida – vetor de posições cartesianas).

6.2.1 Simulador gráfico

A partir da definição das equações de cinemática direta e inversa, optou-se pela utilização do *RobModel*, *software* desenvolvido na UnB, no laboratório do Graco, pelo aluno Luciano Selva Ginani.

Nesse *software*, é possível visualizar a movimentação do manipulador, bem como observar se os valores obtidos através das equações referentes à cinemática estão coerentes. A Figura 6.3 mostra o robô desenvolvido, visualizado dentro do *RobModel*.

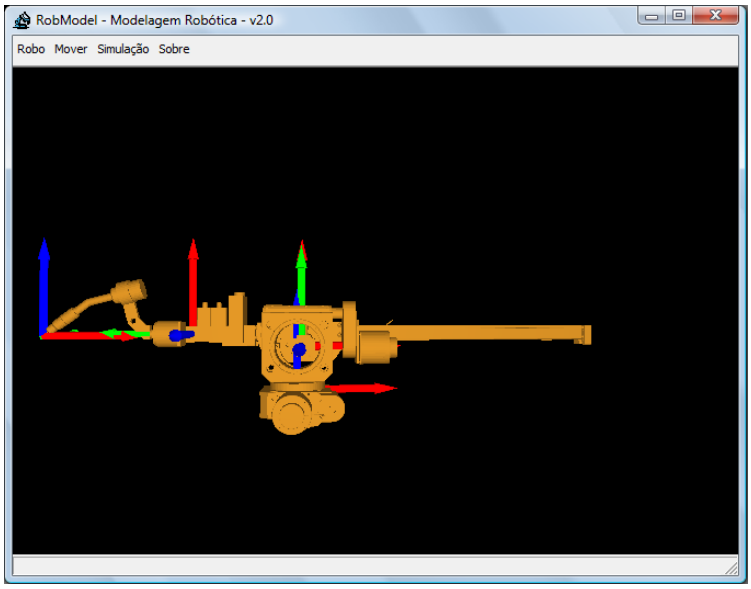


Figura 6.3: *RobModel*, com manipulador desenhado

Esse *software* apresenta uma série de arquivos de configuração, onde são assinalados os parâmetros D-H, bem como a determinação dos eixos do manipulador e as equações de cinemática direta e inversa. Desse modo, pode-se observar a semelhança entre o modelo simulado e o modelo construído em laboratório.

O *RobModel* apresenta varias outras características positivas, visando a realização dos testes. Uma dessas é que é possível inserir nele as restrições de posição do manipulador. Desse modo, já se comprovam algumas impressões iniciais, através do mero conceito visual.

Contudo, a característica do *RobModel* que representou maior ganho para esse projeto foi à capacidade de simular a movimentação dos eixos, tanto usando a cinemática direta quanto a inversa. A Figura 6.4 mostra a tela de configuração do posicionamento dessas juntas.

Figura 6.4: Menu *Mover Juntas*

Como é visto na Figura 6.4, é possível realizar a movimentação através da cinemática direta (Figura 6.4 a) ou cinemática inversa (Figura 6.4 b). Desse modo, é possível avaliar a exatidão de uma movimentação, aliando a ferramenta gráfica aos parâmetros definidos. Outra importante ferramenta de verificação dos modelos obtidos era o fato de o RobModel fornecer a matriz de posição do manipulador, que pode ser vista na Figura 6.5.

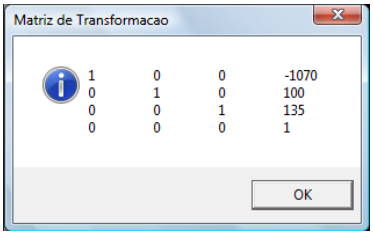


Figura 6.5: Matriz de posição do Manipulador

O principal teste aplicado através desse *software* foi a realização de movimentações utilizando a cinemática direta, alterando o valor de ângulos e distâncias e indicando essas posições. Em seguida, foram realizadas movimentações utilizando essas posições cartesianas, e verificando as posições das juntas, através da cinemática inversa.

6.2.2 Simulação numérica

A simulação gráfica mostrou-se bastante eficiente num primeiro momento, como ferramenta de verificação de erros mais grosseiros e iniciais, observados nas equações cinemáticas. Contudo, a estrutura do *software* não permitia o teste de uma quantidade maior de posições, num curto espaço de tempo. Isso porque demandava a inserção dos parâmetros das juntas individualmente, para cada teste. Desse modo, buscou-se uma

solução alternativa que permitisse o teste de diversas posições de maneira mais rápida, eficiente e sistemática.

Desenvolveu-se, então, um *software* de nome sugestivo "*verifica_tudo*". Esse *software* foi desenvolvido em linguagem C, utilizando o *software* DevCpp. O *verifica_tudo* utiliza as funções básicas citadas no capítulo 3, para gerar coordenadas cartesianas e de juntas.

Esse *software* realiza um teste semelhante ao feito utilizando o simulador gráfico, mas executa uma varredura mais completa, percorrendo uma quantidade de posições muito maior. O procedimento executado pelo *software* é explicitado abaixo e pode ser visto em maiores detalhes na Figura 6.6.

- Determina-se um vetor de posição inicial das juntas: $Q = \{\theta1, \theta2, D3, \theta4, \theta5\}$;
- Calcula-se a cinemática direta, gerando o vetor de posições cartesianas: $P = \{Tx, Ty, Tz, Rz, Ry, Rx\}$;
- Calcula-se a cinemática inversa em relação a *P*, produzindo-se o vetor Q_{new};
- Comparam-se os vetores *Q* e Q_{new};
- Caso haja uma diferença maior que *0.001mm* entre algum dos itens de *Q* e Q_{new}, é gerado um *log* do erro;
- Incrementa-se o vetor *Q*, e o processo recomeça.

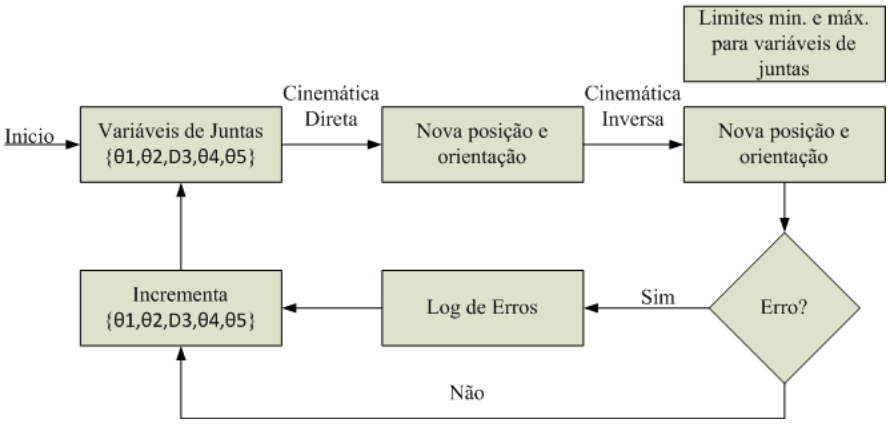


Figura 6.6: Algoritmo do *verifica_tudo*

Escolhendo-se um incremento (passo) tão pequeno quanto desejado, o teste pode ficar mais completo e obviamente mais demorado. Além disso, existe a questão dos limites utilizados para a variação de cada um dos parâmetros. Através desses testes, foi possível analisar os algoritmos implementados. O resultado final desses testes pode ser visto na Tabela 6.1.

Tabela 6.1: Testes para o *software verifica_tudo*

Variáveis de Juntas	Limites		Passo	Posições Testadas	Total de posições testadas	Erros
θ1 (º)	-176	180	4	90	469640700	0 (0%)
θ2 (º)	-45	45	3	31		
D3 (mm)	350	440	10	10		
θ4 (º)	-92	92	2	93		
θ5 (º)	-90	90	1	181		

Como apontado na tabela 6.1, o erro foi de 0% ao considerarmos como erros apenas distâncias maiores que 0,001 mm.

Através desses testes, foi possível considerar como válidas as equações de cinemática direta e inversa utilizadas, além de garantir a eficiência dos algoritmos que implementam essas equações.

É importante ressaltar que parte da capacidade do manipulador de seguir uma trajetória correta depende do tratamento matemático dado pelo microprocessador as varáveis e as equações implementadas. E, como o *NIOS* utiliza uma linguagem C, tem uma arquitetura de 32 bits e usa a mesma biblioteca *math.h* do PC onde rodou o *verifica_tudo*, poder-se-ia supor que as equações funcionariam nele da mesma maneira que na plataforma PC.

Desse modo, o microprocessador que foi implementado estava habilitado a calcular as equações da forma desejada.

6.3 VALIDAÇÃO DO ALGORITMO DE TRAJETÓRIAS EM LINHA RETA

A validação dos algoritmos de cinemática direta e indireta garante o posicionamento do manipulador. Contudo, o ajuste de trajetória demanda testes mais específicos. Para a

execução desses, foi utilizada a ferramenta MatLab, para geração e visualização das trajetórias.

No capítulo 5, foi explicitado o algoritmo para movimentação do manipulador seguindo uma trajetória especifica. Tal algoritmo foi implementado no *software robô_final_v01*, gerado em linguagem C, no *software* DevCpp. Tal *software* possui as mesmas funcionalidades do *software* implementado no microprocessador *NIOS*, exceto a parte de acionamento dos motores. Desse modo, simula toda a funcionalidade do manipulador, facilitando a observação e correção de erros, bem como a execução de testes.

O uso desse *software* permite a validação do modelo de geração de trajetória adotado, num ambiente controlado, onde puderam ser isolados os erros de programação e das equações utilizadas.

6.3.1 *Software* para teste de posicionamento final

Dentro do *software robô_final_v01*, foi necessário implementar uma entrada de dados, via *software*, onde são determinadas as coordenadas cartesianas dos pontos iniciais e finais da movimentação. A partir daí, o *software* realiza uma simulação de movimentação de cada junta do manipulador, observando as restrições de velocidade e de tempo apontadas.

Como comentado no capítulo 5, para determinar uma trajetória em linha reta – como desejada para a implementação de um cordão de solda – busca-se diminuir o tempo entre as iterações com os motores, de modo a aproximar o movimento de seu elemento terminal por uma reta. Novamente, quando a distância entre o ponto atual e o próximo ponto a ser alcançado é menor que *0.001 mm* (limite), o movimento é finalizado. O movimento também é finalizado caso o próximo incremento de posição seja maior que essa distância.

Outra variável analisada nesse *software* foi o tempo entre iterações (*TEI*). Como comentado no capitulo 5, a *TEI* utilizada nesse projeto é de *0.05 s*. E, através desse *software* foi possível observar a variação do erro entre a posição desejada e a posição simulada pelo *software*. A Figura 6.7 traz um gráfico onde se pode observar a variação desse erro para três deslocamentos distintos, diferenciados pela distância entre a posição inicial e a posição desejada para o fim do movimento.

Para elaboração desse gráfico, foram determinadas três posições iniciais e finais distintas, diferenciadas pela distância entre o ponto inicial e o final da movimentação. As distâncias foram 79,598 mm, 61,934 mm e 63,18 mm. E, variou-se o tempo entre iterações, para determinar o erro entre a posição final desejada e a posição apontada pela simulação.

Pode-se observar no gráfico, por exemplo, que sendo a distância entre os pontos inicial e final do movimento igual a 79,598 mm, o erro na posição final foi de aproximadamente 0,25 mm para uma *TEI* de 0,5 s. Também é possível observar que esse erro cai para quase 0 mm, quando a *TEI* é reduzida para 0,01 s.

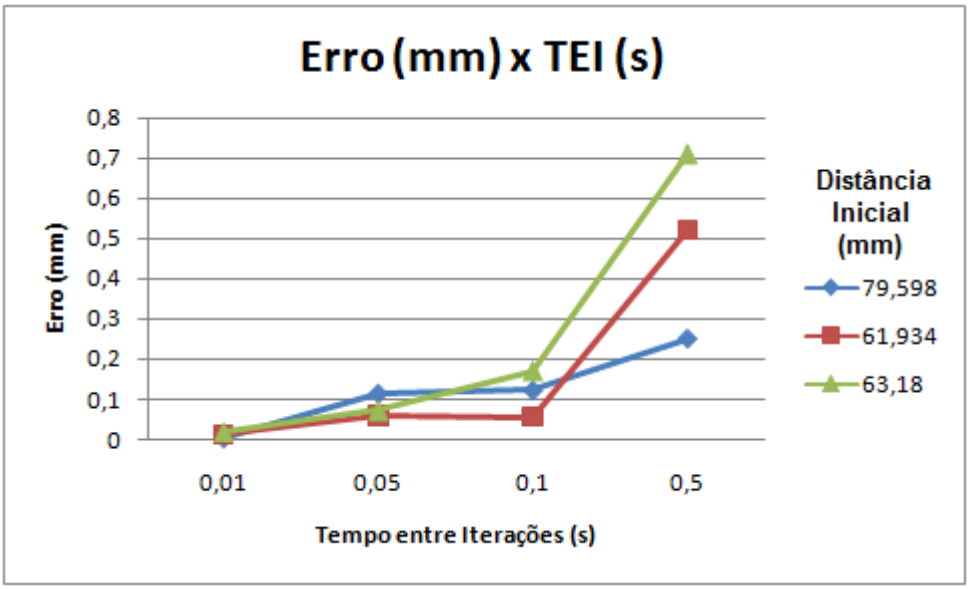


Figura 6.7: Gráfico indicando a variação do erro entre a posição desejada e a posição alcançada

Logo, observando as três curvas apresentadas na Figura 6.7, é possível concluir que a precisão do posicionamento do manipulador aumenta bastante, caso o tempo de iteração vá de *0,5 s* para *0,1 s*. Contudo, já não apresenta uma variação tão grande para as opções *0,1* e *0,05 s*.

O gráfico mostra ainda que a opção ideal é a utilização do tempo de iteração de *0,01 s*, onde o erro foi sempre menor que o das outras opções. Essa escolha, na implementação final no *NIOS* implica na velocidade que o algoritmo deve ser rodado para que o movimento seja realizado corretamente. Isso porque, quanto menor o tempo entre iterações, mais cálculos devem ser feitos, num mesmo tempo, visando respeitar as restrições de tempo e velocidade do sistema.

6.3.2 *Software* para validação da trajetória percorrida

O *software* anterior, *robô_final_v01*, à medida que vai realizando o algoritmo de geração e movimentação ao longo da trajetória vai preenchendo dois arquivos "*txt*", com as posições percorridas, indicando a posição cartesiana da ponta da ferramenta (até o momento, vista como a ponta do manipulador) e os valores das variáveis das juntas. Utilizando esses arquivos, é possível analisar dois aspectos:

- Através do arquivo que contém as posições das juntas ao longo do tempo, é possível observar se os valores não variam bruscamente, o que invalidaria a aplicação do Jacobiano. Tal validação é feita através de um programa em linguagem C, que observa a posição atual e a anterior, comparando-as, indicando a maior variação dentre as posições registradas;

- Através do arquivo que contém as coordenadas cartesianas ao longo de cada iteração, pode-se analisar a trajetória percorrida pelo manipulador. Esse segundo aspecto é capaz de indicar as tendências de erro do posicionamento do manipulador.

Para realizar essa análise, foi utilizado o *Matlab*, como ferramenta gráfica. Dessa forma, foram definidos três vetores – *Px*, *Py* e *Pz* –, contendo os conjuntos de posições referentes aos termos *Tx*, *Ty* e *Tz* do vetor de coordenadas cartesianas. Isso porque a orientação foi considerada constante nesses testes, onde apenas a posição do elemento terminal era considerada.

Então, sejam:

$$\begin{gathered} P_1 = P_{Inicial} = \left[P_{x1}, P_{y1}, P_{z1}\right] \\ P_2 = \left[P_{x2}, P_{y2}, P_{z2}\right] \\ \ldots \\ P_{n-1} = \left[P_{xn-1}, P_{yn-1}, P_{zn-1}\right] \\ P_n \cong P_{Final} = \left[P_{xFinal}, P_{yFinal}, P_{zFinal}\right] \end{gathered} \tag{6.1}$$

Têm-se que os *n* vetores *P* constituem pontos onde *deseja-se* que a ferramenta do manipulador percorra, sendo a trajetória entre esses pontos desconhecida. A equação (**6.2)** mostra um dos conjuntos de vetores – P_{In} e P_{Fn} –, utilizados para realização desses testes.

$$P_{In} = [-390{,}00; 100{,}00; 140{,}00]$$
$$P_{Fn} = [-359{,}99; 38{,}065; 100{,}00] \quad (6.2)$$

As tabelas 6.2 e 6.3 mostram as análises de posição para a trajetória desejada, em termos cartesianos e em termos de coordenadas de juntas, respectivamente:

Tabela 6.2: Análise de posição em termos cartesianos, para os vetores P_{in} e P_{fn}

	Tx (mm)	Ty (mm)	Tz (mm)
Pin	-390	100	140
PFn (desejado)	-359,999	38,065	100
PFn (alcançado)	-359,94	37,96	99,93
Erro (Distancia)	-0,059	0,105	0,07
Distância inicial (total)	79,59 mm		
Erro Final (Distância)	0,113965 mm		
Tempo Entre Iterações (TEI)	0,05s		

Tabela 6.3: Análise de posição em termos de coordenadas de juntas, para os vetores P_{in} e P_{fn}

	θ1 (º)	θ2 (º)	D3 (mm)
QIn	0	0	390
QFn (desejado)	10	6,59	346,76
QFn (alcançado)	10,01859	6,6033	346,7
Erro (Distância)	-0,01859	-0,0133	0,06
Erro Final (Distância)	0,064206527 mm		

Importante ressaltar que esses resultados, para os itens *alcançados*, foram obtidos através do *software robô_final_v01*.

O erro entre o ponto final desejado e o alcançado é explicada pela arquitetura implementada, uma vez que, alcançando-se uma distância *x* entre o *ponto atual* onde encontra-se a ferramenta do manipulador e o *ponto desejado* é menor que o próximo passo do manipulador. Desse modo, evita-se que a ferramenta ultrapasse o ponto desejado.

A partir dos pontos obtidos, é possível assinalar os pontos que serão alcançados pelo manipulador. A Figura 6.8 mostra uma comparação entre a trajetória desejada e os pontos alcançados pelo manipulador. Nessa figura, a reta de cor azul indica os pontos percorridos

pelo manipulador, enquanto a reta de cor vermelha indica os pontos por onde ele deveria passar. Como pode-se observar, elas estão tão próximas que é inclusive difícil de diferenciá-las.

Para facilitar a observação da proximidade entre essas retas, a Figura 6.9 mostra a mesma comparação, mas utilizando uma escala menor. Nela, pode-se observar que os pontos mostrados encontram-se muito próximos à trajetória desejada.

