Universidade Federal
do Rio de Janeiro
Escola Politécnica

CONTROLE DE UM MANIPULADOR ROBÓTICO LEVE DE 4-DOF COM SOFTWARE BASEADO EM ROS E QT

Luís Gustavo Oliveira Silva

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação, da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro de Controle e Automação.

Orientador: Fernando Cesar Lizarralde

Rio de Janeiro
Fevereiro de 2017

CONTROLE DE UM MANIPULADOR ROBÓTICO LEVE DE 4-DOF COM SOFTWARE BASEADO EM ROS E QT

Luís Gustavo Oliveira Silva

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

_____

Prof. Fernando Lizarralde, D.Sc.

_____

Prof. Antonio Candea Leite, D.Sc

_____

Prof. Afonso Celso Del Nero Gomes, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2017

Oliveira Silva, Luís Gustavo

Controle de um Manipulador Robótico leve de 4-DOF com software baseado em ROS e Qt/Luís Gustavo Oliveira Silva. – Rio de Janeiro: UFRJ/Escola Politécnica, 2017.

XIII, 82 p.: il.; 29, 7cm.

Orientador: Fernando Cesar Lizarralde

Projeto de graduação – UFRJ/Escola Politécnica/Curso de Engenharia de Controle e Automação, 2017.

Referências Bibliográficas: p. 73 – 75.

1. robótica. 2. controle. 3. cinemática. 4. servovisão. 5. visão computacional. I. , Fernando Cesar Lizarralde. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

Dedico esse trabalho a minha querida mãe Áurea a quem devo tudo que sou hoje.

Agradecimentos

Gostaria de agradecer primeiramente a minha mãe Áurea, pela educação, pelos valores e carinho que sempre me deu ao longo da vida.

Agradeço ao meu professor e orientador Fernando Lizarralde pelo suporte na elaboração deste trabalho e por exigir sempre o melhor de seus alunos. Agradeço aos membros da banca, professores Afonso Celso Del Nero Gomes e Antonio Candea Leite pela disposição em avaliar o trabalho e pelas sugestões feitas de forma a melhorá-lo.

Agradeço a todos os amigos do LEAD e LabCon, especialmente Alex Neves, por estar sempre disposto a ajudar, resolvendo nossos problemas de *software* num passe de mágica, mas sem se esquecer de Marco Xaud, Guilherme Carvalho, Renan Freitas, João Monteiro, Gabriel Loureiro, Mariana Rufino, Henrique Faria e outros com quem passamos momentos de trabalho mas também de descontração.

Resumo do Projeto de Graduação apresentado à POLI/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação.

CONTROLE DE UM MANIPULADOR ROBÓTICO LEVE DE 4-DOF COM SOFTWARE BASEADO EM ROS E QT

Luís Gustavo Oliveira Silva

Fevereiro/2017

Orientador: Fernando Cesar Lizarralde

Curso: Engenharia de Controle e Automação

Apresenta-se, neste projeto de graduação, a modelagem e controle cinemáticos de um manipulador de quatro graus de liberdade, com aplicação no manipulador TETIS, desenvolvido no projeto DORIS. Expõe-se as estratégias de controle desenvolvidas e implementadas através de abordagem cinemática, destacando-se controle proporcional com feedforward para rastreamento de trajetórias; controle por servovisão baseado em posição utilizando algoritmos de visão computacional e controle proporcional integral de força. Detalha-se o desenvolvimento e aquitetura de um software para controle de manipuladores robóticos utilizando Robot Operating System e Qt como *frameworks*. Por fim discute-se os resultados de simulação e experimentais obtidos.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Control and Automation Enginner.

CONTROL OF A 4-DOF LIGHTWEIGHT ROBOTIC MANIPULATOR WITH ROS AND QT BASED SOFTWARE

Luís Gustavo Oliveira Silva

February/2017

Advisor: Fernando Cesar Lizarralde

Course: Control and Automation Engineering

In this graduation project, is presented the kinematic modeling and control of a manipulator with four degrees of freedom, with application on the TETIS manipulator, developed by the DORIS project. Control strategies based on the kinematic approach were developed and implemented, highlighting the proportional with feedforward controller for trajectory tracking, the position based visual servoing using computer vision algorithms and force control with a proportional integral controller.

The development and architecture of a software to control robotic manipulators using Robot Operating System and Qt as frameworks is shown. Lastly, simulation and experimental results are discussed.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desde a década de 60, robôs tem sido utilizados em ambientes industriais. Manipuladores robóticos foram capazes de aumentar a produtividade, a eficiência e garantir um maior controle de qualidade dos processos. Além de serem capazes de realizar tarefas repetitivas em uma linha de montagem, muitos manipuladores o fazem com maior precisão e rapidez que um ser humano.

Recentemente além de linhas de produção da industria, a robótica tem encontrado aplicação em instalações *offshore* de óleo e gás. É de grande interesse utilizá-los para realizar tarefas onde a presença humana torna-se difícil, arriscada ou até mesmo impossível. Muitas empresas já tem utilizado soluções automatizadas tanto em ambientes submersos quanto acima do nível do mar. Braços robóticos tem sido de grande importância para executar tarefas que exigem interação mais complexa com o ambiente. Com isso em mente, foi desenvolvido um manipulador leve para o DORIS, robô guiado por trilhos para monitoração, inspeção e supervisão de ambientes não submersos de plataformas de petróleo (Xaud 2016).

1.1 Motivação

1.1.1 Robótica em instalações *offshore*

O setor de óleo e gás tem crescido constantemente, com a produção sendo gradualmente expandida até locais não explorados e inacessíveis. As condições de trabalho encontradas em geral nesses locais incluem chuvas pesadas, vento, temperaturas extremas, ambiente explosivo, corrosivo e produtos químicos tóxicos. Nesse cenário, empresas de óleo e gás tem investido no desenvolvimento de tecnologias que permitam a produção nesses ambientes. Com isso, a aplicação de automação de processos e robótica tem crescido rapidamente. Robôs tem sido utilizados para tarefas de inspeção, manutenção, reparo e supervisão de instalações *offshore*, aumentando a eficiência dos processos, a segurança e saúde dos funcionários, além de diminuir os

custos de operação e logística.