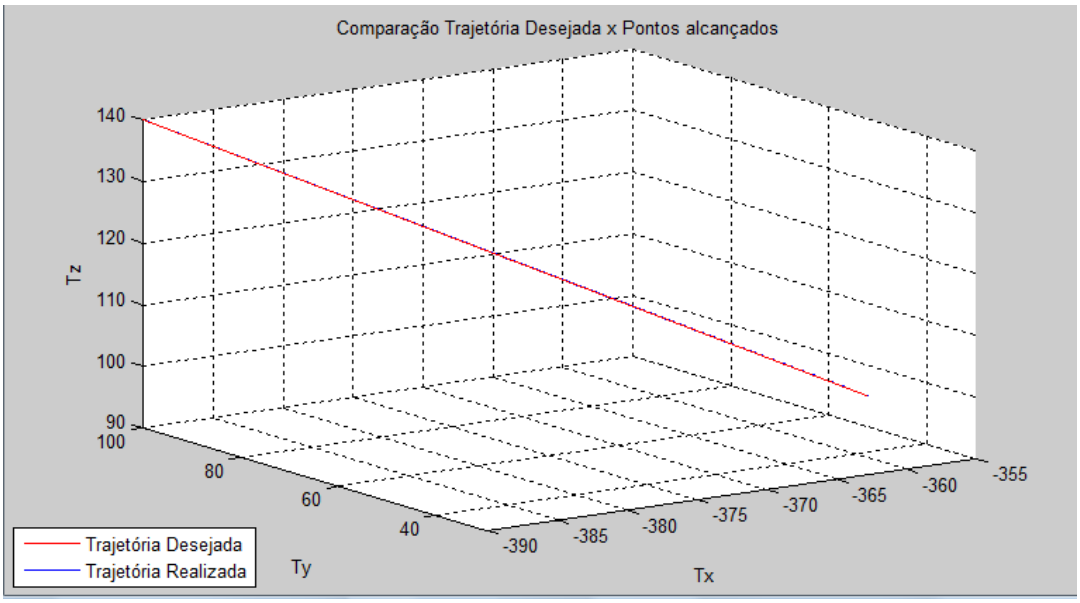


Figura 6.8: Pontos alcançados pelo manipulador

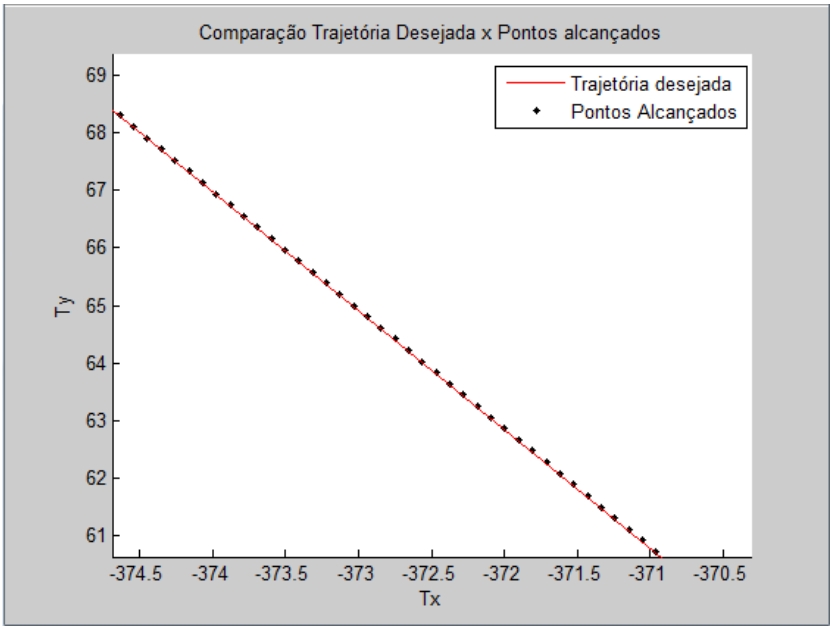


Figura 6.9: Comparação Trajetória Desejada x Pontos Alcançados (escala menor)

Como a distância entre os pontos apontados na Figura 6.9 é muito pequena, é bastante razoável afirmar que a trajetória a ser descrita pelo manipulador é bastante próxima à trajetória desejada. A Tabela 6.4 mostra os erros máximos entre os pontos obtidos e a trajetória desejada.

Tabela 6.4: Erro Máximo entre os pontos obtidos e a trajetória desejada

	Eixo x	Eixo y	Eixo z
Erro máximo (mm)	0,012599	0,024533	0,000997

Considerando um erro máximo de 1mm para soldagens desse tipo, pode-se afirmar que esse erro máximo satisfaz as exigências de projeto.

6.4 IMPLEMENTAÇÃO DO *HARDWARE*

As tabelas 6.5 e 6.6 descrevem a utilização da capacidade da *FPGA*. A Tabela 6.5 apresenta a relação direta entre a capacidade total da placa em relação aos dispositivos utilizados durante a elaboração do *hardware*. Já a

Tabela **6.6** apresenta uma análise dos itens que compõem esse *hardware* embarcado na *FPGA*.

Tabela 6.5: *FPGA*: Capacidade x Utilização

	Dispositivo	LE	Total de Pinos	Total de bits de Memória	DSP Blocks	PLL	DLL
Capacidade	EP3SL150 F1152C2	142500	744	5630976	384	8	4
Utilizados		4400 (3%)	48 (6%)	2161088 (38%)	4 (1%)	1 (13%)	0 (0%)

Tabela 6.6: Utilização dos recursos pelos principais componentes instanciados na *FPGA*

	Logic Cells	Memory Bits	DSP Elements
Freq_div_new (1)	95	0	0
Freq_div_new (2)	99	0	0
Freq_div_new (3)	99	0	0
Inicial (1)	51	0	0
Inicial (2)	51	0	0
Inicial (3)	51	0	0

NIOS	1510	62912	4
I/O	221	0	0
JTAG	125	1024	0
On-*Chip* Memory	77	2097152	0
SysId	4	0	0
Timer	127	0	0
Rs232	210	0	0

Algumas considerações podem ser feitas a partir da análise sobre a utilização da *FPGA*. Dentre essas, nota-se que o *NIOS* é o maior utilizador de unidades lógicas da placa. Isso porque trata-se de um microprocessador com uma série de opções instauradas para facilitar seu uso, incluindo a interface com o barramento *Avalon*.

Outro aspecto importante no que se refere ao uso dessa placa trata do uso da memória pelo *software* embarcado no *NIOS*. Dentro do *NIOS*, foi instanciado um bloco de memória de 256 Kb, dos quais 158 Kb foram utilizados.

6.5 MOVIMENTAÇÃO DO MANIPULADOR

Como comentado no capitulo 1, o principal objetivo da construção do manipulador descrito nesse trabalho é o desenvolvimento de uma ferramenta autônoma capaz de realizar soldagens em turbinas. E, durante esse processo, serão feitos, sobre a pá da turbina danificada, cordões de solda. Uma simplificação desse processo pode ser vista através da construção de linhas retas de tamanho variável.

Desse modo, o objetivo do controlador desenvolvido nesse presente trabalho é garantir ao manipulador a capacidade de realizar essa ação da maneira mais correta possível (seguindo uma trajetória desejada, com velocidade determinada).

6.5.1 Ferramenta de testes do manipulador

A construção de uma trajetória da ferramenta de um manipulador, como o apresentado nesse trabalho, em linha reta depende apenas da movimentação dos três primeiros eixos, desde que não haja mudança de orientação da ferramenta, considerando o plano XZ. O mesmo é valido, de maneira aproximada, para pequenas variações na coordenada Y. Desse

modo, foram implementados os algoritmos de movimentação apontados no capitulo 5, que movimentavam apenas os eixos 1, 2 e 3, quando desejava-se o deslocamento da ferramenta seguindo uma trajetória de linha reta.

Contudo, visando a realização de testes da capacidade do manipulador seguir a trajetória desejada, foi necessário o desenvolvimento de uma ferramenta que permitisse a visualização dessa movimentação.

Foi, então, desenvolvido um suporte a ser fixado na ponta do manipulador, para que fosse acoplado a esse um lápis. Desse modo, a movimentação do manipulador podia ser registrada e avaliada através da escrita desse lápis, sobre uma folha de papel, fixada a mesa.

Além disso, para viabilizar os testes, e garantir a repetibilidade dos mesmos, foi necessária a definição de outros parâmetros fixos. Então, o robô foi fixado a uma mesa, de modo a não movimentar-se em relação a sua base. Por último, foi montada uma plataforma inclinada em cima da mesa de fixação do manipulador.

A Figura 6.10 mostra a estrutura montada para a realização dos testes do manipulador.

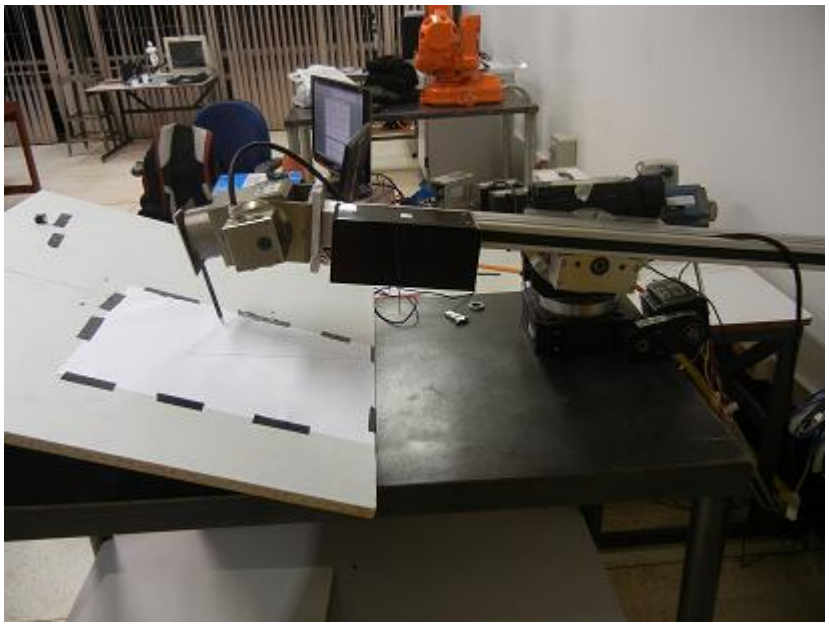


Figura 6.10: Plataforma de Testes do manipulador

Pode-se observar que, na Figura 6.10, que o elemento terminal, suporte da ferramenta, é diferente do mostrado nas figuras Figura 6.1 e Figura 6.2.

Experimentalmente, pode-se aferir que o deslocamento provocado pelos dois eixos finais do manipulador, bem como pelo comprimento do lápis foi de 180 mm no eixo X e, no eixo Z, de – 140 mm, a serem aplicados na equação (3.15).

6.5.2 Comparando os Algoritmos *RTA* e *LA*

No capitulo 5, foram comentados os dois algoritmos implementados no *NIOS*, responsáveis pelo controle de trajetória do manipulador: *RTA* e *LA*. Inicialmente, foi implementado o *RTA*, visando avaliar a capacidade da *FPGA* de realizar os cálculos de cinemática do manipulador e acionar os atuadores.

Entretanto, observou-se que essa solução não atendia aos requisitos de qualidade de movimentação do manipulador, como a manutenção da velocidade e execução da trajetória. Isso porque, a cada iteração do algoritmo, os atuadores dos eixos 1 e 2 tinham sua movimentação habilitada e desabilitada.

Foram feitos testes de movimentação separadamente com os eixos 1, 2 e 3. E, pode-se observar que a movimentação dos eixos 1 e 2 não seguiam a velocidade definida pelo algoritmo. Durante esses testes, foi constatado que os motores dos referidos eixos perdiam passos, em função da aceleração e freio desses, muito rapidamente. Isso acontece em função da capacidade de resposta ao comando de *enable* desses atuadores, que ficava aquém da frequência desejada para uma movimentação suave e correta.

Já o eixo 3 do manipulador não sofria esse tipo de influência de seu atuador, percorrendo sua trajetória sem sobressaltos ou perda de passos.

Em função das limitações dos atuadores dos motores dos eixos 1 e 2, optou-se pela implementação do algoritmo *LA*. Como explicado no capitulo 5, esse algoritmo calcula a trajetória a ser seguida pelo manipulador, determina as posições, velocidades e direções a serem seguidas por cada junta e armazena essas informações em vetores. Esses, após o calculo estar completo, são passados aos atuadores do manipulador. Desse modo, como o comando de *enable* das juntas 1 e 2 é utilizado apenas no inicio e no final do movimento, variando apenas a freqüência dos pulsos de acionamento, o deslocamento é mais suave e sem perda de passos, permitindo o alcance da trajetória e velocidade desejada.

6.5.3 Movimentações

Buscando demonstrar a capacidade de seguir uma trajetória determinada, foram determinados dois testes para o manipulador. O primeiro referia-se ao desenho de um quadrado no plano XY do manipulador. O segundo teste consistia na construção de uma linha reta no plano inclinado colocado sobre a mesa do manipulador, promovendo uma movimentação nos três eixos, X, Y e Z.

- Quadrado no plano XY

Com esse experimento, buscava-se avaliar a movimentação dos eixos 1 e 3 do manipulador, para uma observação do algoritmo de geração de trajetórias. Desse modo, foi determinado um quadrado cujos vértices podem ser vistos na Tabela 6.7.

Tabela 6.7: Vértices do quadrado do experimento

	Coordenada X	*Coordenada Y*	*Coordenada Z*
Posição Inicial	-330	100	0
Vertice 1 (V1)	-400	100	0
Vertice 2 (V2)	-600	-100	0
Vertice 3 (V3)	-600	-100	0
Vertice 4 (V4)	-400	-100	0

Novamente, como comentado no capitulo 5, o manipulador parte de uma posição inicial para então começar sua movimentação.

A Figura 6.11 mostra o resultado dessa movimentação. Nela, pode-se observar que, no fim do movimento, o traço torna-se mais fraco, em função do esforço lateral promovido no lápis. Além disso, pode-se observar que os vértices 3 e 4 não são tão bem definidos quanto o vértice 2. Isso se deve ao fato de que o movimento entre os vértices 1 e 2 é feito somente pela junta prismática do manipulador, enquanto os vértices 3 e 4 são feitos a partir de mudanças bruscas de direção da junta 1 do manipulador.

Contudo, as medições de distância realizadas entre os vértices não apresentaram variações significativas, sendo menores que 1 mm.

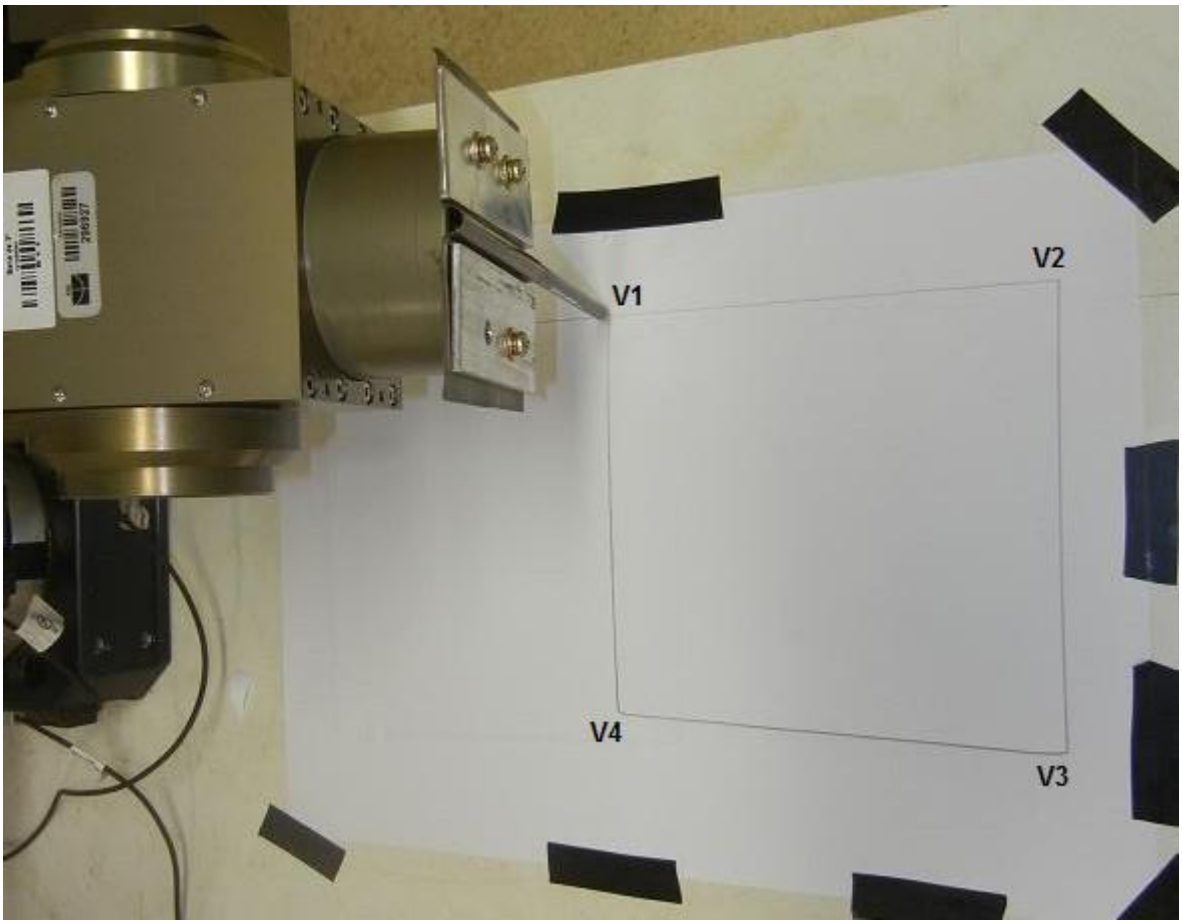


Figura 6.11: Quadrado desenhado pelo manipulador

- Linha reta no plano inclinado X, Y e Z

O segundo experimento buscava acompanhar a movimentação do manipulador fora do plano XY. Para isso, foi utilizado o plano inclinado acoplado a mesa do manipulador onde foram feitos dois testes. O primeiro referiu-se a uma movimentação do manipulador somente nos eixos 2 e 3. O segundo foi feito utilizando os três eixos do manipulador.

A Tabela 6.8 mostra o ponto inicial e final de cada um desses movimentos, sendo que o primeiro é descrito entre os pontos P1 e P2, enquanto o segundo teste foi feito com o deslocamento entre os pontos P1 e P3. A Figura 6.12 mostra o resultado dessas movimentações.

Tabela 6.8: Pontos percorridos no teste sobre o plano inclinado

	Coordenada X	Coordenada Y	Coordenada Z
Posição Inicial	-330	100	0
P1	-420	100	0
P2	-610,78	100	80,67
P3	-638,81	214,18	87,98

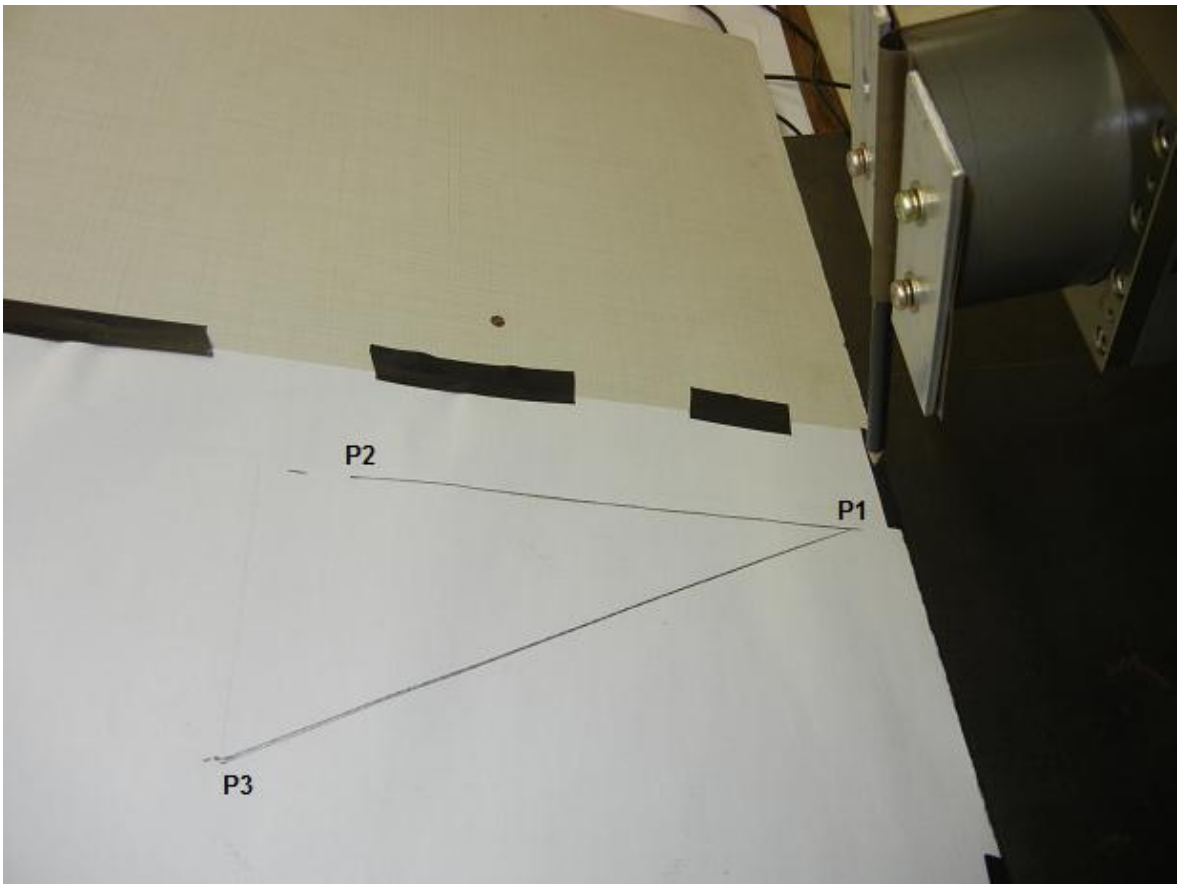


Figura 6.12: Experimento realizado sobre o plano inclinado

6.6 CONCLUSÕES DO CAPÍTULO

Ao longo desse capítulo foram vistos os resultados e as metodologias de validação dos algoritmos e sistemas implementados no manipulador. Além disso, foram abordadas questões de *hardware*, tratando da utilização, capacidade e alguns problemas encontrados.

Abordaram-se também os testes realizados com o manipulador e a integração entre os diversos módulos, com a realização da comunicação e das diversas movimentações. Além disso, foram observadas as diferenças de resultado apresentados entre a implementação das

duas formas de calculo de trajetória – *RTA* e *LA*. Comentou-se também o motivo da opção pelo algoritmo LA, em virtude do tempo de resposta dos atuadores das juntas 1 e 2 do manipulador.

Por último, foram apresentados os resultados práticos da movimentação do manipulador, confirmando a capacidade deste de seguir a trajetória determinada.

7 CONCLUSÕES, ANÁLISESE SUGESTÕES DE TRABALHOS FUTUROS

7.1 ASPECTOS GERAIS

Ao longo desse trabalho, foi apresentado o desenvolvimento e implementação de uma plataforma de controle de trajetória em linha reta para um manipulador robótico de 5 graus de liberdade. Nesse processo, foram desenvolvidas as diversas estruturas para que o manipulador tivesse condições de movimentação, bem como avaliadas suas formas de movimentação e as estratégias de movimentação seguindo a trajetória desejada.

Além disso, outro aspecto mostrado nesse projeto foi à implementação e validação de um sistema de controle robótico numa *FPGA*. Tal ideia aponta para uma série de implementações futuras, onde pode-se ganhar bastante em termos de tempo de processamento. Isso porque algumas das operações de cinemática direta e inversa e principalmente os cálculos envolvendo o Jacobiano, podem ser implementadas em *hardware*, gerando um aumento de performance, ao realizar diversas operações em paralelo, como as multiplicações e inversões de matrizes.

Foi desenvolvida, também, uma série de metodologias de testes que podem ser utilizadas para outros projetos, demandando apenas a mudanças das equações referentes ao robô especifico. Desse modo, é mais fácil analisar imperfeições dos modelos elaborados, gerando um aumento da qualidade do mesmo.

Concluindo, pode-se dizer que os principais objetivos do trabalho foram alcançados de maneira bastante satisfatória, uma vez que foram validados os aspectos teóricos envolvidos, bem como os modelos desenvolvidos – dinâmico e cinemático. Além disso, os algoritmos foram validados via *software* e no manipulador, confirmando sua correção e a capacidade de movimentação seguindo uma linha reta, como definido.

7.2 MODELAGEM DO MANIPULADOR

Inicialmente, desenvolveu-se o modelo do manipulador desejado. Para isso, foram determinados seus parâmetros e limitações de movimentação, para, então, determinarem-se sua cinemática direta e inversa. Em seguida, através de derivações das equações de cinemática inversa, foi desenvolvido o Jacobiano do manipulador. Através da concepção desses modelos matemáticos, foi possível controlar a posição, movimentação e velocidade do manipulador ao longo da trajetória desejada, de uma linha reta.

Essa modelagem mostrou-se correta em relação ao manipulador desenvolvido. Isso pode ser visto através dos testes realizados nos *software*s desenvolvidos ao longo do projeto, visando avaliar essa modelagem. Além disso, o fato de o manipulador ter sido capaz de seguir as trajetórias desejadas, respeitando as velocidades e restrições, atesta a assertividade da modelagem.

7.3 ALGORITMOS E SISTEMAS

Para a execução do projeto e implementação dos testes, foi necessária a definição de uma plataforma aonde desenvolver o controlador do manipulador, bem como as interfaces entre esse controlador e os atuadores dos motores das juntas. Logo, foi escolhida uma placa contendo uma *FPGA*, com capacidade de implementação dos algoritmos desejados. Foram feitas as configurações e implementações de estruturas de *hardware* visando o posicionamento inicial e o acionamento dos manipuladores durante a execução da trajetória desejada.

Desse modo, foi desenvolvido um projeto de *hardware* embarcado na *FPGA*, baseado no uso de um microprocessador nativo da *Altera*, o *NIOS II*, associado à uma série de periféricos, responsáveis por realizar a comunicação com outros elementos do projeto, temporizar ações, validar o processo, armazenar os dados do programa, etc. E, sobre esse processador, foi desenvolvido um *software* responsável por realizar as operações matemáticas referentes aos cálculos de posição e velocidade dos eixos do manipulador.

Sobre essa plataforma, foi desenvolvido o *software* responsável por controlar a movimentação do manipulador. Para tanto, inicialmente, o manipulador determinava as posições inicial e final de cada movimento, dentro da trajetória desejada e então a *FPGA* atuava sobre os motores, de modo a promover efetivamente a movimentação.

Foi possível validar esses cálculos através da utilização de ferramentas matemáticas, capazes de mostrar que as posições calculadas pelo planejador de trajetória realmente seguiam as características desejadas para a movimentação, determinando uma linha reta.

7.3.1 Movimentação do Manipulador

Já a atuação sobre os motores apresentou alguns problemas, inicialmente, quando implementado o *RTA*. Sua aplicação mostra um principio ideal de utilização da *FPGA* que

seria baseado no calculo em tempo real de cada iteração do manipulador, acionando os atuadores do manipulador. Contudo, essa aplicação não se mostrou satisfatória no manipulador, gerando perda de passos nos eixos 1 e 2 do manipulador. Tal fato deve-se ao alto tempo de resposta inicial desses eixos.

Desse modo, foi desenvolvido um segundo algoritmo que não sofria com essa limitação, mostrando-se mais eficiente, ao suavizar a movimentação e diminuir sensivelmente a perda de passos dos eixos.

Pôde-se, então, observar na prática que o manipulador é capaz de, a partir de um ponto inicial e final, desenvolver uma trajetória em linha reta entre eles, de maneira eficiente e respeitando as restrições de projeto, como velocidade, por exemplo.

A comparação entre os algoritmos mostra a perda da capacidade de calculo em tempo real em prol de uma movimentação que permitisse a implementação correta do modelo de trajetória desejada.

7.4 ESCOLHAS DE PROJETO

Ao longo do desenvolvimento desse trabalho, foram tomadas várias decisões que definiram o projeto e suas restrições. Entre essas, pode-se discutir a escolha dos motores, atuadores, juntas e formas de realização dos testes. Entretanto, no contexto do desenvolvimento do controlador aqui apontado, uma das questões que desponta refere-se à escolha da *FPGA* como plataforma de desenvolvimento.

Essa foi uma escolha baseada em algumas expectativas, mas também de algumas definições de projeto.

Inicialmente, a grande vantagem apresentada pela *FPGA* foi a obvia capacidade de adaptação do *hardware* sem a necessidade de mudanças de equipamentos. A partir disso, foi possível elaborar os sistemas de acionamento das juntas da forma desejada. Bem como também era possível a modificação do microprocessador, com a utilização de mais ou menos pinos de comunicação, I/O ou memória sem necessidade de troca do mesmo, bastando compilá-lo novamente.

Essa característica de modificações e testes traz ainda outra vantagem. A possibilidade de o *hardware* aqui desenvolvido servir de protótipo para a construção de um novo *hardware* dedicado, tendo exatamente as características aqui apontadas, diminuindo custos de produção e chances de erros de projeto.

Outra característica apontada como vantajosa foi a capacidade de implementação do controle distribuído dos atuadores. Ainda que seu cálculo de movimentação seja feito ao mesmo tempo, seu controle é feito por blocos isolados e independentes. Ainda que isso não seja uma vantagem tão grande, uma vez que o tempo de *pooling* entre as saídas não fosse tão alto, a ponto de gerar perda de dados, essa é uma implementação que pode ser usada em qualquer outro sistema.

Por último, pensando em expansão do projeto, a utilização de uma *FPGA* de grande capacidade permite que os sistemas que serão futuramente implementados no manipulador (controle de visão e geração de nuvem de pontos) possam ser implementados diretamente nela. Desse modo, pode-se ter um único *hardware* capaz de controlar todo o processo de movimentação do manipulador, desde a observação da superfície a ser reparada, passando pela geração de pontos de soldagem até a soldagem propriamente dita, a ser realizada pelo manipulador.

7.5 PROJETOS FUTUROS

Como comentado anteriormente, a elaboração do sistema de controle de movimentação do manipulador é apenas um dos módulos desse grande projeto, de reparação de turbinas danificadas. Desse modo, alguns projetos futuros são essenciais, tendo em vista a conclusão do conjunto da obra. E, com relação ao desenvolvimento das arquiteturas e do *hardware* utilizado, outras melhorias podem ser feitas ainda.

7.5.1 Melhorias físicas no manipulador

Visando um melhor posicionamento inicial do manipulador, é interessante a instalação de novos sensores de fim de curso nos seus eixos 1 e 2. Por meio dessa instalação e inserção desses dados no controle do manipulador, a repetibilidade do seu movimento será muito maior e mais garantida.

7.5.2 Integração dos Módulos

Tendo em vista a conclusão do sistema completo de reparação de turbinas, as seguintes atividades são sugeridas como próximos passos:

- Implementação do sistema de calibração dos parâmetros, permitindo, assim, a correta movimentação dos eixos do manipulador;
- Implementação de rotina para envio/recebimento de lista de posições, para a realização de uma soldagem completa;
- Definir uma malha fechada de controle, com a utilização dos *encoders*, que facilitará a implementação de diferentes trajetórias, além de aumentar a capacidade do sistema de tratar com folgas e pequenos erros de mecânica;
- Testes de trajetória circular, para realização de soldas de diferentes perfis, variando aspectos da referência a ser seguida;
- Projeto de fixação do manipulador à turbina.

7.5.3 Otimização dos Sistemas

Com relação aos sistemas implementados na *FPGA*, algumas otimizações podem ser levadas em consideração. Essas devem visar resolver alguns gargalos encontrados durante a execução desse projeto:

- **Rotinas de cálculos complexos executadas em *hardware***: a implementação das rotinas de cinemática inversa e direta em bem como do calculo do jacobiano em *hardware* aumentaria a velocidade das ações de controle sobre o manipulador, diminuindo o tempo de calculo para a movimentação do robô;
- **Implementação de mais de um microprocessador *NIOS***: como descrito no capitulo 5, a partir das posições inicial e final da trajetória, são calculados os pontos intermediários a serem percorridos, antes de a movimentação ser iniciada. Desse modo, a implementação de mais de um microprocessador vai otimizar esse processo, ao acelerar a determinação desses pontos, permitindo que o movimento inicie-se mais rapidamente;

BIBLIOGRAFIA

Altera. (Março de 2005). Handbook. *Quartus II Altera Handbook* .

Altera. (2011). *Stratix 3 Handbook.* San Jose.

Altera. (Setembro de 2003). Stratix PCI Development Board. *Datasheet* .

Altera. (2009). *www.altera.com*. Acesso em Março de 2009, disponível em Altera: www.altera.com

Asimo. (2009). Acesso em 2 de 10 de 2009, disponível em http://www.asimo.pl/dodatki/robotdef.pdf

Berger Lahr. (s.d.). Catalogue. *Stepper Motor Drives SD3* .

Brown, S. V. (2000). *Fundamentals of Digital Logic with vhdl Design.* Toronto: Mc Graw Hill.

Configware. (2005). Acesso em 5 de 10 de 2009, disponível em The Configware Page: http://configware.org

CONTO, A. M. (2003). *Proposta de um Teach Pendant para Robô de Eixo Redundante.* UFSC, Departamento de automação e sistemas. Florianopolis: Relatório.

Danaher Motion. (2004). SLO-SYN MD808 & MD808-128 Microstep Drive Module. *Installation Instructions* . United States.

Directed Perception. (s.d.). Pan-Tilt Unit - Model PTU-D46. *User's Manual* .

Eclipse. (2009). *Eclipse*. Acesso em 10 de 08 de 2009, disponível em eclipse.org

Euron. (2009). *www.euron.org*. Acesso em 10 de 8 de 2009, disponível em EURON: www.euron.org

Forest, A. R. (1969). Coordinates, Transformations and Visualization Techniques. *CAD Group Document No. 23* .

Hamblen, J. F. (2001). *Rapid Prototyping of Digital System.* Kluwer Academic Publishers.

Harteinsten, R. (2000). Configware / Software Co-Design: Be Prepared for the Next Revolution! *ACM CF04* , pp. 357-362.

Hartenstein, R. (2006). Morphware and Configware. In: A. Zomaya, *Handbook of Innovative Computing Paradigms.* Springer Verlag, New York,.

Hartenstein, R. (1 de Março de 2006). Why we need Reconfigurable Computing Education. *opening session of the 1st International Workshop on Reconfigurable Computing Education* . Karlsruhe, Germany.

Hauck, S. (Abril de 1998). The Roles of FPGAs in Reprogrammable Systems. *Proceedings of the IEEE* , p. 86/4.

Hutton, M. Y. (2006). A Methodology for FPGA to Structured-ASIC Synthesis and Verification. *Proceedings of the Conference of Design, Automation and Test in Europe* , pp. pp. 64-69.

IFR. (2009). Acesso em 8 de 9 de 2009, disponível em International Federation of Robotics: http://www.ifr.org/

Jimenez, C., & Almeida, J. (1998). O Brasil na era dos robôs. *Epoca* .

Jun, K. H., Seok, A. H., Min, B. Y., & Kyu, S. I. (2006). Black Box for Robot Manipulation. *Nios II Embedded Processor Design Contest—Outstanding Designs 2006* (pp. 285-294). Altera.

KAPP, W. (2000). Proposta da Fase II: Engenharia de Aplicação e Processos de Preparação e Acabamento Superficial. *Relatório projeto Roboturb LACTEC e Universidade Federal de Santa Catarina* . Santa Catarina.

Kung, Y.-S., & Shu, G.-S. (2005). Development of a FPGA-based Motion Control IC for Robot Arm. *IEEE* , pp. 1397-1402.

LEAL, R. D. (Março de 2005). IMPACTOS SOCIAIS E ECONÔMICOS DA ROBOTIZAÇÃO: ESTUDO DE CASO DO PROJETO ROBOTURB. *Dissertação de Mestrado* . Frorianopolis, SC, Brasil.

Lintech. (s.d.). Lintech Positioning System. *300 and 400 series* .

McKerrow, P. J. (1993). *Introduction to Robotics.* Addison Wesley.

Moore, G. (1997). The Microprocessor: Engine of the Technology Revolution. *Communications of the AMC, Vol. 40 (2)* , pp. 112-114.

Motta, J. (2001). Robot Calibration using a 3D Vision-based. *Robotics and Computer Integrated Manufacturing,* , Vol 17, n 6; pp 487 - 497.

MUÑOZ, D. M. (2006). *Implementação e Simulação de Algoritmos de Escalonamento para Sistemas de Elevadores Usando Arquiteturas Reconfiguráveis - Dissertação de Mestrado em Sistemas Mecatrônicos,.* Brasilia: Departamento de Engenharia Mecânica, Universidade de Brasília.

Paul, R. P. (1989). *Robot manipulators : Mathematics, programming, and control: the computer control of robot manipulators.* Cambridge: Mit Press.

Rammig, F. (s.d.). Visions from the IT Engine Room. (I. T.-C. Technology, Entrevistador)

Reis Robotics. (2009). *Reis Robotics*. Acesso em 10 de Junho de 2009, disponível em Reis robotics: http://www.reisrobotics.de/reisrobotics_media/Bilder/01_bilder/rl300p_kl.jpeg

Schilling, R. J. (1990). *Fundamentals of Robotics: Analysis and Control.* New Jersey: Prentice Hall.

Schlussel, K. (30 de 3 de 1985). Robotics and Artificial Intelligence Across the Atlantic and Pacific. *IEEE Transactions on Industrial Electronics* , pp. 244-251.

Silva, J. Y. (2010). *Implementação de Técnicas de processamento de Imagens no Dominio Espacial em Sistemas Reconfiguráveis.* Brasília.

Slabaugh, G. G. (1999). *Computing Euler angles from a rotation matrix(denoted as TRTA implementation from: http://www.starfireresearch.com/services/java3d/samplecode/FlorinEulers.html.).*

Soska, G. (1985). Third Generation Robots: Their Definition, Characteristics, and Applications. *Robotics Age 7* , 14-16.

Spong, M. W. (2006). *Robot Modeling and Control.* Wiley.

Sun, X. S., & Mills, J. K. (2006). A New Motion Control Hardware Architecture with FPGA-Based IC Design for Robotic Manipulators. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, (pp. 3520-3525). Orlando - Flórida.

Taghavi, T. G. (2004). Innovative or Perish: FPGA Physical Design. *ACM* , pp. ISDP04, USA.

Tartari Filho, S. C. (2006). Modelagem e Otimização de um Robô de Arquitetura Paralela Para Aplicações Industriais. *Dissertação de Mestrado* . São Paulo: Escola Politécnica da Universidade de São Paulo.

Terasic. (2009). *DE3 User Manual.*

Terasic. (Janeiro de 2012). *terasic*. Acesso em Janeiro de 2012, disponível em terasic: www.terasic.com

The Flowware Page. (2001). Acesso em 5 de 10 de 2009, disponível em Flowware: http://flowware.net/

The Kress-Kung Machine Page. (2001). Acesso em 10 de 9 de 2009, disponível em Anti Machine Page: http://anti-machine.org/

VASCONCELOS FILHO, E. F., ROZENWALD, G., & QUINTEROS, H. (2008). An Environment for Robot Soccer Game Simulation using Reconfigurable Architectures Based on a FPGA-PCI Board. In: *ABCM Symposium Series In Mechatronics, Vol. 3 - Section VII - Emerging Technologies and Applications.* (pp. p. 726-735). Rio de Janeiro: Brazilian Society of Mechanical Sciences and Engineering.