O uso de robôs instalações de óleo e gás tem diversas vantagens (Skourup & Pretlove 2009). Pode reduzir o custo com manutenção de múltiplos sensores ao longo instalação e substituir humanos na realização de tarefas repetitivas, especialmente aquelas realizadas em ambientes perigosos, confinados ou prejudiciais a saúde.

Hoje a maioria dos robôs utilizados no setor de óleo e gás é composta de Veículos Operados Remotamente (ROVs[1]) e Veículos Submarinos Autônomos (AUVs [2]), utilizados para operações submarinas como mapeamento do solo oceânico, perfuração, inspeção e reparo de equipamento submarino, tubulações, âncoras, entre outros. No entanto, pesquisas atuais tem focado também em aplicações de robótica na área *topside* das plataformas, buscando realizar tarefas como:

- Monitorar indicadores de processos.
- Supervisionar o bom funcionamento máquinas e equipamentos.
- Detectar o vazamento de gás ou fluido, anomalias, fogo e fumaça.
- Manipular válvulas, botões e alavancas.
- Supervisão de pessoal não autorizado e objetos.

(a) MIMROex (b) ARTIS

Figura 1.1: MIMROex e ARTIS

Dentre os robôs para áreas *topside* de plataformas, podemos mencionar o MIMROex, desenvolvido pela Fraunhoffer IPA para tarefas de inspeção e monitoração (Bengel & Pfeiffer 2007), mostrado na Figura 1.1a.

Robôs como esse movem-se ao longo da plataforma e possuem muitos graus de liberdade, tornando seu projeto de controle bastante complexo. Portanto, foram propostas soluções como o ARTIS, *Autonomous Railguided Tank Inspection System* (GmbH 2011) da DFKI (*German Research Center for Artificial Intelligence*). Esse

[0]ROV do inglês Remotely Operated underwater Vehicle
[0]AUV, do inglês Autonomous Underwater Vehicle

robô se move de forma restrita a um trilho que é projetado de forma a levá-lo até todos os pontos de interesse, isso diminui a complexidade do projeto.

O robô Sawyer™, mostrado na Figura 1.2, é a segunda geração de robôs de alto desempenho desenvolvidos pela Rethink Robotics®, é projetado par trabalhar lado a lado com humanos (Ackerman & Guizzo 2015). Algumas de suas especificações são mostradas na tabela

Tabela 1.1: Especificações do Rethink Robotics® Sawyer™

Especificação	Descrição
Peso	19Kg
Payload	4Kg
Atuação	*Harmonic Drive* com *encoder* óptico.
Sensor de força	Sensores de força de alta resolução em cada junta.
Visão	Câmera Cognex com amplo compo de visão e fonte de luz no punho para aplicações de precisão.
Software	Plataforma Intera

Figura 1.2: Rethink Robotics® Sawyer™

Dentre os seus diferenciais destaca-se seu peso de apenas $19Kg$, atuadores *Harmonic Drive* com encoder óptico, que garantem maior precisão e repetibilidade, e o *software* Intera baseado em ROS(Robot Operating System). Essas características tornam o Sawyer™um sistema promissor para aplicações de robótica em instalações *offshore*.

1.1.2 DORIS

O projeto DORIS, desenvolvido pelo laboratório LEAD-GSCAR, da Universidade Federal do Rio de Janeiro, introduz um sistema robótico onde vagões guiados por trilhos carregam diversas câmeras, sensores e dispositivos para monitorar e inspecionar diferentes áreas e equipamento na parte *topside* de plataformas (Freitas et al. 2015, Galassi et al. 2014, Nunes et al. 2013).

As tarefas desse robô consistem principalmente em: monitorar perfis de temperatura utilizando câmeras térmicas infravermelhas, supervisão de pessoal não autorizado, detecção de anomalias sonoras utilizando microfones, detecção de vazamentos de gás com sensores de hidrocarbonetos, inspeção de padrões de vibração de maquinário crítico, interação com interfaces *touchscreen* na plataforma, processamento de dados coletados, armazenamento e transmissão de áudio e vídeo em tempo real.

A interação com o robô é feita através de um *software* chamado de RobotGUI, também desenvolvido pela equipe do LEAD-GSCAR. Através dele o operador é capaz de visualizar dados dos diversos sensores, reproduzir a transmissão de áudio e vídeo, enviar comandos de controle e configurar parâmetros, através de uma interface gráfica customizável.

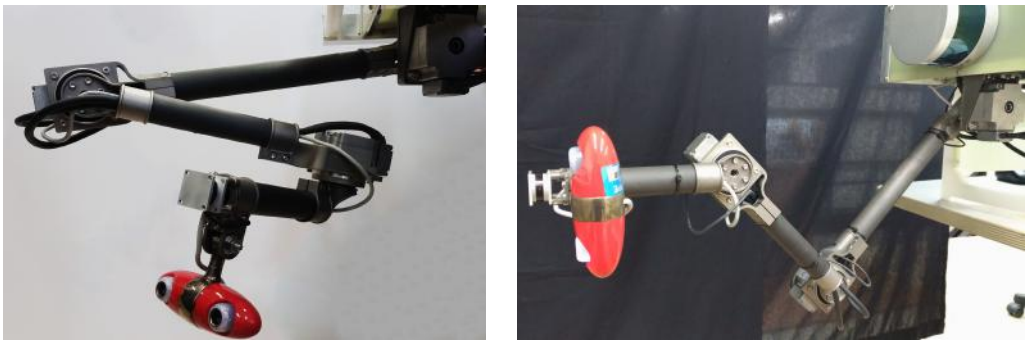
Figura 1.3: DORIS em operação no CENPES

1.1.3 Manipulador TETIS

Considerando as tarefas mencionadas foi proposta a adição de um manipulador leve, de modo a extender o espaço de trabalho do robô. Com esse manipulador, pretende-se solucionar os problemas de

- mover a câmera acoplada ao efetuador, de modo a posicioná-la melhor ao longo da instalação,
- posicionar um sensor de vibração corretamente sobre a superfície de um equipamento na plataforma e
- interagir com *touchscreens*.