Vasconcelos Filho, Ê. P., Ginani, L., Motta, J. M., Caribe, G., Llanos, C., & Britto, W. V. (Abril de 2009). DESENVOLVIMENTO DO MODELO CINEMÁTICO DE CALIBRAÇÃO DE UM ROBÔ DE GEOMETRIA ESFÉRICA PARA SOLDAGEM GMAW POSICIONAL MULTICAMADAS. *COBEF* . Belo Horizonte, MG, Brasil.

Vu, H., & Esfandiari, R. (1998). *Dynamic systems: modeling and analysis.* New York: McGraw-Hill.

Vu, H., & Esfandiari, R. (1998). *Dynamic systems: modeling and analysis.* New York: McGraw-Hill.

Yili, Z., Hanxu, S., Qingxuan, J., & Guozhen, S. (2008). Kinematics Control for a 6-DOF Space Manipulator Based on ARM Processor and FPGA Co-processor. *The IEEE International Conference on Industrial Informatics (INDIN 2008)*, (pp. 129-134). Daejeon, Korea.

ANEXOS

ANEXO A – CÓDIGO FONTE

Aqui, encontra-se o código fonte do *software* embarcado no *NIOS II*

```
//Endereço de saida padrao UART_BASE
//Includes
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "sys/alt_timestamp.h"
//#include <time.h>
#include "alt_types.h"
#include "system.h"
#include <io.h>
//#include "altera_avalon_uart_regs.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_log_printf.h"
#include <math.h>
#include <sys/alt_timestamp.h>
#include <sys/time.h>


//Defines
#define VAR_JUNTAS  5
#define VAR_CART    6
#define pi          3.141592654f
#define limiar      0.00001f
#define TSTRING     11
#define TAM_INST    4
#define TAM_PAL     24
#define TERMINAL    1
#define RS232       0

//#define vel_max     5       //mm/s
#define vel_max     10      //mm/s
//#define int_tempo   0.05f  //s
#define int_tempo   0.5f  //s
#define FREQ_ORIG   100000

////Constantes de seno e cosseno
#define c1 cos(var_juntas[0])
#define s1 sin(var_juntas[0])
#define c2 cos(var_juntas[1])
#define s2 sin(var_juntas[1])
#define c4 cos(var_juntas[3])
#define s4 sin(var_juntas[3])
#define c5 cos(var_juntas[4])
#define s5 sin(var_juntas[4])

#define d3 var_juntas[2]

#define cr cos(var_cart[3])
#define sr sin(var_cart[3])
#define cp cos(var_cart[4])
#define sp sin(var_cart[4])
#define cy cos(var_cart[5])
#define sy sin(var_cart[5])

//Parametros do Robô

#define px var_cart[0]
```

```
#define py var_cart[1]
#define pz var_cart[2]

#define xx    (cr*cp)
#define yx    (-sr*cy+cr*sp*sy)
#define zx    (sr*sy+cr*sp*cy)
#define xy    (sr*cp)
#define yy    (cr*cy+sr*sp*sy)
#define zy    (-cr*sy+sr*sp*cy)
#define xz    (-sp)
#define yz    (cp*sy)
#define zz    (cp*cy)
#define k41    0
#define k42    0
#define k43    0
#define k44    1

#define t11    ((-c1*s2*c4-s1*s4)*c5-c1*c2*s5)
#define t12    (-(-c1*s2*c4-s1*s4)*s5-c1*c2*c5)
#define t13    (-c1*s2*s4+s1*c4)
#define t14    (-c1*c2*d3-c1*L2*s2+s1*D2)
#define t21    ((-s1*s2*c4+c1*s4)*c5-s1*c2*s5)
#define t22    (-(-s1*s2*c4+c1*s4)*s5-s1*c2*c5)
#define t23    (-s1*s2*s4-c1*c4)
#define t24    (-s1*c2*d3-s1*L2*s2-c1*D2)
#define t31    (c2*c4*c5-s2*s5)
#define t32    (-c2*c4*s5-s2*c5)
#define t33    (c2*s4)
#define t34    (-s2*d3+L2*c2+D1)
#define t41    0
#define t42    0
#define t43    0
#define t44    1

//Var Globais
double matriz_t[4][4];//matriz rTh
double matriz_k[4][4];//matriz genérica

double var_juntas[VAR_JUNTAS];//variáveis cartesianas e polares
    int iVar_juntas[VAR_JUNTAS];
    char sVar_juntas[VAR_JUNTAS][TSTRING];

double var_cart[VAR_CART];
    int iVar_cart[VAR_CART];
    char sVar_cart[VAR_CART][TSTRING];

int     dir_out[VAR_JUNTAS][1000];
int     passos_out[VAR_JUNTAS][1000];
double  div_freq_out[VAR_JUNTAS][1000];

double conta_passos=0;
double passos[5];

double GRAU_RAD, RAD_GRAU;
double D2,D1,L2;
char dado[15];

float res_pulso[3];                                  //graus/pulso   -->
360/res_total

//Prototipos de funcoes
```

```
//Funções de Impressão/Leitura
int imprime(float fEnt,int saida,char *buffer);
int imprime_rs232(char *imp);
int le_pc(int );

//Funções de controle do motor
int define_velocidade();
int comunica_motores();

//Funções de Calculo
int calcula_matriz();
int calcula_var_cart();
int calcula_angulos();

//Funções de controle dos motores
void controla_movimentacao(double *var_cart_inicio, double
*var_cart_fim);
int movimento();
int comanda_motores(int cont, int *passos_total);
int calcula_comandos_motores(double *vel_juntas,double *dist,int cont);
int posiciona(double* pos_in_P,double *pos_fn_P);

//funções estruturais
int inicia_variaveis();
int principal();
int parametros_motores();
int posicao_inicial();

double calcula_vel_0(double *);
double zera_manipulador();

int sw=0;
int sw_new=1;

//-------------------------------------------------------------------------
---------------
//Projeto principal:
//Fica aguardando a ordem vinda dos outros programas.
//-------------------------------------------------------------------------
---------------
int main(){
    sw_new=IORD_ALTERA_AVALON_PIO_DATA(REPETE_BASE);
    sw=!IORD_ALTERA_AVALON_PIO_DATA(REPETE_BASE);
    printf("Inicio do Software! %d\n",sw);
    while(1){
        if(sw_new!=sw){
            IOWR_ALTERA_AVALON_PIO_DATA(ZERA_POS_BASE,0x001);
            //--------------------------------------
            //Desab. Motores
            //--------------------------------------
            IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
            IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
            IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);
            //--------------------------------------
            //Iniciando variaveis de controle
            //--------------------------------------
            inicia_variaveis();
            //--------------------------------------
            //Posicionando o manipulador na posição "Zero"
            //--------------------------------------
            posicao_inicial();
```

```
            //--------------------------------------
            //Controlando o movimento do manipulador
            //--------------------------------------
            movimento();
            sw=sw_new;
            conta_passos=0;
            //--------------------------------------
            //Desab. Motores
            //--------------------------------------
            IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
            IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
            IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);
        }else{
            sw=sw_new;
        }
        sw_new=IORD_ALTERA_AVALON_PIO_DATA(REPETE_BASE);
    }
}//fim do main
//------------------------------------------------------------------------
-------
//int inicia_variaveis()
//Função que zera os valores de variaveis cartesianas e polares
//------------------------------------------------------------------------
-------
int inicia_variaveis(){
    int i;
    for (i=0;i<VAR_CART;i++){
       var_cart[i]=0;
    }
    for (i=0;i<VAR_JUNTAS;i++){
       var_juntas[i]=0;
    }

    D1=110.0f;
    D2=-100.0f;
    L2=30.0f;

    GRAU_RAD =pi/180;
    RAD_GRAU =180/pi;

    var_juntas[0]=21.8014f*GRAU_RAD;
    var_juntas[1]=0.0f*GRAU_RAD;
    var_juntas[2]=498.5164808f;
    var_juntas[3]=45.0f*GRAU_RAD;
    var_juntas[4]=45.0f*GRAU_RAD;

    calcula_matriz();
    calcula_var_cart();
    parametros_motores();

    return 0;
}

//------------------------------------------------------------------------
-------
//int calcula_matriz(){
//Função responsável por calcular as matrizes e associa-las com seus
valores
//definidos;
//------------------------------------------------------------------------
-------
```

```
int calcula_matriz(){

    matriz_t[0][0]=t11;
    matriz_t[0][1]=t12;
    matriz_t[0][2]=t13;
    matriz_t[0][3]=t14;
    matriz_t[1][0]=t21;
    matriz_t[1][1]=t22;
    matriz_t[1][2]=t23;
    matriz_t[1][3]=t24;
    matriz_t[2][0]=t31;
    matriz_t[2][1]=t32;
    matriz_t[2][2]=t33;
    matriz_t[2][3]=t34;
    matriz_t[3][0]=t41;
    matriz_t[3][1]=t42;
    matriz_t[3][2]=t43;
    matriz_t[3][3]=t44;

    return 0;
}
//-------------------------------------------------------------------------
-------
//int calcula_var_cart()
//Função principal responsável pela determinação da cinemática direta do
sistema
//-------------------------------------------------------------------------
-------
int calcula_var_cart(){

    double val1;
    double val2;
    //---------------------------------------
    //Definindo os valores de Px, Py e Pz
    //---------------------------------------
    var_cart[0]= matriz_t[0][3];//calcula Px
    var_cart[1]= matriz_t[1][3];//calcula Py
    var_cart[2]= matriz_t[2][3];//calcula Pz

    if(fabs(t31)<=0.999999){//se não for singular
        var_cart[3]= atan2(matriz_t[1][0],matriz_t[0][0]);
        val1=matriz_t[0][0]*matriz_t[0][0];
        val2=matriz_t[1][0]*matriz_t[1][0];
        var_cart[4]= atan2(-matriz_t[2][0],sqrt(val1+val2));
        var_cart[5]= atan2(matriz_t[2][1],matriz_t[2][2]);
    }else{//se for singular
        //Setando o angulo RZ para Zero
        var_cart[3] = 0;
        if(t31 < 0)
        {
            var_cart[4] =  M_PI_2;
            var_cart[5] =  atan2(t12,t13);
        }else{
            var_cart[4] = -M_PI_2;
            var_cart[5] =  M_PI + atan2(t12,t13);
        }
    }
    return 0;
}
```

```
//-------------------------------------------------------------------------
-------
//int parametros_motores()
//Função que determina parametros dos motores
//-------------------------------------------------------------------------
-------
int parametros_motores(){

    float reducao[3];
    float resolucao[3];
    float res_total[3];                                  //passos/rev    --
> reducao*resolucao

//Redução acoplada ao motor
    reducao[0]=90.0f;
    reducao[1]=90.0f;
    reducao[2]=1.0f;

//Resolução do motor (passos por revolução)
    resolucao[0]=5000.0f;
    resolucao[1]=5000.0f;
    resolucao[2]=100.0f;   // modificado em 15/11/2011, para arranjo de
resolução do motor.
                           //Valor antigo: 1000.0 -- Valor Novo 100.0

//Motor1
    res_total[0]=reducao[0]*resolucao[0];
    res_pulso[0] = (360.0f/res_total[0])*GRAU_RAD;
    //res_pulso[0] = (360.0f/res_total[0]);

//Motor2
    res_total[1]=reducao[1]*resolucao[1];
    res_pulso[1] = (360.0f/res_total[1])*GRAU_RAD;
    //res_pulso[1] = (360.0f/res_total[1]);

//Motor3
    res_total[2]=reducao[2]*resolucao[2];
    res_pulso[2] = 1.0f/res_total[2];

    return 0;
}

//-------------------------------------------------------------------------
-------
//int posicao_inicial()
//Função que zera a posição do manipulador, no inicio do movimento
//-------------------------------------------------------------------------
-------
int posicao_inicial(){
    printf("Pos1: %d\n",IORD_ALTERA_AVALON_PIO_DATA(POS_1_BASE));
    printf("Pos2: %d\n",IORD_ALTERA_AVALON_PIO_DATA(POS_2_BASE));
    printf("Pos3: %d\n",IORD_ALTERA_AVALON_PIO_DATA(POS_3_BASE));
    //-----------------------------------------------
    //Conferindo se Manipulador está na posição 0
    //-----------------------------------------------
    while(!IORD_ALTERA_AVALON_PIO_DATA(POS_1_BASE) ||
          !IORD_ALTERA_AVALON_PIO_DATA(POS_2_BASE) ||
          !IORD_ALTERA_AVALON_PIO_DATA(POS_3_BASE)){
       //Função necessária por questões de temporização
       IOWR_ALTERA_AVALON_PIO_DATA(ZERA_POS_BASE,0x000);
       //printf("Waiting pos 0\n");
```

```
        //printf("Pos1: %d\n",IORD_ALTERA_AVALON_PIO_DATA(POS_1_BASE));
        //printf("Pos2: %d\n",IORD_ALTERA_AVALON_PIO_DATA(POS_2_BASE));
        //printf("Pos3: %d\n",IORD_ALTERA_AVALON_PIO_DATA(POS_3_BASE));
        usleep(10000);
    }
    IOWR_ALTERA_AVALON_PIO_DATA(ZERA_POS_BASE,0x001);
    printf("initial Position - OK\n");
    //-----------------------------------------------------
    //Zerando os contadores
    //-----------------------------------------------------
    var_juntas[0]=0.0f*GRAU_RAD;
    var_juntas[1]=0.0f*GRAU_RAD;
    //var_juntas[2]=150.0f;                                  //modificado
de 290.0 para 150 em função do novo Posicionamento - 14/11;
    var_juntas[2]=330.0f;
    var_juntas[3]=45.0f*GRAU_RAD;
    var_juntas[4]=45.0f*GRAU_RAD;

    calcula_matriz();
    calcula_var_cart();
return 0;
}
//---------------------------------------------------------------------------
-------
//int movimento()
//Função que recebe, do mundo externo, as posições inicial e final do
movimento
//---------------------------------------------------------------------------
-------
int movimento(){

/*
    //-----------------------------------------------------
    //Fazer quadrado no plano
    //-----------------------------------------------------
    //0;0;330;45;45
    double pos_in_P[6]={-330.0f,100.0f,140.0f,144.735586f*GRAU_RAD,-
30.0f*GRAU_RAD,-35.264388f*GRAU_RAD};
    //double pos_fn_P[6]={-400.000f,-
250.0000f,140.0f,144.735586f*GRAU_RAD,-30.0f*GRAU_RAD,-
35.264335f*GRAU_RAD};
    //20;0;480;45;45
    //double pos_fn_P[6]={-485.254472f,-
70.200405f,140.0f,164.735586f*GRAU_RAD,-30.0f*GRAU_RAD,-
35.264335f*GRAU_RAD};
    double pos_fn_P[6]={-400.000f,100.00f,140.0f,164.735586f*GRAU_RAD,-
30.0f*GRAU_RAD,-35.264335f*GRAU_RAD};
    double pos_fn_P2[6]={-600.000f,100.00f,140.0f,164.735586f*GRAU_RAD,-
30.0f*GRAU_RAD,-35.264335f*GRAU_RAD};
    double pos_fn_P3[6]={-600.000f,-100.00f,140.0f,164.735586f*GRAU_RAD,-
30.0f*GRAU_RAD,-35.264335f*GRAU_RAD};
    double pos_fn_P4[6]={-400.000f,-100.00f,140.0f,164.735586f*GRAU_RAD,-
30.0f*GRAU_RAD,-35.264335f*GRAU_RAD};

    posiciona(pos_in_P,pos_fn_P);
    controla_movimentacao(pos_in_P,pos_fn_P);  //Indo para P1
    controla_movimentacao(pos_fn_P,pos_fn_P2); //P1 - P2
    controla_movimentacao(pos_fn_P2,pos_fn_P3);//P2 - P3
    controla_movimentacao(pos_fn_P3,pos_fn_P4);//P3 - P4
    controla_movimentacao(pos_fn_P4,pos_fn_P); //P4 - P1
*/
```

```
    //-----------------------------------------------------
    //Fazer reta no espaço
    //-----------------------------------------------------

    double pos_in_P[6]={-330.0f,100.0f,140.0f,144.735586f*GRAU_RAD,-
30.0f*GRAU_RAD,-35.264388f*GRAU_RAD};
    double pos_fn_P[6]={-420.0f,100.0f,140.0f,144.735586f*GRAU_RAD,-
30.0f*GRAU_RAD,-35.264388f*GRAU_RAD};
    //double pos_fn_P3[6]={-
610.780028f,100.0f,220.668584f,140.758813f*GRAU_RAD,-37.7765f*GRAU_RAD,-
28.154788f*GRAU_RAD};
    double pos_fn_PP[6]={-
638.813515f,214.182719f,227.979051f,140.758813f*GRAU_RAD,-
37.7765f*GRAU_RAD,-28.154788f*GRAU_RAD};
    posiciona(pos_in_P,pos_fn_P);
    controla_movimentacao(pos_in_P,pos_fn_P);
    controla_movimentacao(pos_fn_P,pos_fn_PP);
    return 0;
}

/*-------------------------------------------------------------------------
-------
int posiciona()
Função que verifica a posição atual do manipulador em relação a posição
desejada para o inicio da trajetória controlada. Comanda o movimento do
manipulador;
-------------------------------------------------------------------------
-----*/
int posiciona(double *pos_in_P,double *pos_fn_P){
    double var_juntas_temp [5];
    int i;
    int flag_p[6];
    double dist_Q[5];

    //-----------------------------------------------------
    //Zerando os contadores
    //-----------------------------------------------------
    var_juntas[0]=0.0f*GRAU_RAD;
    var_juntas[1]=0.0f*GRAU_RAD;
    //var_juntas[2]=150.0f;
    var_juntas[2]=330.0f;
    var_juntas[3]=45.0f*GRAU_RAD;
    var_juntas[4]=45.0f*GRAU_RAD;

    calcula_matriz();
    calcula_var_cart();

    printf("Moving to P1!\n");
    //-----------------------------------------------------
    //copiando a posição atual para variáveis temporárias
    //-----------------------------------------------------
    for (i=0;i<=4;i++){
        var_juntas_temp[i]=var_juntas[i];
        flag_p[i]=0;
    }
    //-----------------------------------------------------
    //Calculando a posição angular desejada
    //-----------------------------------------------------
    for (i=0;i<=5;i++){
        var_cart[i]=pos_in_P[i];
    }
```

```
    calcula_matriz();
    calcula_angulos();
    //var_juntas[0]=10.07599*GRAU_RAD;
    //var_juntas[1]=7.5536*GRAU_RAD;
    //-----------------------------------------------------
    //Calculando a distancia angular entre o Pdesejado e o Patual
    //-----------------------------------------------------
    for (i=0;i<=4;i++){
        dist_Q[i]= var_juntas[i]-var_juntas_temp[i];
    }
    //-----------------------------------------------------
    //Determinando que será contado o número de passos
    //-----------------------------------------------------
    IOWR_ALTERA_AVALON_PIO_DATA(PASSOS_BASE,0x001);
    //-----------------------------------------------------
    //Calculando o numero de passos a serem dados
    //-----------------------------------------------------
    if (dist_Q[0]<0.0001f && dist_Q[0]>-0.0001f){
        IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M1_BASE,0x000);
    } else{
        if(dist_Q[0]<0.00){
            IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M1_BASE,round(-
dist_Q[0]/res_pulso[0]));
            IOWR_ALTERA_AVALON_PIO_DATA(DIR_M1_BASE,0x001);//sentido
negativo  (anti-horario)
        }else{

IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M1_BASE,round(dist_Q[0]/res_pulso[0]));
            IOWR_ALTERA_AVALON_PIO_DATA(DIR_M1_BASE,0x000);//sentido
positivo  (horario)
        }
    }
    if (dist_Q[1]<0.0001f && dist_Q[1]>-0.0001f){
        IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M2_BASE,0x000);
    } else{
        if (dist_Q[1]<0.00){
            IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M2_BASE,round(-
dist_Q[1]/res_pulso[1]));
            IOWR_ALTERA_AVALON_PIO_DATA(DIR_M2_BASE,0x000);//sentido
negativo  (cima)
        }else{

IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M2_BASE,round(dist_Q[1]/res_pulso[1]));
            IOWR_ALTERA_AVALON_PIO_DATA(DIR_M2_BASE,0x001);//sentido
positivo  (baixo)
        }
    }
    if (dist_Q[2]<0.0001f && dist_Q[2]>-0.0001f){
        IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M3_BASE,0x000);
    } else{
        if(dist_Q[2]<0.00){
            IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M3_BASE,round(-
dist_Q[2]/res_pulso[2]));
            IOWR_ALTERA_AVALON_PIO_DATA(DIR_M3_BASE,0x000);//sentido
negativo  (tras)
        }else{

IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M3_BASE,round(dist_Q[2]/res_pulso[2]));
            IOWR_ALTERA_AVALON_PIO_DATA(DIR_M3_BASE,0x001);//sentido
positivo  (frente)
        }
```

```
    }
    passos[0]=round(dist_Q[0]/res_pulso[0]);
    passos[1]=round(dist_Q[1]/res_pulso[1]);
    passos[2]=round(dist_Q[2]/res_pulso[2]);

    //-----------------------------------------------------
    //setando as frequencias de movimentação
    //-----------------------------------------------------
    IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M1_BASE,50);
    IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M2_BASE,50);
    IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M3_BASE,50);
    //-----------------------------------------------------
    //Movimentação
    //-----------------------------------------------------
    usleep(10000);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x000);
    usleep(10000);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x000);
    usleep(10000);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x000);
    usleep(10000);

    while(flag_p[2]==0 || flag_p[1]==0 || flag_p[0]==0){
        //printf("While...!\n");
        if (dist_Q[0]<0.0001f && dist_Q[0]>-0.0001f){
            flag_p[0]=1;
        } else{
            flag_p[0]=IORD_ALTERA_AVALON_PIO_DATA(FIM_M1_BASE);
        }
        usleep(10000);
        if (dist_Q[1]<0.0001f && dist_Q[1]>-0.0001f){
            flag_p[1]=1;
        } else{
            flag_p[1]=IORD_ALTERA_AVALON_PIO_DATA(FIM_M2_BASE);
        }
        usleep(10000);
        if (dist_Q[2]<0.0001f && dist_Q[2]>-0.0001f){
            flag_p[2]=1;
        } else{
            flag_p[2]=IORD_ALTERA_AVALON_PIO_DATA(FIM_M3_BASE);
        }
        usleep(10000);
    }

    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);

    //-----------------------------------------------------------------------
---
    //após a movimentação, determina a posição correta do manipulador
    //-----------------------------------------------------------------------
---
    for(i=0;i<=2;i++){
        var_juntas[i]=var_juntas_temp[i]+passos[i]*res_pulso[i];
    }
    //-----------------------------------------------------------------------
---
    printf("Terminou!\n");
    return 0;
}
```

```
//-------------------------------------------------------------------------
-------
//int calcula_angulos()
//Função principal na de terminação da cinemática inversa
//A partir dos valores das coordenadas cartesianas, define os valores das
//coordenadas de juntas
//-------------------------------------------------------------------------
-------
int calcula_angulos(){
   // Calculando as Variaveis de Juntas

   double k1, k2, k3, k4;
   double C1,S1;
   double C2,S2;
//   double D3;
   double C4,S4;
   double C5,S5;
   double Raiz;

   //////////////
   // Theta 2  //
   //////////////
   k1       = px*px + py*py - D2*D2 + (D1-pz)*(D1-pz);
   k2       = 2*L2*(D1-pz);
   k3       = px*px + py*py - D2*D2 - L2*L2;
   k4       = k2*k2 + 4*k1*k3;
   if(k4<=0){k4=0.0f;}
   Raiz     = sqrt(k4);

   C2       = (-k2 + Raiz)/(2*k1);

   k4=px*px + py*py - D2*D2;
   if(k4<=0){k4=0.0f;}
   Raiz     = sqrt(k4);
   S2       = (L2 + C2*(D1-pz))/Raiz;

   var_juntas[1] = atan2(S2,C2);

   //////////////
   // Delta 3  //
   //////////////

   // S2 != 0
   if(fabs(S2)>limiar){
      var_juntas[2] = (C2*L2 + D1 - pz)/S2;
   }else{// S2  = 0
      k4=px*px + py*py - D2*D2;
      if(k4<=0){k4=0.0f;}
      var_juntas[2] = sqrt(k4)/C2;
   }
   ////////////////////////////////////////
   // Verificacao de Theta 2 || Delta 3 //
   ////////////////////////////////////////
   if(var_juntas[2] < 0)
   {
      // Recalculando Theta 2
      k4       = k2*k2 + 4*k1*k3;
      if(k4<=0){k4=0.0f;}
      Raiz=sqrt(k4);
      C2       = (-k2 - Raiz)/(2*k1);
```

```
    k4       = px*px + py*py - D2*D2;
    if(k4<=0){k4=0.0f;}
    Raiz     = sqrt(k4);
    S2       = (L2 + C2*(D1-pz))/Raiz;
    var_juntas[1] = atan2(S2,C2);
    // Recalculando Delta 3
    // S2 != 0
    if(fabs(S2)>limiar){
        var_juntas[2] = (C2*L2 + D1 - pz)/S2;
    }else{      // S2  = 0
        k4       = px*px + py*py - D2*D2;
        if(k4<=0){k4=0.0f;}
        var_juntas[2] = sqrt(k4)/C2;
    }
}
//////////////
// Theta 1  //
//////////////
k1        = (C2*var_juntas[2] + S2*L2);
k2        = (C2*var_juntas[2] + S2*L2)*(C2*var_juntas[2] + S2*L2);

C1        = -(px*k1 + py*D2)/(D2*D2 + k2);
S1        =  (px + C1*k1)/(D2);

var_juntas[0] = atan2(S1,C1);
//////////////
// Theta 4  //
//////////////
// S2 != 0
if(fabs(S2)>limiar){
    S4    = -(S1*zy + C1*zx)/S2;
}else{// S2  = 0
    S4 =  zz/C2;
}
// C1 != 0
if(fabs(C1)>limiar){
    C4     = -(zy + S1*S2*S4)/C1;
}else{// C1  = 0
    C4     = zx/S1;
}
var_juntas[3] = atan2(S4,C4);
//////////////
// Theta 5  //
//////////////
// S4 != 0
if(fabs(S4)>limiar){
    S5 =  (S1*yx - C1*yy)/S4;
}else{// S4  = 0
    S5 = -(xz*S2 + C2*C4*yz);
}
// S2 != 0
if(fabs(S2)>limiar){
    C5 = -(yz + C2*C4*S5)/S2;
}else{// S2  = 0 & S1 != 0
    if(fabs(S1)>limiar){
        C5 =  -(yy + C1*S4*S5)/(S1*C2);
    }else{// S2  = 0 & S1  = 0
        C5 =  -yx/(C1*C2);
    }
}
var_juntas[4] = atan2(S5,C5);
```

```
//   printf("\nANGULOS: \n");
//   printf("Q1 - %f\n",var_juntas[0]*RAD_GRAU);
//   printf("Q2 - %f\n",var_juntas[1]*RAD_GRAU);//theta2
//   printf("D3 - %f\n",var_juntas[2]);//d3
//   printf("Q4 - %f\n",var_juntas[3]*RAD_GRAU);
//   printf("Q5 - %f\n\n",var_juntas[4]*RAD_GRAU);

   return 0;
}

/*-------------------------------------------------------------------------
-------
int controla_movimentacao()
Função que determina movimento do manipulador
Implementando mudanças, onde é calculada, realmente, a velocidade do
sistema.
-------------------------------------------------------------------------
-----*/
void controla_movimentacao(double *var_cart_inicio, double
*var_cart_fim){

    double dist_atual;
    double dist_ant;

    double diferenca_P[6];
    double diferenca_Q[5];
    double var_juntas_temp[5];
    double vel_cart[6];
    double vel_juntas[5];

    double C1,C2,C4;
    double S1,S2,S4;
    double D3;
    double L2_Q,D2_Q,C4_Q,S4_Q,D3_Q,C2_Q,S2_Q;
    double C2_D3,S2_D3,C4_S4,L2_S2,C1_S2_S4,C2_S2_D3;

    int cont=0;
    double T;
    double z;
    int i;
    int j;
    double divisor;
    int passos_total[VAR_JUNTAS];

    for(i=0;i<5;i++){
        vel_juntas[i]=0.0f;
    }
    for (i=0;i<=2;i++){
       passos_total[i]=0;
    }


    //-------------------------------------------------------------------
---
    //Atribuindo os valores da posição atual para calculo da cinemática
inversa!
    //-------------------------------------------------------------------
---
    for(i=0;i<=5;i++){
        var_cart[i]=var_cart_inicio[i];
```

```
    }
    //-------------------------------------------------------------------
---
    //Calculando a diferença entre a posição inicial e a final em cada
eixo
    //-------------------------------------------------------------------
---
    for (i=0;i<=5;i++){
        diferenca_P[i]=var_cart_fim[i]-var_cart[i];
    }
    //-------------------------------------------------------------------
---
    //calculando a distancia atual
    //-------------------------------------------------------------------
---

dist_atual=sqrt(diferenca_P[0]*diferenca_P[0]+diferenca_P[1]*diferenca_P[
1]+diferenca_P[2]*diferenca_P[2]);
    //-------------------------------------------------------------------
---
    //enquanto a diferença não for muito proxima do limiar, continua o
processo
    //-------------------------------------------------------------------
---
    printf("Aki1!!! dist aual: %f \n",dist_atual);
    //while(dist_atual>0.001){
    while(dist_atual>1.0){

//printf("\ndist_Atual: %f\n",dist_atual);
       // getch();
        //-------------------------------------------------------------------
-------
        //Calculando o tempo restante até o fim do movimento da tocha
        //-------------------------------------------------------------------
-------
        T=dist_atual/vel_max;
        //printf("Tempo: %f \n",T);

        //-------------------------------------------------------------------
-------
        //Calculando a velocidade de cada eixo cartesiano
        //-------------------------------------------------------------------
-------

        for(i=0;i<=5;i++){
            vel_cart[i]=diferenca_P[i]/T;
            //vel_cart[i]=diferenca_P[i]/int_tempo;
            //printf("vel_cart[%d]: %f \n",i,vel_cart[i]);
        }
      /*
        vel_cart[0]=-3.90f;
        vel_cart[1]=-3.0f;
        vel_cart[2]=0.0f;
        vel_cart[3]=0.0f;
        vel_cart[4]=0.00f;
        vel_cart[5]=0.00f;
       */
        //printf("vel_cart[%d]: %f \n",0,vel_cart[0]);
        //printf("vel_cart[%d]: %f \n",1,vel_cart[1]);
        //-------------------------------------------------------------------
-------
```

```
        //Calculando a cinemática Inversa!
        //-------------------------------------------------------------------
-------
        calcula_matriz();
        calcula_angulos();
    //printf("var_juntas[0]: %f \n",var_juntas[0]);
    //printf("var_juntas[2]: %f \n",var_juntas[2]);
        //-------------------------------------------------------------------
-------
        //trocando Rx com Rz
        //-------------------------------------------------------------------
-------
        z=vel_cart[3];
        vel_cart[3]=vel_cart[5];
        vel_cart[5]=z;
        //-------------------------------------------------------------------
-------
        //Calculando antecipadamente senos e cossenos
        //-------------------------------------------------------------------
-------
        C1=c1;
        C2=c2;
        C4=c4;
        S1=s1;
        S2=s2;
        S4=s4;
        D3=d3;
        L2_Q=L2*L2;
        D2_Q=D2*D2;
        C4_Q=C4*C4;
        S4_Q=S4*S4;
        D3_Q=D3*D3;
        C2_Q=C2*C2;
        S2_Q=S2*S2;
        C2_D3=C2*D3;
        S2_D3=S2*D3;
        C4_S4=C4*S4;
        L2_S2=L2*S2;
        C1_S2_S4=C1*S2*S4;
        C2_S2_D3=C2*S2*D3;

        //-------------------------------------------------------------------
-------
        //calculos que determinam o vetor vel_juntas
        //SE a distância é pequena, vp[i]=0
        //-------------------------------------------------------------------
-------

        divisor=(+C2*(+D3*(+2*S2*C4*(-
L2*C4+D2*S4)+2*L2_S2+D3*(+C2*(+1+D3_Q-
L2_Q)+2*D3*L2_S2))+C4_Q*C2*(+L2_Q+D2_Q)+C2*(-L2_Q-D2_Q))
                        +L2_Q*(+D3_Q+1)+D2_Q+C4_Q*(-L2_Q-D2_Q));
        vel_juntas[0]=
(vel_cart[0]*(+C1*S2*(+D2*S2+D3*C4_S4*C2)+S1*L2_S2+S1*D3*(+D3*(+D3*C2+L2_
S2)+C2)+C4_Q*(-C1*D2-S1*C2_D3+C1*C2_Q*D2-S1*L2_S2))
                        +vel_cart[1]*(+C1*D3*(-D3*L2_S2-D3_Q*C2-
C2)+S1*(+S2*(+D2*S2+D3*C4_S4*C2))-C1*L2_S2+C4_Q*(-
S1*D2*S2_Q+C1*(+L2_S2+C2_D3)))
                        +vel_cart[2]*(+C2*(+S4*(-D2*S2*S4-
C2*C4*D3)))+vel_cart[3]*(-S1*D2*S2_D3+C4_S4*D3*(-D2*C1*S2_Q-
C2_D3*S1)+C4_Q*(+S2_D3*
```

```
                          (-C1*C2_D3+D2*S1)))+vel_cart[4]*(+D3*(+C4_S4*(-
D2*S1*S2_Q+C2_D3*C1)+C1*D2*S2+C4_Q*S2*(-S1*C2_D3-D2*C1)))+

vel_cart[5]*(+D3*C4*C2*(+D3*C2*C4+D2*S2*S4)))/divisor;
        vel_juntas[1]= (vel_cart[0]*(+C4_S4*C2*(+C1*D2+C2*(+S1*D3-
C1*C2*D2))+C2*S2*(+S1*(+C4_S4*L2-D2*D3_Q)+C2_D3*C1*(-L2_Q+C4_Q+D3_Q))
                          +D3*L2_S2*(+C1*(+2*C2_S2_D3+L2)-
D2*S1*S2))+vel_cart[1]*(+S1*C2_Q*(+D3*(+S2*(-L2_Q+D3_Q+C4_Q)-2*C2_D3*L2)
                          -C2*D2*C4_S4)+C4_S4*C2*(+S1*D2+C1*(-C2_D3-
L2_S2))+D2*D3*C1*(-L2*C2_Q+S2_D3*C2+L2)+S1*D3*L2*(+L2_S2+2*C2_D3))
                          +vel_cart[2]*(+C2*(+C2*(+C2_D3*(-D3_Q+L2_Q-
C4_Q)+S2*(-D2*C4_S4-2*D3_Q*L2))-L2_Q*D3))
                          +vel_cart[3]*(+C4_S4*(+C2_Q*C1*(-
2*C2_D3*L2+S2*(-D2_Q+D3_Q-
L2_Q))+C1*(+S2*(+L2_Q+D2_Q)+2*C2_D3*L2)+C2*D2*S2_D3*S1)
                          +C4_Q*(+S1*(-L2_Q-D2_Q)+C2*(-
C2_Q*C2*D2*D3*C1+D2*D3*C1+S1*(+C2*(+D2_Q+L2_Q-D3_Q)-2*D3*L2_S2)))
                          +S1*(+D2_Q+L2_Q+2*C2_D3*L2_S2+C2_Q*(-D2_Q-
L2_Q+D3_Q)))+vel_cart[4]*(+C2*(+C2*(+C1*D2_Q-C1*D3_Q+C1*L2_Q
                          +C4*(+S1*S2*S4*(-D2_Q+D3_Q-L2_Q)+S1*C2*(+D3*(-
D2*C4-2*L2*S4))+C4*C1*(+D3_Q-L2_Q)))
                          +C4*(+C1*(-S4*D2*S2_D3-
C2*D2_Q*C4+2*C4*D3*L2_S2)+D3*S1*(+2*L2*S4+D2*C4))-
2*C1*D3*L2_S2)+C4*(+S1*S2*S4*(+D2_Q+L2_Q)
                          +C4*C1*(+L2_Q+D2_Q))+C1*(-D2_Q-
L2_Q))+vel_cart[5]*(+C2*(+C2*(+C2*C4_S4*(+L2_Q+D2_Q-D3_Q)+C4*S2_D3*(-
2*S4*L2-D2*C4))
                          +C4_S4*(-L2_Q-D2_Q))))/divisor;
        vel_juntas[2]=-(+vel_cart[0]*(+C2*(+C2*(+C2*C1*(-
L2_Q+D3_Q*D3_Q+D3_Q-D2*C4_S4*L2+L2_Q*C4_Q-3*L2_Q*D3_Q)
                          +C1*S2_D3*(+L2*(+2+3*D3_Q-L2_Q-
C4_Q)+D2*C4_S4)+D3*(+D2*S1*(+L2_Q-1-
D3_Q)+C4_S4*L2*S1))+L2*(+C4*(+S1*S2*(+L2*S4+D2*C4)
                          +C1*(+S4*D2-L2*C4))+D3_Q*(-2*D2*S2*S1+3*C1*L2)-
D2*S1*S2+C1*L2)+C2*D2*C4_Q*S1*D3)+L2_Q*D3*(-D2*S1+C1*L2_S2))
                          +vel_cart[1]*(+C2*(+C2*(+D3*(+S1*(+D3*(+C2*(-
3*L2_Q+1+D3_Q)+3*L2*S2_D3)+D2*C4_S4*S2+2*L2_S2)
                          +C1*(+D2*(+1-L2_Q-C4_Q+D3_Q)-
C4_S4*L2))+S1*L2*(+C2*(+C4*(+L2*C4-D2*S4)-L2)+S2_D3*(-C4_Q-L2_Q)))
                          +L2*(+S1*(+L2*(-
C4_Q+3*D3_Q+1)+C4_S4*D2)+C1*S2*(+D2*(+S4_Q+2*D3_Q)-
C4*L2*S4))+L2_Q*D3*(+D2*C1+S1*L2_S2))
                          +vel_cart[2]*(+C2*(+C2*(+D3*(+D3*(+S2*(+1+D3_Q-
3*L2_Q)-3*C2_D3*L2)+C2*(+C4*(-D2*S4+L2*C4)+L2*(-2+L2_Q)))
                          +L2_S2*(-L2+C4*(-D2*S4+L2*C4)))+D3*(+L2*(-L2_Q-
2*C4_Q+2*D3_Q+2)+2*D2*C4_S4))+L2_Q*S2*(+D3_Q+S4_Q)+S2*D2_Q*S4_Q)

+vel_cart[3]*(+C2*(+C4*(+C2*(+D2*(+D3*(+D3*(+S2*C1*C4+S4*S1)-
D2*C2*C1*S4)-D2*L2*C1_S2_S4+D2*C4*L2*S1)
                          +L2*(+L2_Q*C4*S1-L2_Q*C1_S2_S4+D3*C1*(+S2_D3*S4-
2*L2*C2*S4-C2*D2*C4)))+D3*(+L2*(+C4*(+S1*(-2*L2_S2-C2_D3)+D2*C1)
                          +S4*(+D2*S2*S1+2*L2*C1))+D2_Q*(-
C4*S2*S1+C1*S4)))+L2*S1*(+C2*(-L2_Q+D3_Q-D2_Q)+2*D3*L2_S2)+S1*D2_Q*S2_D3)
                          +L2*(+C4*(+C1_S2_S4*(+D2_Q+L2_Q)+C4*S1*(-D2_Q-
L2_Q))+S1*(+L2_Q+D2_Q)))

+vel_cart[4]*(+C2*(+C4*(+S4*(+S1*(+C2*(+D3*(+C2*(-D2_Q-
2*L2_Q)+L2*S2_D3)+S2*L2*(-L2_Q-D2_Q))+D3*(+D2_Q+2*L2_Q))
                          +D3*C1*D2*(-L2_S2-
C2_D3))+C4*(+S1*(+D3*D2*(+C2*(-
C2*L2+S2_D3)+L2))+C1*(+S2_D3*(+2*L2_Q+D2_Q)+C2*L2*(-L2_Q-D2_Q+D3_Q))))
```

```
                          +C1*(+L2*(-2*D3*L2_S2+C2*(+D2_Q -D3_Q+L2_Q))-
D2_Q*S2_D3))+L2*(+C1*(+C4_Q*(+D2_Q+L2_Q)-L2_Q-D2_Q)

+C4_S4*S2*S1*(+L2_Q+D2_Q)))+vel_cart[5]*(+C2*(+C2*C4*(+D3*(+D2*C4*(-
L2_S2-C2_D3)+S4*S2*(-D2_Q-2*L2_Q))
                          +C2*S4*L2*(+L2_Q-D3_Q+D2_Q))+C4_S4*L2*(-L2_Q-
D2_Q))))/divisor;

        vel_juntas[3]=0;
        vel_juntas[4]=0;
        //-----------------------------------------------------------------
-------
        //Calculando as novas coordenadas das Juntas