Foi dado a esse manipulador o nome TETIS. O projeto mecânico permite que ele assuma configurações de juntas diferentes, como mostra a Figura 1.4.



(a) Configuração 1 (b) Configuração 2

Figura 1.4: Manipulador TETIS

1.1.4 Robot Operating System

Com sua primeira versão em 2007, o *framework* ROS (Robot Operating System) tem ganhado grande aceitação da comunidade para construção de aplicações para robôs. Consiste em um conjunto de bibliotecas e ferramentas que auxiliam no desenvolvimento de *software* para robótica. Essa não é uma tarefa simples, pois o cada vez mais o escopo dos projetos aumenta, englobando algoritmos de controle, visão computacional, percepção e tomada de decisão, entre outros. Dentre as principais filosofias adotadas, destaca-se a modularidade. Muitos projetos tem a tendência de englobar código específico para o robô em questão, de modo que torna-se difícil reaproveitar algorítimos úteis. O ROS encoraja a comunidade de contribuidores a desenvolver pacotes que possam ser reutilizados em outros projetos.

Processos que utilizam ROS são chamados de nós e comunicam entre si através de uma topologia ponto-a-ponto, podendo ser executados na mesma máquina ou em diferentes computadores em uma LAN. Dessa forma, os diferentes componentes necessários para o *software* de um robô podem ser encapsulados em nós, o que garante uma maior possibilidade de reaproveitamento, facilidade de alteração e versatilidade de execução.

1.2 Objetivos

A partir do projeto mecânico e eletrônico do manipulador finalizado, segue a etapa de modelagem e implementação de estratégias de controle. Nesse contexto, insere-se este trabalho. Utilizando a configuração do manipulador mostrada na Figura 1.4b, alternativa àquela abordada em (Xaud 2016), busca-se:

- Elaborar o modelo cinemático e implementar estratégias de controle.
- Utilizar a câmera para controle por servovisão onde o alvo é um marcador em algum ponto de interesse, como por exemplo uma máquina a ser inspecionada.

- Com o sensor de força montado no efetuador do manipulador, realizar controle de força sobre uma superfície de restrição suave de geometria conhecida.
- A partir do RobotGUI, que já interage com os outros dispositivos do DORIS, desenvolver um *software* para controle de manipuladores robóticos. O uso do ROS proporciona uma maior versatilidade e facilidade para incorporação de novas funcionalidades, em comparação com abordagens tradicionais.
- Utilizar *Julia Language*, uma linguagem de programação de alto nível e alta performance orientada para computação científica, para permitir que o algoritmo de controle seja modificado em tempo de execução. Essa linguagem possui mecanismos de compilação em tempo de execução, o que dá mais flexibilidade ao programa, sem perder desempenho.

1.3 Organização do trabalho

No primeiro capítulo o trabalho foi contextualizado, descrevendo os objetivos do projeto DORIS e do manipulador TETIS. No segundo capítulo são mostrados conceitos de robótica necessários para a modelagem e controle de manipuladores robóticos. No capítulo 3 esses conceitos são aplicados ao manipulador TETIS. No capítulo 4 é descrito o hardware do robô DORIS. No capítulo 5 são abordados detalhes das ferramentas e da arquitetura utilizada para a implementação do *software*. No capítulo 6, são mostrados e discutidos os resultados de simulação e de testes experimentas para os diferentes modos de controle implementados. Por último, no capítulo 7 são apresentadas as conclusões sobre o trabalho e pontos para serem abordados em trabalhos futuros.

Capítulo 2

Modelagem e Controle Cinemático

Neste capitulo serão abordados os conceitos necessários para modelagem e controle de manipuladores robóticos, focando no que foi utilizado neste projeto, para implementação no manipulador 4-DOF chamado de TETIS. Os tópicos tratados aqui podem ser encontrados em (Corke 2011, Siciliano et al. 2009).

2.1 Posição e Orientação de um Corpo Rígido

Um corpo rígido é descrito no espaço por sua posição e orientação (pose) em relação a um sistema de coordenadas de referência. Escolhe-se um ponto do corpo e afixa-se um sistema de coordenadas. Denota-se como $\bar{E}$ um sistema de coordenadas ortonormal com $\vec{x}$, $\vec{y}$ e $\vec{z}$ como vetores unitários.

Sejam o sistema de coordenadas inercial $\bar{E}_a = [\vec{x}_a\ \vec{y}_a\ \vec{z}_a]$ e o sistema de coordenadas do corpo $\bar{E}_b = [\vec{x}_b\ \vec{y}_b\ \vec{z}_b]$, como mostra a Figura 2.1.

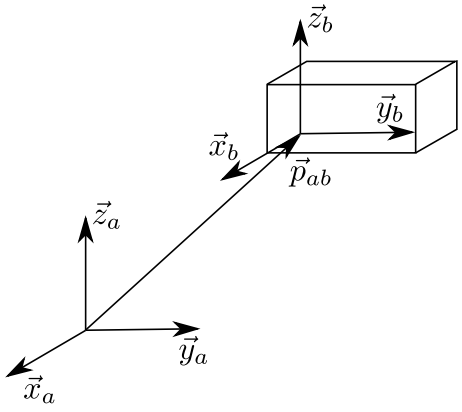


Figura 2.1: Sistemas de coordenadas de referência $\bar{E}_a$ e do corpo $\bar{E}_b$.

A **posição** da origem O_b do sistema de coordenadas do corpo rígido em relação ao sistema de coordenadas inercial é dada por

$$p_{ab} = p_{ab_x}\vec{x}_a + p_{ab_y}\vec{y}_a + p_{ab_z}\vec{z}_a, \tag{2.1}$$

onde p_{b_x}, p_{b_y} e p_{b_z} denotam as componentes do vetor $p_b \in \mathbb{R}^3$ representadas no sistema de coordenadas $\bar{E}_a$. Pode ser escrita de forma compacta como um vetor (3×1)

$$p_{ab} = \begin{bmatrix} p_{ab_x} \\ p_{ab_y} \\ p_{ab_z} \end{bmatrix} \tag{2.2}$$

As coordenadas de $\vec{x}_b$, $\vec{y}_b$ e $\vec{z}_b$ representadas no sistema de coordenadas $\bar{E}_a$ são dadas por x_{ab}, y_{ab} e z_{ab} em

$$x_{ab} = \bar{E}_a^* \vec{x}_b \tag{2.3}$$

$$y_{ab} = \bar{E}_a^* \vec{y}_b \tag{2.4}$$

$$z_{ab} = \bar{E}_a^* \vec{z}_b \tag{2.5}$$

onde $\bar{E}^* = [\vec{e}_1 \cdot \ \ \vec{e}_2 \cdot \ \vec{e}_3 \cdot]$ denota o operador adjunto de $\bar{E}$.