        //SOMENTE PARA OS TRES PRIMEIROS GRAUS DE LIBERDADE, EM FUNÇÃO DA
        //IMPLEMENTAÇÃO DA LINHA RETA!!!!
        //-----------------------------------------------------------------
-------
        for(i=0;i<=2;i++){
            var_juntas_temp[i]=var_juntas[i];
            var_juntas[i]=var_juntas[i]+int_tempo*vel_juntas[i];
            diferenca_Q[i]=var_juntas[i]-var_juntas_temp[i];
        }
        calcula_comandos_motores(vel_juntas,diferenca_Q,cont);
        //-----------------------------------------------------------------
-------
        //após a movimentação, determina a posição correta do manipulador
        //-----------------------------------------------------------------
-------
        for(i=0;i<=2;i++){
            var_juntas[i]=var_juntas_temp[i]+passos[i]*res_pulso[i];
        }
        //-----------------------------------------------------------------
-------
        //Calculando a Posição Atual!
        //-----------------------------------------------------------------
-------
        //printf("vel_juntas[0]: %f \n",vel_juntas[0]);
        calcula_matriz();
        calcula_var_cart();
    printf("var_cart[0]: %f \n",var_cart[0]);
    printf("var_cart[1]: %f \n",var_cart[1]);
    printf("var_cart[2]: %f \n",var_cart[2]);
    printf("var_cart[3]: %f \n",var_cart[3]);
    printf("var_cart[4]: %f \n",var_cart[4]);
    printf("var_cart[5]: %f \n",var_cart[5]);
        //-----------------------------------------------------------------
-------
        //Calculando a diferença entre a posição inicial e a final em
cada eixo
        //-----------------------------------------------------------------
-------
        for (i=0;i<=5;i++){
            diferenca_P[i]=var_cart_fim[i]-var_cart[i];
        }

        //-----------------------------------------------------------------
-------
        //calculando a distancia atual
        //-----------------------------------------------------------------
-------
        dist_ant=dist_atual;
```

```
dist_atual=sqrt(diferenca_P[0]*diferenca_P[0]+diferenca_P[1]*diferenca_P[
1]+diferenca_P[2]*diferenca_P[2]);
        //dist_ant=0.001;
        //dist_atual=0.002;
        if(dist_ant<dist_atual){
    //printf("var_juntas[0]: %f \n",var_juntas[0]*RAD_GRAU);
    //printf("var_juntas[2]: %f \n",var_juntas[2]);
    //printf("dist_atual: %f\n",dist_atual);
    //printf("dist_ant: %f\n",dist_ant);
            break;
        }
        printf("c= %d\n",cont);
        cont++;
        //if (cont==1){
            //printf("c= %d\n",cont);
            //cont=0;
        //    dist_atual=0.001;
        //}

    }//fim do while()
/*
   printf("\nCont: %d\n",cont);
   printf("\ndist_Atual: %f\n",dist_ant);
   printf("Vetor de direções: \n");
   for (i=0;i<=2;i++){
    for (j=0;j<cont;j++){
       printf("dir[%d][%d]=%d \n",i,j,dir_out[i][j]);
    }
   }

   printf("Vetor de div_freq: \n");
   for (i=0;i<=2;i++){
    for (j=0;j<cont;j++){
       printf("div_freq[%d][%d]=%f \n",i,j,div_freq_out[i][j]);
    }
   }
*/
   printf("Vetor de passos: \n");
   for (i=0;i<=2;i++){
    for (j=0;j<cont;j++){
       //printf("passos[%d][%d]=%d \n",i,j,passos_out[i][j]);
       passos_total[i]=passos_total[i]+passos_out[i][j];
    }
   }
/*
   for (i=0;i<=2;i++){
       printf("passos_total[%d]=%d \n",i,passos_total[i]);
   }
*/
   printf("\nANGULOS: \n");
   printf("Q1 - %f\n",var_juntas[0]*RAD_GRAU);
   printf("Q2 - %f\n",var_juntas[1]*RAD_GRAU);//theta2
   printf("D3 - %f\n",var_juntas[2]);//d3
   printf("Q4 - %f\n",var_juntas[3]*RAD_GRAU);
   printf("Q5 - %f\n\n",var_juntas[4]*RAD_GRAU);

//    printf("\nO vetor de coordenadas cartesianas, na forma
T(Tx,Ty,Tz)R(Rz,Ry,Rx): \n\t");
//    for(i=0;i<3;i++){
//        printf("%f ",var_cart[i]);
```

```
//    }
//    for(i=3;i<VAR_CART;i++){
//        printf("%f ",var_cart[i]*RAD_GRAU);
//    }
    comanda_motores(cont,passos_total);
}

/*-------------------------------------------------------------------------
-------
int comanda_motores_v2()
Função que envia comandos de deslocamento para os motores
-------------------------------------------------------------------------
-----*/
int calcula_comandos_motores(double *vel_juntas,double *dist,int cont){

    double freq_desej;
    double div[3];
    int i;

    //IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
    //IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
    //IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);

//Definindo a direção do Movimento
    if (vel_juntas[0] >= 0.000f){//1/0 -> Horario  0/1->antihorario
        //IOWR_ALTERA_AVALON_PIO_DATA(DIR_M1_BASE,0x000);//sentido
positivo  (horario)
        dir_out[0][cont]=0x000;
    }else {
        //IOWR_ALTERA_AVALON_PIO_DATA(DIR_M1_BASE,0x001);//sentido
negativo  (anti-horario)
        dir_out[0][cont]=0x001;
    }
    if (vel_juntas[1] >= 0.000f){
        //IOWR_ALTERA_AVALON_PIO_DATA(DIR_M2_BASE,0x001);//sentido
positivo  (baixo)
        dir_out[1][cont]=0x001;
    }else {
        //IOWR_ALTERA_AVALON_PIO_DATA(DIR_M2_BASE,0x000);//sentido
negativo  (cima)
        dir_out[1][cont]=0x000;
    }
    if (vel_juntas[2] >= 0.000f){
        //IOWR_ALTERA_AVALON_PIO_DATA(DIR_M3_BASE,0x001);//sentido
positivo  (frente)
        dir_out[2][cont]=0x001;
    }else {
        //IOWR_ALTERA_AVALON_PIO_DATA(DIR_M3_BASE,0x000);//sentido
negativo  (tras)
        dir_out[2][cont]=0x000;
    }

//-------------------------------------------------------------------------
//Definindo a frequencia desejada para a movimentação
    //calcula a frequencia para os dois primeiros eixos
    for (i=0;i<=1;i++){
        if (vel_juntas[i]<0.0001f && vel_juntas[i]>-0.0001f){
            freq_desej = 0.5f;
        } else if(vel_juntas[i]<0.0f){
            freq_desej = (-vel_juntas[i]/res_pulso[i]);
        }else {
```

```
            freq_desej = ( vel_juntas[i]/res_pulso[i]);
        }
        div[i]=FREQ_ORIG/freq_desej;
        div[i]=div[i]/2.0f;
    }
    //calcula a frequencia para o 3 eixo
    if (vel_juntas[2]<0.0001f && vel_juntas[2]>-0.0001f){
        freq_desej = 0.5f;
    } else if(vel_juntas[2]<0.0f){
        freq_desej = -vel_juntas[2]/res_pulso[2];
    }else {
        freq_desej = vel_juntas[2]/res_pulso[2];
    }
    div[2]=FREQ_ORIG/freq_desej;
    div[2]=div[i]/2.0f;
    //printf("vel_junta[2]: %f\n",vel_juntas[2]);
    //printf("freq_desej[2]: %f\n",freq_desej);
    //printf("dist[2]: %f\n",dist[2]);
//Calculando o numero de passos a serem dados
    if (vel_juntas[0]<0.0001f && vel_juntas[0]>-0.0001f){
        //IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M1_BASE,0x000);
        passos_out[0][cont]=0x000;
    } else{
        if(dist[0]<0.00){
            //IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M1_BASE,round(-
dist[0]/res_pulso[0]));
            //printf("Passos [0]: %f\n",round(-dist[0]/res_pulso[0]));
            passos_out[0][cont]=round(-dist[0]/res_pulso[0]);
        }else{

//IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M1_BASE,round(dist[0]/res_pulso[0]));
            //printf("Passos [0]: %f\n",round(dist[0]/res_pulso[0]));
            passos_out[0][cont]=round(dist[0]/res_pulso[0]);
        }
    }
    if (vel_juntas[1]<0.0001f && vel_juntas[1]>-0.0001f){
        //IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M2_BASE,0x000);
        passos_out[1][cont]=0x000;
    } else{
        if (dist[1]<0.00){
            //IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M2_BASE,round(-
dist[1]/res_pulso[1]));
            passos_out[1][cont]=round(-dist[1]/res_pulso[1]);
        }else{

//IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M2_BASE,round(dist[1]/res_pulso[1]));
            passos_out[1][cont]=round(dist[1]/res_pulso[1]);
        }
    }
    if (vel_juntas[2]<0.0001f && vel_juntas[2]>-0.0001f){
        //IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M3_BASE,0x000);
        passos_out[2][cont]=0x000;
    } else{
        if(dist[2]<0.00){
            //IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M3_BASE,round(-
dist[2]/res_pulso[2]));
            passos_out[2][cont]=round(-dist[2]/res_pulso[2]);
    //        printf("Passos [2]: %f\n",round(-dist[2]/res_pulso[2]));
        }else{

//IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M3_BASE,round(dist[2]/res_pulso[2]));
```

```
            passos_out[2][cont]=round(dist[2]/res_pulso[2]);
    //        printf("Passos [2]: %f\n",round(dist[2]/res_pulso[2]));
        }
    }
    passos[0]=round(dist[0]/res_pulso[0]);
    passos[1]=round(dist[1]/res_pulso[1]);
    passos[2]=round(dist[2]/res_pulso[2]);

    conta_passos=conta_passos+round(-dist[0]/res_pulso[0]);
    conta_passos=conta_passos+round(-dist[2]/res_pulso[2]);

    //printf("Div 2: %f\n",div[2]);
    //printf("Dist 2: %f\n",dist[2]);
    //printf("Conta 2: %f\n",conta_passos);

    //IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M1_BASE,div[0]);
    div_freq_out[0][cont]=div[0];
    //IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M2_BASE,div[1]);
    div_freq_out[1][cont]=div[1];
    //IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M3_BASE,div[2]);//0x169
    div_freq_out[2][cont]=div[2];

/*
//Verificando se o motor deve ou não movimentar-se
    if (vel_juntas[0]<0.0001f && vel_juntas[0]>-0.0001f){
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
    } else{
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x000);
    //    printf("Habilitou 0\n");
    }
    if (vel_juntas[1]<0.0001f && vel_juntas[1]>-0.0001f){
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
    } else{
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x000);
    //    printf("Habilitou 1\n");
    }
    if (vel_juntas[2]<0.0001f && vel_juntas[2]>-0.0001f){
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);
    } else{
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x000);
    //    printf("Habilitou 2\n");
    }
usleep(10000);
    //while(!IORD_ALTERA_AVALON_PIO_DATA(FIM_1_BASE) ||
!IORD_ALTERA_AVALON_PIO_DATA(FIM_2_BASE) ||
!IORD_ALTERA_AVALON_PIO_DATA(FIM_3_BASE));
    while(IORD_ALTERA_AVALON_PIO_DATA(FIM_M3_BASE)==0||
          IORD_ALTERA_AVALON_PIO_DATA(FIM_M2_BASE)==0||
          IORD_ALTERA_AVALON_PIO_DATA(FIM_M1_BASE)==0){
        //printf("espera...\n");
    }


    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);
*/
    return 0;
}
/*-------------------------------------------------------------------------
-------
```

```
int comanda_motores()
Função que envia comandos de deslocamento para os motores
--------------------------------------------------------------------------
-----*/
int comanda_motores(int cont, int *passos_total){

    int i,cont1,cont2,cont3;
    int passos_lim[VAR_JUNTAS];
    cont1=0;
    cont2=0;
    cont3=0;
  //  printf("Vetor de passos: \n");
    for (i=0;i<=2;i++){
  //      printf("passos[%d]=%d \n",i,passos_total[i]);
        passos_lim[i]=passos_out[i][0];
  //      printf("passos_lim[%d]=%d \n",i,passos_lim[i]);
    }

    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);

  //  printf("Inicio do movimento?? \n");
    //-----------------------------------------------------
    //Determinando que será contado o número de passos
    //-----------------------------------------------------
    IOWR_ALTERA_AVALON_PIO_DATA(PASSOS_BASE,0x001);

    IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M1_BASE,passos_total[0]);
    IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M2_BASE,passos_total[1]);
    IOWR_ALTERA_AVALON_PIO_DATA(STEPS_M3_BASE,passos_total[2]);

  //  printf("passos_Total[0]= %d
\n",(passos_total[0]+passos_out[0][cont]));
  //  printf("passos_Total[1]= %d
\n",(passos_total[1]+passos_out[1][cont]));
  //  printf("passos_Total[2]= %d
\n",(passos_total[2]+passos_out[2][cont]));
    //-------------------------------------------------------------
    //Enquanto não finalizar-se o movimento nos três eixos
    //-------------------------------------------------------------
    while(IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M1_BASE)<
(passos_total[0]+passos_out[0][cont])
        ||IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M2_BASE) <
(passos_total[1]+passos_out[1][cont])
        ||IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M3_BASE) <
(passos_total[2]+passos_out[2][cont])){

        if (IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M1_BASE) ==
passos_total[0] &&
            IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M2_BASE) ==
passos_total[1] &&
            IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M3_BASE) ==
passos_total[2]){
            break;
        }

    //for (i=0;i<=2;i++){
  /*
    printf("passos_Total[0]= %d
\n",(passos_total[0]+passos_out[0][cont]));
```

```
    printf("passos_Total[1]= %d
\n",(passos_total[1]+passos_out[1][cont]));
    printf("passos_Total[2]= %d
\n",(passos_total[2]+passos_out[2][cont]));
        printf("passos[0]= %d
\n",IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M1_BASE));
        printf("passos[2]= %d
\n",IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M3_BASE));
        printf("freq[0][%d]= %f \n",cont1,div_freq_out[0][cont1]);
        printf("freq[2][%d]= %f \n",cont3,div_freq_out[2][cont3]);
  */
    //}

        //-------------------------------------------------------------
        //escrevendo o divisor de frequencia
        //-------------------------------------------------------------
        IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M1_BASE,div_freq_out[0][cont1]);
        IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M2_BASE,div_freq_out[1][cont2]);
        IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M3_BASE,div_freq_out[2][cont3]);
        //IOWR_ALTERA_AVALON_PIO_DATA(FREQ_M3_BASE,200);

        //-------------------------------------------------------------
        //escrevendo a direção
        //-------------------------------------------------------------
        IOWR_ALTERA_AVALON_PIO_DATA(DIR_M1_BASE,dir_out[0][cont1]);
        IOWR_ALTERA_AVALON_PIO_DATA(DIR_M2_BASE,dir_out[1][cont2]);
        IOWR_ALTERA_AVALON_PIO_DATA(DIR_M3_BASE,dir_out[2][cont3]);

        //-------------------------------------------------------------
        //Verificando se acabou o passo M1
        //-------------------------------------------------------------
        if(IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M1_BASE)>=passos_lim[0]){
            //printf("Incrementei o indice M1??\n");
            cont1++;
            //-------------------------------------------------------------
--
            //Somando os passos, para conferencia
            //-------------------------------------------------------------
--
            passos_lim[0]=passos_lim[0]+passos_out[0][cont1];
        }

        //-------------------------------------------------------------
        //Verificando se acabou o passo M2
        //-------------------------------------------------------------
        if(IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M2_BASE)>=passos_lim[1]){
            //printf("Incrementei o indice M2??\n");
            cont2++;
            //-------------------------------------------------------------
--
            //Somando os passos, para conferencia
            //-------------------------------------------------------------
--
            passos_lim[1]=passos_lim[1]+passos_out[1][cont2];
        }

        //-------------------------------------------------------------
        //Verificando se acabou o passo M3
        //-------------------------------------------------------------
        if(IORD_ALTERA_AVALON_PIO_DATA(PASSOS_A_M3_BASE)>=passos_lim[2]){
            //printf("Incrementei o indice M3??\n");
```

```
            cont3++;
            //-------------------------------------------------------------
--
            //Somando os passos, para conferencia
            //-------------------------------------------------------------
--
            passos_lim[2]=passos_lim[2]+passos_out[2][cont3];
        }
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x000);
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x000);
        IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x000);
        //usleep(10000);
    }
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M1_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M2_BASE,0x001);
    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_M3_BASE,0x001);

    printf("Fim do movimento??\n");

return 0;
}
```

ANEXO B - DATASHEET

Nesse anexo, são encontrados trechos dos *datasheets* dos componentes utilizados no projeto:

B.1 - OPTO ACOPLADOR - 6N137

HEWLETT PACKARD

High CMR, High Speed TTL Compatible Optocouplers

Technical Data

6N137
HCNW137 **HCPL-0631**
HCNW2601 **HCPL-0661**
HCNW2611 **HCPL-2601**
HCPL-0600 **HCPL-2611**
HCPL-0601 **HCPL-2630**
HCPL-0611 **HCPL-2631**
HCPL-0630 **HCPL-4661**

Features

- 5 kV/µs Minimum Common Mode Rejection (CMR) at V_{CM} = 50 V for HCPL-X601/X631, HCNW2601 and 10 kV/µs Minimum CMR at V_{CM} = 1000 V for HCPL-X611/X661, HCNW2611
- High Speed: 10 MBd Typical
- LSTTL/TTL Compatible
- Low Input Current Capability: 5 mA
- Guaranteed ac and dc Performance over Temperature: -40°C to +85°C
- Available in 8-Pin DIP, SOIC-8, Widebody Packages
- Strobable Output (Single Channel Products Only)
- Safety Approval
  UL Recognized - 2500 V rms for 1 minute and 5000 V rms* for 1 minute per UL1577
  CSA Approved
  VDE 0884 Approved with V_{IORM} = 630 V_{peak} for HCPL-2611 Option 060 and V_{IORM} = 1414 V_{peak} for HCNW137/26X1
  BSI Certified (HCNW137/26X1 Only)
- MIL-STD-1772 Version Available (HCPL-56XX/66XX)

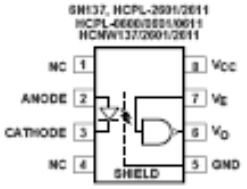
Applications

- Isolated Line Receiver
- Computer-Peripheral Interfaces
- Microprocessor System Interfaces
- Digital Isolation for A/D, D/A Conversion
- Switching Power Supply
- Instrument Input/Output Isolation
- Ground Loop Elimination
- Pulse Transformer Replacement
- Power Transistor Isolation in Motor Drives
- Isolation of High Speed Logic Systems

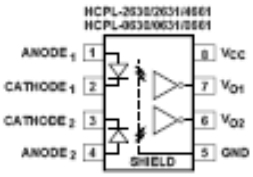
Description

The 6N137, HCPL-26XX/06XX/4661, HCNW137/26X1 are optically coupled gates that combine a GaAsP light emitting diode and an integrated high gain photo detector. An enable input allows the detector to be strobed. The output of the detector IC is

Functional Diagram



TRUTH TABLE (POSITIVE LOGIC)		
LED	ENABLE	OUTPUT
ON	H	L
OFF	H	H
ON	L	H
OFF	L	H
ON	NC	L
OFF	NC	H



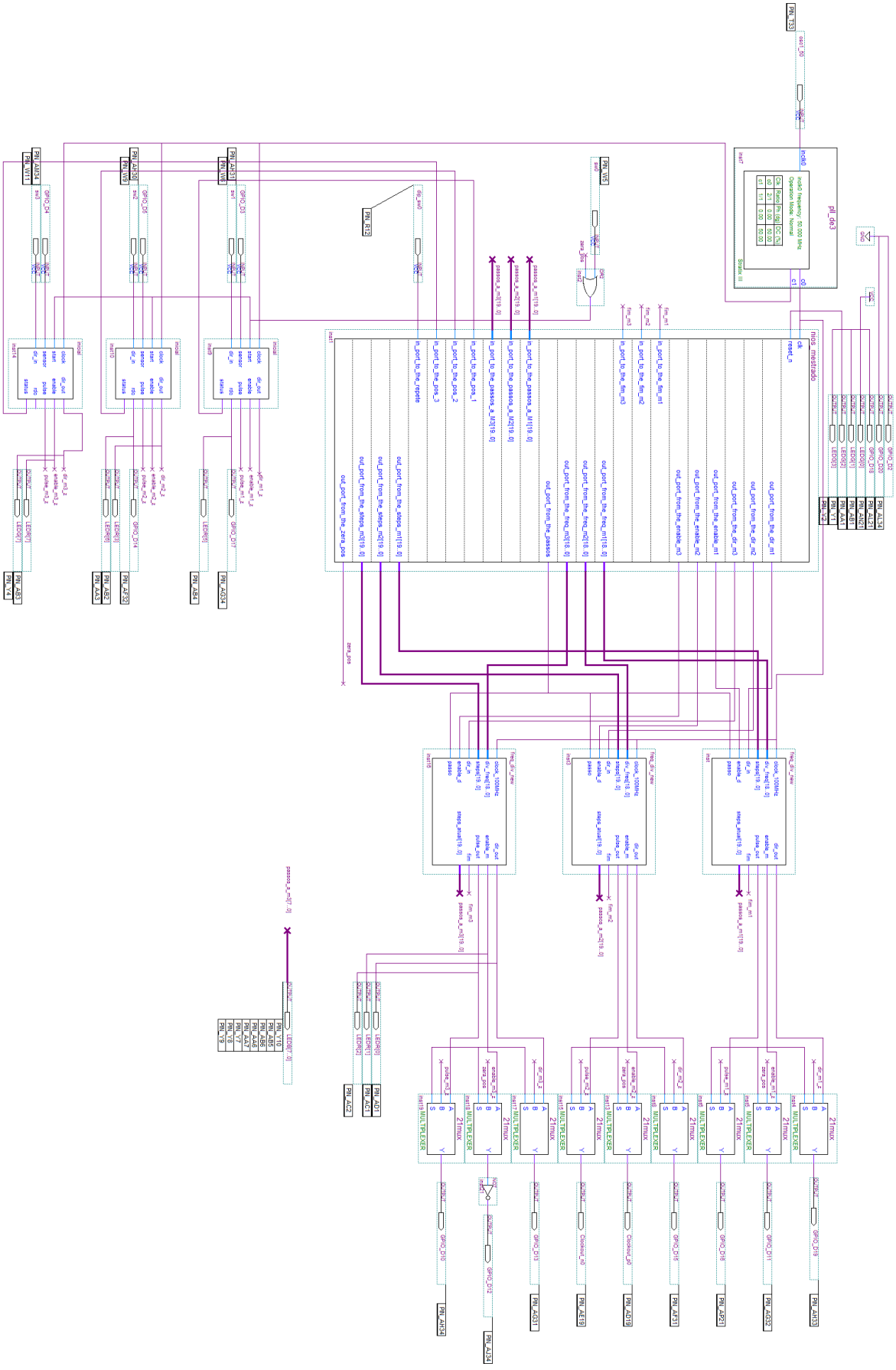
TRUTH TABLE (POSITIVE LOGIC)	
LED	OUTPUT
ON	L
OFF	H

*5000 V rms/1 Minute rating is for HCNW137/26X1 and Option 020 (6N137, HCPL-2601/11/30/31, HCPL-4661) products only.
A 0.1 µF bypass capacitor must be connected between pins 5 and 8.

CAUTION: It is advised that normal static precautions be taken in handling and assembly of this component to prevent damage and/or degradation which may be induced by ESD.

5965-3594E

ANEXO C – *HARDWARE* EMBARCADO



ANEXO D: CINEMÁTICA DA FERRAMENTA

No capitulo 3 desse trabalho, foi tratado o calculo da cinemática direta e inversa, bem como o Jacobiano do manipulador. Contudo, em função da não determinação de uma ferramenta, optou-se por simplificar uma serie de cálculos, isolando a influência dessa na manipulação dos outros eixos do manipulador. Nesse anexo, serão apresentados os mesmos cálculos, levando-se em consideração a influência de uma ferramenta genérica a ser escolhida posteriormente.

D.1. CINEMÁTICA DIRETA

Conforme apresentado na secção 3.2, a transformação cinemática direta para o manipulador pode ser obtida a partir da matriz homogênea, que representa a localização de seu elemento terminal em relação ao sistema de coordenadas fixado em sua base. Em outras palavras, definindo-se as coordenadas cartesianas da posição da origem do sistema de coordenadas fixado na ferramenta do robô e a orientação do mesmo sistema em relação ao sistema de referência fixado na base do robô, tudo em função dos parâmetros de D-H, os quais incluem as variáveis de juntas. Considerando que o sistema fixado à ferramenta localiza-se relativamente ao sistema 5 por meio da matriz ${}^{5}A_{Tool}$, definida pela eq. (D.**1**), a transformação cinemática direta é obtida a partir da eq. (D.**2**). A matriz ${}^{R}A_{Tool}$ representa a localização que se deseja atingir com o elemento terminal do robô e tem uma forma semelhante à apresentada na eq. (3.29).

$$ {}^{5}A_{Tool} = \begin{bmatrix} n_x & o_x & a_x & k_x \\ n_y & o_y & a_y & k_y \\ n_z & o_z & a_z & k_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_1 & o_1 & a_1 & k_1 \\ n_2 & o_2 & a_2 & k_2 \\ n_3 & o_3 & a_3 & k_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.1} $$

$$ {}^{R}T_{Tool} = {}^{0}A_1 \cdot {}^{1}A_2 \cdot {}^{2}A_3 \cdot {}^{3}A_4 \cdot {}^{4}A_5 \cdot {}^{5}A_{Tool} \tag{D.2} $$

A cinemática direta é obtida comparando-se a matriz ${}^{R}T_{Tool}$ com a matriz de transformação genérica do manipulador, separando-se os efeitos de rotação e de translação, conforme mostram as eqs. (D.3), (D.4) e (D.5), em que a orientação do sistema da ferramenta é representada segundo os ângulos Roll-Pitch-Yaw (RPY).

$$ {}^{R}T_{Tool} = {}^{0}A_1 \cdot {}^{1}A_2 \cdot {}^{2}A_3 \cdot {}^{3}A_4 \cdot {}^{4}A_5 \cdot {}^{5}A_{Tool} = Trans(p_x, p_y, p_z) \cdot RPY(\phi, \theta, \psi) \tag{D.3} $$

$$ {}^{R}T_{Tool} = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c\phi \cdot c\theta & c\phi \cdot s\theta \cdot s\psi - s\phi \cdot c\psi & c\phi \cdot s\theta \cdot c\psi + s\phi \cdot s\psi & 0 \\ s\phi \cdot c\theta & s\phi \cdot s\theta \cdot s\psi + c\phi \cdot c\psi & s\phi \cdot s\theta \cdot c\psi - c\phi \cdot s\psi & 0 \\ -s\theta & c\theta \cdot s\psi & c\theta \cdot c\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.4} $$

$$ {}^{R}T_{Tool} = \begin{bmatrix} c\phi \cdot c\theta & c\phi \cdot s\theta \cdot s\psi - s\phi \cdot c\psi & c\phi \cdot s\theta \cdot c\psi + s\phi \cdot s\psi & p_x \\ s\phi \cdot c\theta & s\phi \cdot s\theta \cdot s\psi + c\phi \cdot c\psi & s\phi \cdot s\theta \cdot c\psi - c\phi \cdot s\psi & p_y \\ -s\theta & c\theta \cdot s\psi & c\theta \cdot c\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.5} $$

Conforme explicitado nas equações acima, a transformação cinemática direta para posição é obtida igualando-se os termos de translação da transformação homogênea do lado direito da eq. **Erro! Fonte de referência não encontrada.** aos termos correspondentes da ransformação homogênea resultante da multiplicação matricial explicitada no lado direito da eq. (D.2). As eqs. (D.6), (D.7) e (D.8) mostram isso.

$$ {}^{R}T_{Tool} = {}^{0}A_1 \cdot {}^{1}A_2 \cdot {}^{2}A_3 \cdot {}^{3}A_4 \cdot {}^{4}A_5 \cdot {}^{5}A_{Tool} = {}^{0}A_5 \cdot {}^{5}A_{Tool} \tag{D.6} $$

$$ {}^{R}T_{Tool} = {}^{0}A_5 \cdot {}^{5}A_{Tool} = \begin{bmatrix} \left[{}^{0}A_5\right]_{11} & \left[{}^{0}A_5\right]_{12} & \left[{}^{0}A_5\right]_{13} & \left[{}^{0}A_5\right]_{14} \\ \left[{}^{0}A_5\right]_{21} & \left[{}^{0}A_5\right]_{22} & \left[{}^{0}A_5\right]_{23} & \left[{}^{0}A_5\right]_{24} \\ \left[{}^{0}A_5\right]_{31} & \left[{}^{0}A_5\right]_{32} & \left[{}^{0}A_5\right]_{33} & \left[{}^{0}A_5\right]_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} n_1 & o_1 & a_1 & k_1 \\ n_2 & o_2 & a_2 & k_2 \\ n_3 & o_3 & a_3 & k_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.7} $$

$$ \begin{aligned} {}^{R}T_{Tool} &= \begin{bmatrix} c\phi \cdot c\theta & c\phi \cdot s\theta \cdot s\psi - s\phi \cdot c\psi & c\phi \cdot s\theta \cdot c\psi + s\phi \cdot s\psi & p_x \\ s\phi \cdot c\theta & s\phi \cdot s\theta \cdot s\psi + c\phi \cdot c\psi & s\phi \cdot s\theta \cdot c\psi - c\phi \cdot s\psi & p_y \\ -s\theta & c\theta \cdot s\psi & c\theta \cdot c\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j} \cdot n_j\right) & \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j} \cdot o_j\right) & \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j} \cdot a_j\right) & \left\{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j} \cdot k_j\right) + \left[{}^{0}A_5\right]_{14}\right\} \\ \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j} \cdot n_j\right) & \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j} \cdot o_j\right) & \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j} \cdot a_j\right) & \left\{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j} \cdot k_j\right) + \left[{}^{0}A_5\right]_{24}\right\} \\ \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j} \cdot n_j\right) & \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j} \cdot o_j\right) & \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j} \cdot a_j\right) & \left\{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j} \cdot k_j\right) + \left[{}^{0}A_5\right]_{34}\right\} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{D.8} $$

As eqs. (D.8), (D.9) e (D.10) mostram explicitamente as coordenadas, relativas ao sistema de coordenadas fixado na base do robô, de posição da origem do sistema de coordenadas da tocha de soldagem em função dos parâmetros de D-H.

$$ p_x = \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j} \cdot k_j\right) + \left[{}^{0}A_5\right]_{14} \tag{D.9} $$

$$ p_y = \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j} \cdot k_j\right) + \left[{}^{0}A_5\right]_{24} \tag{D.10} $$

$$ p_z = \left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j} \cdot k_j\right) + \left[{}^{0}A_5\right]_{34} \tag{D.11} $$

Os ângulos RPY, (ϕ,θ e ψ), são obtidos comparando-se as partes de rotação das duas matrizes da eq.**Erro! Fonte de referência não encontrada.**, resultando nas formas explícitas as eqs. (D.12) a (D.13), para o caso em que $[^{R}T_{Tool}]_{31}$, em módulo, é diferente de 1 (um).

$$\phi = R_z = \tan^{-1}\frac{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j}\cdot n_j\right)}{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j}\cdot n_j\right)} \tag{D.12}$$

$$\theta = R_y = \tan^{-1}\frac{-\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j}\cdot n_j\right)}{\sqrt{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j}\cdot n_j\right)^2+\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j}\cdot n_j\right)^2}} \tag{D.13}$$

$$\psi = R_x = \tan^{-1}\frac{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j}\cdot o_j\right)}{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{3j}\cdot a_j\right)} \tag{D.14}$$

Para o caso em que $[^{R}T_{Tool}]_{31}$ assume o valor 1, o valor do ângulo θ = -90º. Nesse caso, devido à singularidade nas equações (D.12) a (D.14), assume-se que o ângulo ϕ = 0 (zero) e o ângulo ψ é calculado pela eq. (D.15). Caso $[^{R}T_{Tool}]_{31}$ = -1, o ângulo θ = 90º e o ângulo ϕ é assumido igual a zero, resultando no cálculo do ângulo ψ conforme a eq. (D.16).

Se $[^{R}T_{Tool}]_{31}$ = 1 então,

$$\psi = R_x = \tan^{-1}\frac{-\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j}\cdot o_j\right)}{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j}\cdot a_j\right)}\ ,\ \theta = R_y = -90^{o}\ \text{ e }\ \phi = R_z = 0 \tag{D.15}$$

$$\text{Se } [^{R}T_{Tool}]_{31} = -1 \text{ então } \psi = R_x = \tan^{-1}\frac{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{1j}\cdot o_j\right)}{\left(\sum_{j=1}^{3}\left[{}^{0}A_5\right]_{2j}\cdot a_j\right)}\ ,\ \theta = R_y = 90^{o} \text{ e } \phi = R_z = 0 \tag{D.16}$$

Em resumo, a transformação cinemática direta é explicitada pelas eqs. (D.6) a (D.16), em que os elementos da transformação homogênea ${}^{0}A_5$, $[{}^{0}A_5]_{ij}$ (para i=1,2,3 e j=1,2,3,4), são explicitados nas eqs. (D.17) a (D.28).

$$\left[{}^{0}A_5\right]_{11} = -(c1s2c4 + s1s4)c5 - c1c2s5 \tag{D.17}$$

$$\left[{}^{0}A_{5}\right]_{21} = (-s1s2c4 + c1s4)c5 - s1c2s5 \tag{D.18}$$

$$\left[{}^{0}A_{5}\right]_{31} = c2c4c5 - s2s5 \tag{D.19}$$

$$\left[{}^{0}A_{5}\right]_{12} = (c1s2c4 + s1s4)s5 - c1c2c5 \tag{D.20}$$

$$\left[{}^{0}A_{5}\right]_{22} = (s1s2c4 - c1s4)s5 - s1c2c5 \tag{D.21}$$

$$\left[{}^{0}A_{5}\right]_{32} = -(c2c4s5 + s2c5) \tag{D.22}$$

$$\left[{}^{0}A_{5}\right]_{13} = -c1s2s4 + s1c4 \tag{D.23}$$

$$\left[{}^{0}A_{5}\right]_{23} = -(s1s2s4 + c1c4) \tag{D.24}$$

$$\left[{}^{0}A_{5}\right]_{33} = c2s4 \tag{D.25}$$

$$\left[{}^{0}A_{5}\right]_{14} = -c1c2d3 - c1l2s2 + s1d2 \tag{D.26}$$

$$\left[{}^{0}A_{5}\right]_{24} = -s1c2d3 - s1l2s2 - c1d2 \tag{D.27}$$

$$\left[{}^{0}A_{5}\right]_{34} = -s2d3 + l2c2 + d1 \tag{D.28}$$

D.2. CINEMÁTICA INVERSA

Para o cálculo da cinemática inversa, a influência da ferramenta pode ser desconsiderada ao utilizar-se a simplificação mostrada na secção 3.4, relacionada ao fato de que a ferramenta genérica a ser fixada no elemento terminal do robô tem sua geometria fixa. Desse modo, a matriz ${}^{5}A_{Tool}$ é constante, o que permite realizar uma simplificação na eq. (D.2), pós-multiplicando-se ambos os seus lados pela inversa de ${}^{5}A_{Tool}$, resultando na transformação ${}^{R}T_{5}$ conforme mostra a eq. (D.29). Desta forma, cada localização almejada para a tocha de soldagem resultará em uma matriz homogênea na forma da eq. (3.29), a qual deverá ser pós multiplicada por $[{}^{5}A_{Tool}]^{-1}$ para se obter a matriz numérica a partir da qual será extraída a cinemática inversa.

$${}^{R}T_{5} = {}^{R}T_{Tool} \cdot [{}^{5}A_{Tool}]^{-1} = {}^{0}A_{1} \cdot {}^{1}A_{2} \cdot {}^{2}A_{3} \cdot {}^{3}A_{4} \cdot {}^{4}A_{5} = \begin{bmatrix} x_x & y_x & z_x & p_x \\ x_y & y_y & z_y & p_y \\ x_z & y_z & z_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{D.29}$$

em que

$$[{}^5A_{Tool}]^{-1} = \begin{bmatrix} n_1 & o_1 & a_1 & k_1 \\ n_2 & o_2 & a_2 & k_2 \\ n_3 & o_3 & a_3 & k_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} n_1 & n_2 & n_3 & -\vec{n}\cdot\vec{k} \\ o_1 & o_2 & o_3 & -\vec{o}\cdot\vec{k} \\ a_1 & a_2 & a_3 & -\vec{a}\cdot\vec{k} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(D.30)}$$

Nesse caso, podem ser seguidos os cálculos apresentados na secção 3.4.1.