A partir das equações (2.3), (2.4) e (2.5), podemos escrever:

$$\vec{x}_b = \bar{E}_a x_{ab} \tag{2.6}$$

$$\vec{y}_b = \bar{E}_a y_{ab} \tag{2.7}$$

$$\vec{z}_b = \bar{E}_a z_{ab}, \tag{2.8}$$

portanto,

$$\bar{E}_b = [\bar{E}_a x_{ab} \ \ \bar{E}_a y_{ab} \ \ \bar{E}_a z_{ab}] = \bar{E}_a [x_{ab} \ \ y_{ab} \ \ z_{ab}] = \bar{E}_a R_{ab} \tag{2.9}$$

A matriz R_{ab} é chamada de matriz de rotação e define a **orientação** do corpo rígido.

$$R_{ab} = \begin{bmatrix} x_{ab} & y_{ab} & z_{ab} \end{bmatrix} \tag{2.10}$$

onde $x_{ab} \in \mathbb{R}^3$, $y_{ab} \in \mathbb{R}^3$ e $z_{ab} \in \mathbb{R}^3$ são as componentes do sistema de coordenadas $\bar{E}_b$ no sistema de coordenadas $\bar{E}_a$, ou seja:

$$R_{ab} = \begin{bmatrix} \vec{x}_a \cdot \\ \vec{y}_a \cdot \\ \vec{z}_a \cdot \end{bmatrix} \begin{bmatrix} \vec{x}_b & \vec{y}_b & \vec{z}_b \end{bmatrix} = \begin{bmatrix} (\vec{x}_a \cdot \vec{x}_b) & (\vec{x}_a \cdot \vec{y}_b) & (\vec{x}_a \cdot \vec{z}_b) \\ (\vec{y}_a \cdot \vec{x}_b) & (\vec{y}_a \cdot \vec{y}_b) & (\vec{y}_a \cdot \vec{z}_b) \\ (\vec{z}_a \cdot \vec{x}_b) & (\vec{z}_a \cdot \vec{y}_b) & (\vec{z}_a \cdot \vec{z}_b) \end{bmatrix} \tag{2.11}$$

2.1.1 Representação de um vetor

Um vetor $\vec{p}$ pode ser representado como $(p)_a = [p_x \ \ p_y \ \ p_z]$ no sistema de coordenadas $\bar{E}_a$ e como $(p)_b = [p'_x \ \ p'_y \ \ p'_z]$ no sistema de coordenadas $\bar{E}_b$. A matriz de rotação R_{ab} representa a transformação de coordenadas de $\vec{p}$ em $\bar{E}_b$ para suas coordenadas em $\bar{E}_a$, através de

$$(p)_a = R_{ab}(p)_b. \tag{2.12}$$

2.1.2 Transformações Homogêneas

A posição de um corpo rígido é expressa em função da posição de um ponto no corpo rígido com respeito ao sistema de coordenadas de referência (translação), enquanto que sua orientação é expressa em termos das componentes dos vetores unitários do sistema de coordenadas do corpo em relação ao sistema de coordenadas de referência (rotação).

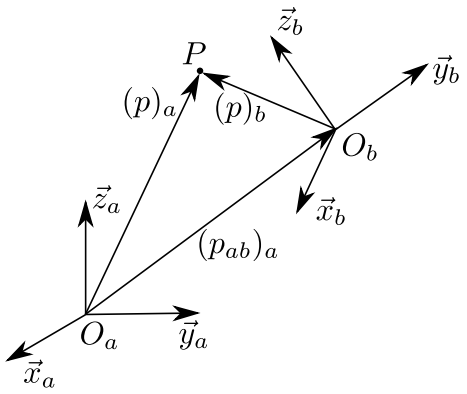


Figura 2.2: Representação de um ponto P em diferentes sistemas de coordenadas.

Seja $(p)_a$ o vetor de coordenadas de um ponto P no espaço, em relação a um sistema de coordenadas de referência $\bar{E}_a = [\vec{x}_a\ \vec{y}_a\ \vec{z}_a]$. Considerando um sistema de coordenadas $\bar{E}_b = [\vec{x}_b\ \vec{y}_b\ \vec{z}_b]$, seja $(p_{ab})_a$ o vetor de coordenadas descrevendo a origem do sistema de coordenadas $\bar{E}_b$ com relação sistema de coordenadas $\bar{E}_a$. A matriz de rotação R_{ab} descreve a orientação do sistema de coordenadas $\bar{E}_b$ em relação a $\bar{E}_a$. Seja, também, $(p)_b$ o vetor de coordenadas do ponto P em relação ao sistema de coordenadas $\bar{E}_b$. A posição do ponto P pode ser representada no sistema de coordenadas de referência como

$$(p)_a = (p_{ab})_a + R_{ab}(p)_b. \tag{2.13}$$

A Equação (2.13) representa a transformação de coordenadas (translação e rotação) de um vetor, de um sistema de coordenadas para outro.

De forma a obter uma representação compacta da relação entre as coordenadas de um mesmo ponto em dois sistemas de coordenadas diferentes, introduz-se a representação homogênea de um vetor $p \in \mathbb{R}^3$ como $\tilde{p} \in \mathbb{R}^4$, formado adicionando um quarto componente unitário[1], ou seja

$$\tilde{p} = \begin{bmatrix} p \\ 1 \end{bmatrix}. \tag{2.14}$$

Adotando essa representação a transformação de coordenadas pode ser escrita atra-

[1] A representação homogênea de um vetor n-dimensional é abordada no Apêndice A.

vés da matriz (4×4)

$$T_{ab} = \begin{bmatrix} R_{ab} & (p_{ab})_a \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{2.15}$$

Portanto a transformação de coordenadas de um vetor de $\bar{E}_b$ para $\bar{E}_a$ pode ser expressa de forma compacta por uma única matriz, como

$$(\tilde{p})_a = T_{ab}(\tilde{p})_b \tag{2.16}$$

enquanto que a transformação de $\bar{E}_a$ para $\bar{E}_b$ é descrita pela matriz T_{ba} que satisfaz a equação

$$(\tilde{p})_b = T_{ba}(\tilde{p})_b = (T_{ab})^{-1}(\tilde{p})_a. \tag{2.17}$$

A matriz T_{ba} pode ser expressa como

$$T_{ba} = \begin{bmatrix} R_{ba} & -R_{ba}(p_{ab})_a \\ 0_{1\times 3} & 1 \end{bmatrix}. \tag{2.18}$$

2.2 Cinemática Direta

Um manipulador robótico é composto de uma série de corpos rígidos denominados *elos* conectados através de *juntas*. Juntas podem ser:

- Revolução
- Prismática

Essa estrutura é chamada de cadeia cinemática. Um extremo da cadeia é fixado a base e o outro ao efetuador. Nesse texto serão abordadas apenas cadeias cinemáticas abertas, ou seja, aquelas em que existe apenas uma sequência de elos conectando os dois extremos da cadeia. Cada junta acrescenta um grau de liberdade (DOF), ao qual está associado a uma variável de junta. No caso de uma junta de revolução um ângulo e no caso de uma junta prismática um deslocamento. O objetivo da cinemática direta é calcular a posição e orientação do efetuador em função das variáveis das juntas.

Uma cadeia cinemática aberta é constituída por $n + 1$ elos numerados de 0 a n, onde o Elo 0 é fixado a base por convenção. O método utilizado consiste em definir um sistema de coordenadas associado a cada elo e calcular a transformação homogênea entre elos consecutivos. Em seguida a transformação do n-ésimo sistema de coordenadas pode ser obtida de forma recursiva como

$$T_{0n}(q) = T_{01}(q_1)T_{12}(q_2)\ldots T_{n-1,n}(q_n) \tag{2.19}$$

onde $T_{i-1,i}(q_i)$ denota a transformação homogênea do sistema de coordenadas solidário ao elo $i-1$ àquele solidário ao elo i e $q \in \mathbb{R}^n$ é o vetor de variáveis de junta, com n sendo o número de juntas.

Logo a transformação homogênea do efetuador final com respeito a base é dada por

$$T_{be}(q) = T_{b0}T_{0n}(q)T_{ne} \tag{2.20}$$

2.2.1 Convenção Denavit-Hartenberg

Para calcular a cinemática direta para uma manipulador de cadeia cinemática aberta de acordo com a equação (2.19) um método sistemático foi definido para obter a relação entre a posição e orientação de dois elos consecutivos. A convenção Denavit-Hartenberg especifica um conjunto de regras sobre como definir os sistemas de coordenadas de cada elo.

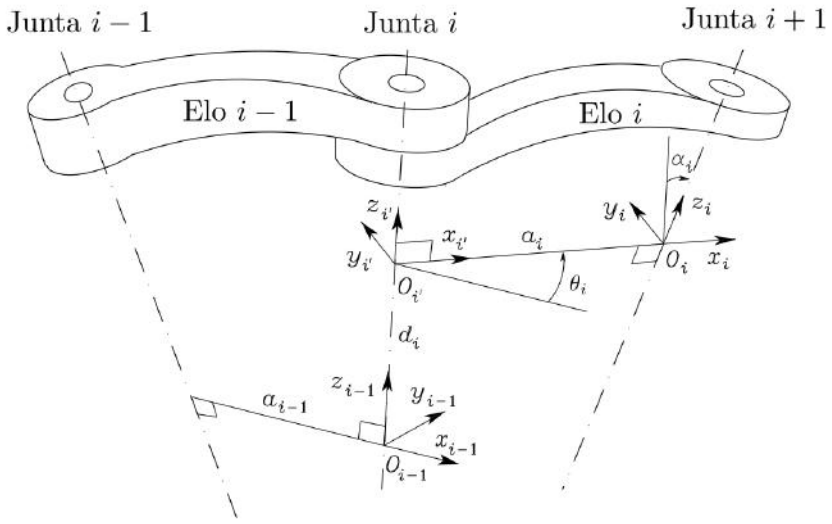


Figura 2.3: Parâmetros Denavit-Hartenberg.

Seja o $\vec{h}_i$ eixo de rotação da junta que conecta o elo $i-1$, ao elo i, então:

- Escolher o eixo $\vec{z}_i$ ao longo do eixo $\vec{h}_{i+1}$.
- Colocar a origem O_i na interseção do eixo $\vec{z}_i$ com a normal comum entre os eixos $\vec{z}_{i-1}$ e $\vec{z}_i$
- Escolher $\vec{x}_i$ ao longo da normal comum aos eixos $\vec{z}_{i-1}$ e $\vec{z}_i$, com direção da junta i para a junta $i+1$.
- O eixo $\vec{y}_i = \vec{z}_i \times \vec{x}_i$ é escolhido de forma a completar o sistema de coordenadas.

Essa convenção resulta em uma definição não única do sistema de coordenadas nos seguintes casos:

- Para o sistema de coordenadas 0, somente a direção do eixo $\vec{z}_0$ é especificada, portanto a escolha de O_0 e $\vec{x}_0$ é arbitrária.
- Para o sistema de coordenadas n, como não existe junta $n+1$, $\vec{z}_n$ não está definido, mas $\vec{x}_n$ deve ser normal ao eixo $\vec{z}_{n-1}$. Tipicamente escolhe-se $\vec{z}_n$ alinhado com $\vec{z}_{n-1}$.
- Quando dois eixos consecutivos são paralelos, a normal comum entre eles não é definida de forma única. Tipicamente escolhe-se O_i na junta $i+1$
- Quando dois eixos consecutivos se interceptam, direção de $\vec{x}_i$ é normal e o sentido é arbitrário. Escolhe-se O_i na intersecção.
- Quando a junta i é prismática a direção de $\vec{z}_{i-1}$ é arbitrária.