D.3. JACOBIANO DO MANIPULADOR

Como alternativa ao calculo do Jacobiano apresentado na secção 3.5.2, optou-se por obter as expressões componentes do Jacobiano do Manipulador a partir de derivação das equações de cinemática direta, no caso de movimentos de translação, e pelo cálculo vetorial, para o caso de movimentos de rotação. Para o primeiro desenvolvimento, parte-se das equações (D.2) a (D.4). Estas podem ser organizadas em forma vetorial conforme mostra a equação (D.31).

Nesta, observa-se que o vetor de posição da origem do sistema de coordenadas fixado na tocha de soldagem, ${}^0\vec{p}_{Tool}$, é obtido pela soma vetorial do vetor que posiciona a origem do sistema 5 (quarta coluna da matriz ${}^0A_5$), ${}^0\vec{p}_5$, e o vetor de posição relativa entre este e o sistema da tocha, ${}^5\vec{k}_{Tool}$, transferido para o sistema de coordenadas da base do robô, por meio da pré-multiplicação pela matriz de rotação contida na matriz ${}^0A_5$, ${}^0Rot_5$. Essa descrição é condensada na equação vetorial mostrada na eq. (D.32). Derivando-se o vetor ${}^0\vec{p}_{Tool}$, obtém-se a equação (D.33).

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} [{}^0A_5]_{14} \\ [{}^0A_5]_{24} \\ [{}^0A_5]_{34} \end{bmatrix} + \begin{bmatrix} [{}^0A_5]_{11} & [{}^0A_5]_{12} & [{}^0A_5]_{13} \\ [{}^0A_5]_{21} & [{}^0A_5]_{22} & [{}^0A_5]_{23} \\ [{}^0A_5]_{31} & [{}^0A_5]_{23} & [{}^0A_5]_{33} \end{bmatrix} \cdot \begin{bmatrix} [{}^5\vec{k}_{Tool}]_1 \\ [{}^5\vec{k}_{Tool}]_2 \\ [{}^5\vec{k}_{Tool}]_3 \end{bmatrix} \quad \text{(D.31)}$$

$${}^0\vec{p}_{Tool} = {}^0\vec{p}_5 + {}^0Rot_5 \cdot {}^5\vec{k}_{Tool} \quad \text{(D.32)}$$

$$\begin{aligned} \frac{d}{dt}\left({}^0\vec{p}_{Tool}\right) &= \frac{d}{dt}\left({}^0\vec{p}_5\right) + \frac{d}{dt}\left({}^0Rot_5 \cdot {}^5\vec{k}_{Tool}\right) \\ &= \frac{d}{dt}\left({}^0\vec{p}_5\right) + {}^0\vec{\omega}_5 \times \left({}^0Rot_5 \cdot {}^5\vec{k}_{Tool}\right) + {}^0Rot_5 \cdot \frac{d}{dt}\left({}^5\vec{k}_{Tool}\right) \end{aligned} \quad \text{(D.33)}$$

Em que ${}^0\vec{\omega}_5$ é o vetor velocidade angular do sistema 5 em relação ao sistema da base do robô. Na equação (D.33), a derivada do vetor ${}^5\vec{k}_{Tool}$ é nulo, tendo em vista o fato de este ser constante. Desta forma, pode-se escrever a mesma equação na forma matricial, explicitando-se os componentes a serem derivados, conforme mostra a equação (D.34).

$$
{}^{0}\begin{bmatrix} \frac{d}{dt} p_x \\ \frac{d}{dt} p_y \\ \frac{d}{dt} p_z \end{bmatrix}_{Tool} = {}^{0}\begin{bmatrix} \frac{d}{dt}\left[{}^{0}A_5\right]_{14} \\ \frac{d}{dt}\left[{}^{0}A_5\right]_{24} \\ \frac{d}{dt}\left[{}^{0}A_5\right]_{34} \end{bmatrix}_5 + {}^{0}\begin{bmatrix} \frac{d\psi}{dt} \\ \frac{d\theta}{dt} \\ \frac{d\phi}{dt} \end{bmatrix}_5 \times {}^{0}\begin{bmatrix} \left[{}^{0}A_5\right]_{11} & \left[{}^{0}A_5\right]_{12} & \left[{}^{0}A_5\right]_{13} \\ \left[{}^{0}A_5\right]_{21} & \left[{}^{0}A_5\right]_{22} & \left[{}^{0}A_5\right]_{23} \\ \left[{}^{0}A_5\right]_{31} & \left[{}^{0}A_5\right]_{23} & \left[{}^{0}A_5\right]_{33} \end{bmatrix}_5 \cdot \begin{bmatrix} {}^{5}\vec{k}_{Tool} \\ {}^{5}\vec{k}_{Tool} \\ {}^{5}\vec{k}_{Tool} \end{bmatrix}_3 \qquad \text{(D.34)}
$$

Já o vetor velocidade angular da tocha pode ser calculado somando os termos de velocidades angulares relativas entre os diversos elos do manipulador, tomando-se o cuidado de antes da soma vetorial, transformá-los para o sistema de coordenadas da base, conforme mostrado na equação (D.35).

$$
{}^{0}\vec{\omega}_{Tool} = {}^{0}\vec{\omega}_1 + {}^{0}Rot_1 \cdot {}^{1}\vec{\omega}_2 + {}^{0}Rot_2 \cdot {}^{2}\vec{\omega}_3 + {}^{0}Rot_3 \cdot {}^{3}\vec{\omega}_4 + {}^{0}Rot_4 \cdot {}^{4}\vec{\omega}_5 + {}^{0}Rot_5 \cdot {}^{5}\vec{\omega}_{Tool} \qquad \text{(D.35)}
$$

Considerando entretanto que a velocidade angular relativa entre o sistema 5 e a tocha de soldagem, ${}^{5}\vec{\omega}_{Tool}$, é nula, verifica-se que a velocidade angular da tocha é igual à velocidade angular do sistema 5, ${}^{0}\vec{\omega}_5$. Considerando ainda que as velocidades angulares relativas ocorrem sempre em relação ao eixo z do elo anterior, pode-se escrever a equação na forma matricial. Essa será o resultado da multiplicação de uma matriz, cujas colunas *i* são iguais às terceiras colunas das matrizes de rotação entre o sistema *0* e o sistema *i-1*, no caso de junta rotacional, e $\vec{0}$, no caso de junta translacional, e o vetor de movimento diferencial no espaço das juntas, $\vec{D}_q$ (vide eq. (3.68)). Na forma matricial, a eq. (D.35) assume a forma da equação (D.36).

$$
{}^{0}\begin{bmatrix} \frac{d\psi}{dt} \\ \frac{d\theta}{dt} \\ \frac{d\phi}{dt} \end{bmatrix}_{5} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \frac{d\theta_1}{dt} + \begin{bmatrix} {}^{0}A_1 \end{bmatrix}_{13} \frac{d\theta_2}{dt} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \frac{dd_3}{dt} + \begin{bmatrix} {}^{0}A_3 \end{bmatrix}_{13} \frac{d\theta_4}{dt} + \begin{bmatrix} {}^{0}A_4 \end{bmatrix}_{13} \frac{d\theta_5}{dt}
$$

$$
= \begin{bmatrix} 0 & s1 & 0 & -c1c2 & -c1s2s4 + s1c4 \\ 0 & -c1 & 0 & -s1c2 & -s1s2s4 - c1c4 \\ 1 & 0 & 0 & -s2 & c2s4 \end{bmatrix} \cdot \begin{bmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \\ \frac{dd_3}{dt} \\ \frac{d\theta_4}{dt} \\ \frac{d\theta_5}{dt} \end{bmatrix} \tag{D.36}
$$

$$
= \begin{bmatrix} J_{41} & J_{42} & J_{43} & J_{44} & J_{45} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} \end{bmatrix} \cdot \vec{D}_q = \begin{bmatrix} {}^{0}J_{Tool,R} \end{bmatrix}_{3x5} \cdot \vec{D}_q
$$

A equação (D.36) representa a parte do Jacobiano do manipulador relativo ao movimento de rotação. Note que o vetor velocidade angular resultante é utilizado para o cálculo da parte do Jacobiano relacionada ao movimento de translação, conforme mostram as equações (D.31) e (D.32). Substituindo a equação (D.36) na equação (D.34) obtém-se a expressão a partir da qual se pode extrair a parte do Jacobiano referente ao movimento de translação. Seu desenvolvimento é mostrado nas equações (D.37) a (D.41).

$$
{}^{0}\begin{bmatrix} \frac{d}{dt}\begin{bmatrix} {}^{0}A_5 \end{bmatrix}_{14} \\ \frac{d}{dt}\begin{bmatrix} {}^{0}A_5 \end{bmatrix}_{24} \\ \frac{d}{dt}\begin{bmatrix} {}^{0}A_5 \end{bmatrix}_{34} \end{bmatrix}_{5} = \begin{bmatrix} s1c2d_3 + s1s2l_2 + c1d_2 & c1s2d_3 - c1c2l_2 & -c1c2 & 0 & 0 \\ -c1c2d_3 - c1s2l_2 + s1d_2 & s1s2d_3 - s1c2l_2 & -s1c2 & 0 & 0 \\ 0 & -c2d_3 - s2l_2 & -s2 & 0 & 0 \end{bmatrix} \cdot \vec{D}_q \tag{D.37}
$$

$$
= \begin{bmatrix} \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{11} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{12} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{13} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{14} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{15} \\ \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{21} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{22} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{23} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{24} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{25} \\ \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{31} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{32} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{33} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{34} & \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{35} \end{bmatrix} \cdot \vec{D}_q
$$

$$
= \begin{bmatrix} {}^{0}J_{5,Trans} \end{bmatrix}_{3x5} \cdot \vec{D}_q
$$

$$ {}^0\vec{\omega}_5 \times {}^0\begin{bmatrix} \sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{1j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right) \\ \sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{2j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right) \\ \sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{3j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right) \end{bmatrix}_5 = \begin{bmatrix} \left[{}^0\vec{\omega}_5\right]_2\cdot\sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{3j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right)-\left[{}^0\vec{\omega}_5\right]_3\cdot\sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{2j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right) \\ \left[{}^0\vec{\omega}_5\right]_3\cdot\sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{1j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right)-\left[{}^0\vec{\omega}_5\right]_1\cdot\sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{3j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right) \\ \left[{}^0\vec{\omega}_5\right]_1\cdot\sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{2j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right)-\left[{}^0\vec{\omega}_5\right]_2\cdot\sum_{j=1}^{3}\left(\left[{}^0A_5\right]_{1j}\cdot\left[{}^5\vec{k}_{Tool}\right]_j\right) \end{bmatrix} \quad \text{(D.38)} $$

Mas

$$ {}^0\vec{\omega}_5 = \left[{}^0J_{Tool,R}\right]_{3x5}\cdot\vec{D}_q = \begin{bmatrix} \sum_{j=1}^{5}J_{4j}\cdot\left[\vec{D}_q\right]_j \\ \sum_{j=1}^{5}J_{5j}\cdot\left[\vec{D}_q\right]_j \\ \sum_{j=1}^{5}J_{6j}\cdot\left[\vec{D}_q\right]_j \end{bmatrix} \quad \text{(D.39)} $$

Então

$$ {}^0\begin{bmatrix} \frac{d}{dt}p_x \\ \frac{d}{dt}p_y \\ \frac{d}{dt}p_z \end{bmatrix}_{Tool} = \left[{}^0J_{5,Trans}\right]_{3x5}\cdot\vec{D}_q + \left(\left[{}^0J_{Tool,R}\right]_{3x5}\cdot\vec{D}_q\right)\times\left({}^0Rot_5\cdot{}^5\vec{k}_{Tool}\right) \quad \text{(D.40)} $$

$$ \begin{aligned} {}^0\begin{bmatrix} \frac{d}{dt}p_x \\ \frac{d}{dt}p_y \\ \frac{d}{dt}p_z \end{bmatrix}_{Tool} &= \left[\left[{}^0J_{5,Trans}\right]_{3x5} + \left[{}^0J_{Tool,R}\right]_{3x5}\otimes\left({}^0Rot_5\cdot{}^5\vec{k}_{Tool}\right)\right]\cdot\vec{D}_q \\ &= \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} \end{bmatrix}\cdot\vec{D}_q = \left[{}^0J_{Tool,Trans}\right]_{3x5}\cdot\vec{D}_q \end{aligned} \quad \text{(D.41)} $$

Em que o termo $\left[{}^0J_{Tool,R}\right]_{3x5}\otimes\left({}^0Rot_5\cdot{}^5\vec{k}_{Tool}\right)$ representa uma matriz cujas colunas são resultados dos produtos vetoriais de cada coluna de $\left[{}^0J_{Tool,R}\right]_{3x5}$ pelo vetor ${}^0\vec{k}_{Tool}={}^0Rot_5\cdot{}^5\vec{k}_{Tool}$. Os componentes da matriz Jacobiana do manipulador referentes ao movimento de translação da ferramenta em relação ao sistema da base são explicitados pelas equações genéricas (D.42) a (D.44)

$$J_{1j} = \left[{}^0 J_{5,Trans}\right]_{1j} + J_{5j} \cdot \left(\sum_{k=1}^{3} \left[{}^0 A_5\right]_{3k} \cdot \left[{}^5 \vec{k}_{Tool}\right]_k \right) - J_{6j} \cdot \left(\sum_{k=1}^{3} \left[{}^0 A_5\right]_{2k} \cdot \left[{}^5 \vec{k}_{Tool}\right]_k \right) \quad \text{(D.42)}$$

$$J_{2j} = \left[{}^0 J_{5,Trans}\right]_{2j} + J_{6j} \cdot \left(\sum_{k=1}^{3} \left[{}^0 A_5\right]_{1k} \cdot \left[{}^5 \vec{k}_{Tool}\right]_k \right) - J_{4j} \cdot \left(\sum_{k=1}^{3} \left[{}^0 A_5\right]_{3k} \cdot \left[{}^5 \vec{k}_{Tool}\right]_k \right) \quad \text{(D.43)}$$

$$J_{3j} = \left[{}^0 J_{5,Trans}\right]_{3j} + J_{4j} \cdot \left(\sum_{k=1}^{3} \left[{}^0 A_5\right]_{2k} \cdot \left[{}^5 \vec{k}_{Tool}\right]_k \right) - J_{5j} \cdot \left(\sum_{k=1}^{3} \left[{}^0 A_5\right]_{1k} \cdot \left[{}^5 \vec{k}_{Tool}\right]_k \right) \quad \text{(D.44)}$$

Em que o índice j é um número inteiro tal que $1 \leq j \leq 5$.

Desta forma, tendo os 15 elementos da matriz $\left[{}^0 J_{Tool,R}\right]_{3x5}$ calculados conforme mostra a eq. (D.36), podem-se calcular os elementos da matriz $\left[{}^0 J_{Tool,Trans}\right]_{3x5}$ utilizando as eqs. (D.42), (D.43) e (D.44). O Jacobiano completo do manipulador, referente ao movimento diferencial da tocha de soldagem genérica em relação ao sistema de coordenadas fixada na base do robô, é obtido concatenando-se as matrizes $\left[{}^0 J_{Tool,Trans}\right]_{3x5}$ e $\left[{}^0 J_{Tool,R}\right]_{3x5}$ conforme mostra a equação (D.45). Esta é a matriz que deve ser utilizada para se calcular o movimento diferencial cartesiano da tocha de soldagem, relativo ao sistema de coordenadas fixado na base do robô, como função do movimento diferencial realizado nas juntas.

$$\left[{}^0 J_{Tool}\right]_{6x5} = \begin{bmatrix} \left[{}^0 J_{Tool,Trans}\right]_{3x5} \\ \left[{}^0 J_{Tool,R}\right]_{3x5} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} \end{bmatrix} \quad \text{(D.45)}$$

D.4. JACOBIANO INVERSO DO MANIPULADOR

Como foi visto anteriormente, por meio do Jacobiano, é possível determinar o vetor de velocidades cartesianas do manipulador, utilizando as velocidades das juntas. Isso pode ser visto na eq.**Erro! Fonte de referência não encontrada.**. Entretanto, como a velocidade do lemento terminal é conhecida (velocidade desejada), é necessário determinar que velocidade cada junta deve ter para que a velocidade cartesiana desejada seja obtida.

Para tanto, é necessário determinar uma forma de calcular a velocidade das juntas do manipulador, em função da velocidade cartesiana do elemento terminal. Isso é feito por meio de outra matriz de derivadas a qual é chamada de Jacobiano Inverso do Manipulador. Essa definição é mostrada na equação (D.46).

$$\begin{bmatrix} d\theta_1 \\ d\theta_2 \\ dd_3 \\ d\theta_4 \\ d\theta_5 \end{bmatrix} = \mathbf{J}^{-1} \cdot \begin{bmatrix} dx \\ dy \\ dz \\ d\psi \\ d\theta \\ d\phi \end{bmatrix} = \begin{bmatrix} \frac{\partial\theta_1}{\partial x} & \frac{\partial\theta_1}{\partial y} & \frac{\partial\theta_1}{\partial z} & \frac{\partial\theta_1}{\partial \psi} & \frac{\partial\theta_1}{\partial \theta} & \frac{\partial\theta_1}{\partial \phi} \\ \frac{\partial\theta_2}{\partial x} & \frac{\partial\theta_2}{\partial y} & \frac{\partial\theta_2}{\partial z} & \frac{\partial\theta_2}{\partial \psi} & \frac{\partial\theta_2}{\partial \theta} & \frac{\partial\theta_2}{\partial \phi} \\ \frac{\partial d_3}{\partial x} & \frac{\partial d_3}{\partial y} & \frac{\partial d_3}{\partial z} & \frac{\partial d_3}{\partial \psi} & \frac{\partial d_3}{\partial \theta} & \frac{\partial d_3}{\partial \phi} \\ \frac{\partial\theta_4}{\partial x} & \frac{\partial\theta_4}{\partial y} & \frac{\partial\theta_4}{\partial z} & \frac{\partial\theta_4}{\partial \psi} & \frac{\partial\theta_4}{\partial \theta} & \frac{\partial\theta_4}{\partial \phi} \\ \frac{\partial\theta_5}{\partial x} & \frac{\partial\theta_5}{\partial y} & \frac{\partial\theta_5}{\partial z} & \frac{\partial\theta_5}{\partial \psi} & \frac{\partial\theta_5}{\partial \theta} & \frac{\partial\theta_5}{\partial \phi} \end{bmatrix} \cdot \begin{bmatrix} dx \\ dy \\ dz \\ d\psi \\ d\theta \\ d\phi \end{bmatrix} \tag{D.46}$$

A partir desse ponto, pode-se utilizar o mesmo desenvolvimento mostrado na secção 3.5.3, a partir da eq. (3.92).