Após determinados os sistemas de coordenadas, é possível determinar a posição e orientação de um referencial em relação ao outro através dos seguintes parâmetros:

- a_i distância entre $\vec{z}_{i-1}$ e z_i ao longo de $\vec{x}_i$
- α_i ângulo entre $\vec{z}_{i-1}$ e $\vec{z}_i$ ao redor de $\vec{x}_i$
- d_i distância entre $\vec{x}_{i-1}$ e $\vec{x}_i$ ao longo de $\vec{z}_{i-1}$
- θ_i ângulo entre $\vec{x}_{i-1}$ e $\vec{x}_i$ ao redor de $\vec{z}_{i-1}$

A orientação de um sistema de coordenadas i em relação ao anterior é dada por uma rotação de θ_i em torno de $\vec{z}$ seguida de uma rotação em torno de $\vec{x}$ de α_i considerando o sistema de coordenadas corrente.

$$R_{i-1,i} = R_z(\theta_i)R_x(\alpha_i) \tag{2.21}$$

A translação é obtida representando as distâncias no sistema de coordenadas $i-1$:

$$\vec{p}_{i-1,i} = d_i\vec{z}_{i-1} + a_i\vec{x}_i \tag{2.22}$$

$$(\vec{p}_{i-1,i})_{i-1} = d_i(\vec{z}_{i-1})_{i-1} + a_i(\vec{x}_i)_{i-1} \tag{2.23}$$

$$(\vec{p}_{i-1,i})_{i-1} = d_i(\vec{z}_{i-1})_{i-1} + a_iR_{i-1,i}(\vec{x}_i)_i \tag{2.24}$$

As duas informações podem ser representadas de uma forma mais compacta como transformação homogênea:

$$T_{i-1,i} = \begin{bmatrix} R_{i-1,i} & (\vec{p}_{i-1,i})_{i-1} \\ 0_{1\times3} & 1 \end{bmatrix} \tag{2.25}$$

que em função dos parâmetros fica:

$$T_{i-1,i} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

onde utiliza-se a notação s_i para $\sin\theta_i$ e c_i para $\cos\theta_i$. No caso de $\sin(\theta_i + \theta_j)$ denota-se s_{ij}.

2.2.2 Espaço das Juntas e Espaço Operacional

Para que o efetuador final de um manipulador realize alguma tarefa é necessário atribuir uma posição e orientação desejada, que pode ser função do tempo (trajetória). Surge então o problema de representar posição e orientação.

Para descrever a posição utiliza-se as coordenadas cartesianas. Para a orientação adota-se uma representação mínima (ângulos de Euler), descrevendo a rotação do efetuador em relação ao sistema de coordenadas da base. Portanto é possível descrever a *pose* do efetuador com relação a base através do vetor

$$x_{be} = \begin{bmatrix} p_{be} \\ \phi_{be} \end{bmatrix} \tag{2.27}$$

onde $p_{be} \in \mathbb{R}^3$ descreve a posição e $\phi_{be} \in \mathbb{R}^3$ é uma representação mínima da orientação.

O *espaço das juntas* denota o espaço em que o vetor $(n \times 1)$ das variáveis das juntas

$$q = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix} \tag{2.28}$$

é definido. Se a junta é de revolução utiliza-se $q_i = \theta_i$, se é prismática $q_i = d_i$.

2.3 Cinemática Diferencial

2.3.1 Jacobiano Analítico

Quando a posição e orientação do efetuador são dadas em função de um número mínimos de parâmetros no espaço operacional é possível computar o Jacobiano pela diferenciação das equações da cinemática direta em função das variáveis das juntas. Para isso utiliza-se a técnica analítica.

Seja $p_{be} \in \mathbb{R}^3$ a posição do sistema de coordenadas do efetuador representada no sistema de coordenadas da base. O vetor $\dot{p}_{be}$ é portanto a velocidade de translação, ou linear.

$$\dot{p}_{be} = \frac{\partial p_{be}}{\partial q}\dot{q} = J_{ap}(q)\dot{q} \tag{2.29}$$

onde $J_{ap} \in \mathbb{R}^{3\times n}$ denota o Jacobiano analítico de posição.

A derivada no tempo $\dot{\phi}_{be}$ não é igual a velocidade angular, no entanto, conhecida a função $\phi_{be}(q)$:

$$\dot{\phi}_{be} = \frac{\partial \phi_{be}}{\partial q}\dot{q} = J_{a\phi}(q)\dot{q} \tag{2.30}$$

onde $J_{a\phi} \in \mathbb{R}^{3\times n}$ denota o Jacobiano analítico de orientação. Assim, a cinemática diferencial pode ser obtida como:

$$\dot{x}_{be} = \begin{bmatrix} \dot{p}_{be} \\ \dot{\phi}_{be} \end{bmatrix} = \begin{bmatrix} J_{a_p}(q) \\ J_{a_\phi}(q) \end{bmatrix} \dot{q} = J_a(q)\dot{q} \tag{2.31}$$

onde $J_a \in \mathbb{R}^{6\times n}$.

2.4 Controle Cinemático

O estratégia de controle cinemático pode ser aplicada quando considera-se que a dinâmica do manipulador pode ser desprezada. Essa hipótese se sustenta quando as seguintes premissas são válidas:

- Elevados fatores de redução nas juntas.
- Baixas velocidades na realização das tarefas.
- Existência uma malha de controle de velocidade de alto desempenho em cada junta.

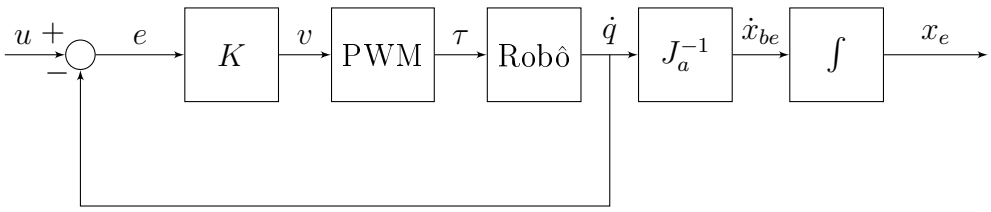


Figura 2.4: Diagrama de Blocos: Malha de Controle de Velocidade a nível de juntas.

A maioria dos manipuladores possui uma malha de controle de velocidade em nível de juntas como na Figura 2.4. Logo, para um controle de alto ganho temos que:

$$u \approx \dot{q}$$

Portanto é possível implementar o controle cinemático segundo o diagrama da Figura 2.5, considerando que o manipulador tem 6 juntas, i.e. $n = 6$. Na Figura 2.5 $k(\cdot)$ denota a cinemática direta do manipulador.

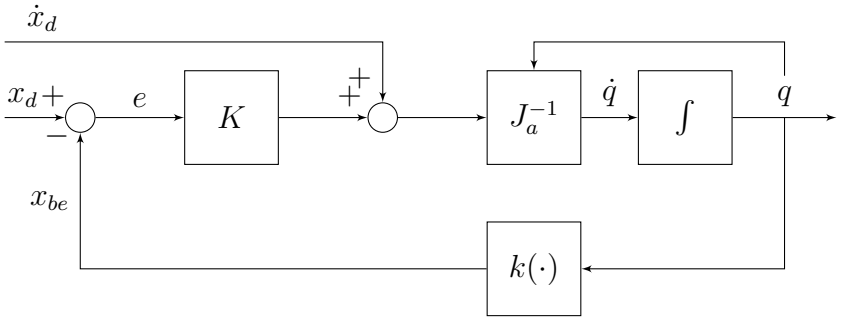


Figura 2.5: Diagrama de Blocos: Controle cinemático proporcional com feedforward

Se $x_{be} \in \mathbb{R}^6$ é uma representação da posição e orientação e x_d o valor desejado nessa representação, seja o erro no espaço operacional:

$$e = x_d - x_{be} \tag{2.32}$$

Derivando em relação ao tempo

$$\dot{e} = \dot{x}_d - \dot{x}_{be} \tag{2.33}$$

podemos escrever a partir da equação 2.31:

$$\dot{e} = \dot{x}_d - J_a(q)\dot{q} \tag{2.34}$$

Sendo $x_d(t)$ uma trajetória desejada, deseja-se que x_{be} atinja $x_d(t)$ em $t \to \infty$. A entrada de controle para o sistema é um valor de $u = \dot{q}$, logo, assumindo que $J_a(q)$ é quadrada e não singular, a escolha da lei de controle

$$u = J_a^{-1}(q)\bar{u} \tag{2.35}$$

leva ao sistema linear:

$$\dot{e} = \dot{x}_d - \bar{u} \tag{2.36}$$

Se for escolhido $\bar{u}$:

$$\bar{u} = \dot{x}_d + K_t(x_d - x_{be}) \tag{2.37}$$

obtem-se a seguinte dinâmica para o erro

$$\dot{e} + K_t e = 0 \tag{2.38}$$

onde $K_t = k_t I$. Se $k_t > 0$ o sistema é assintoticamente estável, com $e \to 0$ quando

$t \to \infty$.

2.5 Servovisão

A tarefa proposta na Servovisão é controlar a posição e orientação do efetuador do manipulador, em relação a um alvo, usando características visuais extraídas de uma imagem. A câmera pode ser carregada pelo manipulador (montada no efetuador) ou colocada em um ponto fixo, observando tanto o efetuador como o alvo.

Primeiramente é importante entender os princípios de formação de imagem através de câmeras (Corke 2011).

2.5.1 Transformação de Perspectiva

Em visão computacional geralmente é utilizado o modelo de perspectiva central mostrado na Figura 2.6, onde f é a distância focal da lente. Seja $P = [X\ Y\ Z]^T$ as coordenadas de um ponto expresso no sistema de coordenadas da câmera e $p = [x\ y]^T$ as coordenadas projetadas no plano da imagem por

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z} \tag{2.39}$$

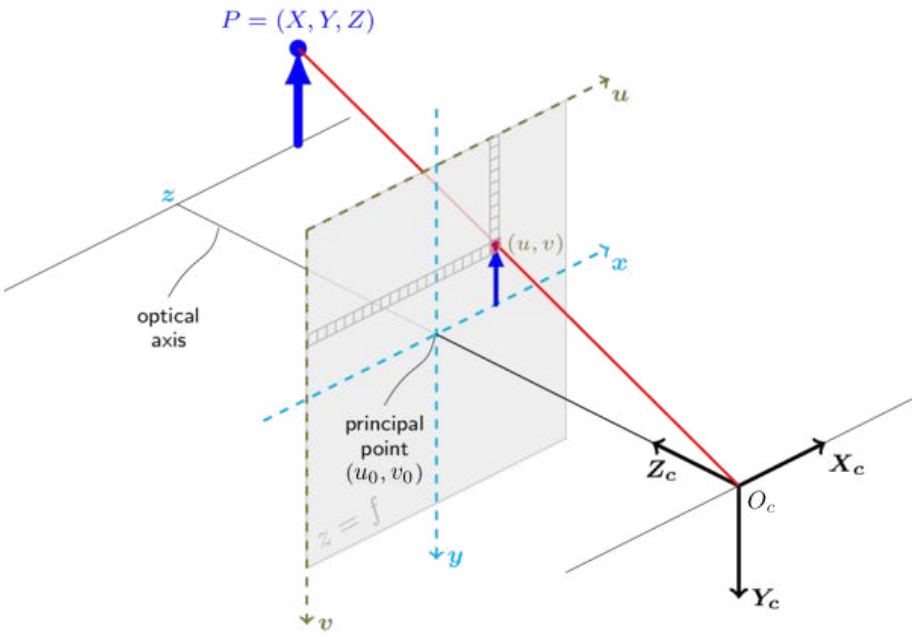


Figura 2.6: Modelo da câmera (disponível em (*Camera Calibration and 3D Reconstruction* 2016))

É possivel expressar o ponto no plano da imagem em coordenadas homogêneas (conforme definido no Apêndice A) na forma $\tilde{p} = [x'\ y'\ z']$ onde

$$x' = \frac{fX}{z'} \qquad y' = \frac{fY}{z'} \qquad z' = 1 \tag{2.40}$$

Escrevendo em forma matricial:

$$\tilde{p} = \frac{1}{Z} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.41}$$

Conforme mencionado no Apêndice A as coordenadas homogêneas tornam indiferente um fator de escala, ou seja $\tilde{p}_1$ e $\tilde{p}_2 = \alpha\tilde{p}_1$ com $\alpha \neq 0$ representam o mesmo ponto Euclidiano, cujas coordenadas no plano da imagem são dadas por:

$$x = \frac{x'}{z'} \qquad y = \frac{y'}{z'}$$

O ponto P expresso no sistema de coordenadas inercial pode ser representado em coordenadas homogêneas como $(\tilde{P})_C = [X\ Y\ Z\ 1]^T$ no referencial da câmera. A projeção de perspectiva pode ser escrita em forma linear, considerando que o fator de escala é indiferente, como

$$\tilde{p} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} (\tilde{P})_C. \tag{2.42}$$

A matriz pode ser fatorada como

$$\tilde{p} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} (\tilde{P})_C \tag{2.43}$$

onde a primeira matriz é chamada de matriz da câmera e a segunda de matriz de projeção.

Uma imagem digital pode ser interpretada como uma matriz bidimensional de *pixels*. As coordenadas de um ponto são expressas, portanto, em *pixels* como um vetor de números inteiros $[u\ v]$. As coordenadas no plano da imagem se relacionam com as coordenadas em *pixels* por

$$u = \frac{x}{\rho_w} + u_0 \qquad v = \frac{y}{\rho_h} + v_0 \tag{2.44}$$

onde ρ_w e ρ_h são a largura e altura de cada *pixel* respectivamente e $[u_0\ \ v_0]$ são as coordenadas do ponto em que o eixo ótico intercepta o plano da imagem.

Reescrevendo a equação (2.42) adicionando uma matriz K de parâmetros

$$\tilde{p}_{px} = \begin{bmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} (\tilde{P})_C \tag{2.45}$$

onde $\tilde{p}_{px} = [u' \quad v' \quad w']$ são as coordenadas homogêneas em pixels do ponto P. As coodenadas cartesianas no plano da imagem são dadas por

$$u = \frac{u'}{w'} \qquad v = \frac{v'}{w'}. \tag{2.46}$$

A câmera possui uma posição e orientação arbitrárias com respeito ao sistema de coordenadas inercial, logo a posição do ponto com respeito a câmera é dada por $(\tilde{P})_c = T_{0c}^{-1}\tilde{P}$. Utilizando a equação (2.45) podemos escrever a projeção da câmera na sua forma geral como

$$\tilde{p}_{px} = \begin{bmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} T_{0c}^{-1}(\tilde{P})_0 \tag{2.47}$$

$$= KP_0T_{0c}^{-1}(\tilde{P})_0 \tag{2.48}$$

$$= C(\tilde{P})_0 \tag{2.49}$$

A equação (2.47) expressa a posição de um ponto no plano da imagem em coordenadas homogêneas. A matriz C realiza a mudança de escala, transformação e projeção de perspectiva.

2.5.2 Estimação da Pose

O problema de estimar a *pose* consiste em determinar a posição e orientação T_{ct} do alvo com respeito ao sistema de coordenadas da câmera. Considera-se que a geometria do alvo é conhecida, isto é um conjunto de pontos característicos $[X_i \; Y_i \; Z_i]$ com $i \in [1 \; \cdots \; N]$, assim como os parâmetros intrínsecos da câmera. A imagem capturada pela câmera é processada e as coordenadas no plano da imagem $p_{px} = [u_i \; v_i]$ são determinadas utilizando algoritmos de visão computacional. Esse problema é conhecido como *Perspective-n-Point*.

Existem diversas abordagens para solucionar esse problema. Aqui será destacado o caso simples com 3 pontos e comentada a abordagem de (Dementhon & Davis 1995) para N pontos pois esta foi utilizada na implementação desse projeto. Maiores detalhes sobre implementações disponíveis em código aberto podem ser encontradas no apêndice B.

P3P: Estimação da *pose* com 3 pontos

Para entender o problema, considera-se o caso mais simples, com 3 pontos, já que, teoricamente como a *pose* pode ser representada por 6 parâmetros independentes, três pontos seriam capazes de resolver o problema (Marchand et al. 2016). Sejam $P_i = [X_i \ Y_i \ Z_i]^T$ onde $i = 1 \dots 3$ três pontos com coordenadas representadas no referencial da câmera.

Primeiramente, é feita uma estimativa da coordenada de profundidade Z_i de cada ponto utilizando a lei dos cossenos no triângulo dado por CP_iP_j onde C é o ponto onde a câmera está posicionada. Para cada par de correspondências $P_i \leftrightarrow p_i$ e $P_j \leftrightarrow p_j$ podemos escrever (Quan & Lan 1999)

$$d_{ij}^2 = w_i^2 + w_j^2 - 2w_iw_j \cos\theta_{ij} \tag{2.50}$$

onde $d_{ij} = ||P_i - P_j||$, $w_i = ||P_i - C||$ e $w_j = ||P_j - C||$. Cada restrição pode ser escrita como

$$f_{ij}(w_i, w_j) = w_i^2 + w_j^2 - 2w_iw_j \cos\theta_{ij} - d_{ij}^2 = 0 \tag{2.51}$$

resultando no sistema

$$\begin{cases} f_{12}(w_1, w_2) = 0 \\ f_{13}(w_1, w_3) = 0 \\ f_{23}(w_2, w_3) = 0 \end{cases}$$

Este sistema possui 8 soluções, no entanto como não possui termos lineares as soluções ocorrem em 4 pares. É possível manipular as equações de modo a chegar em uma polinomial de oitavo grau em w_1 somente com termos pares, isto é, uma polinomial de quarto grau em $w = w_1^2$.

$$g(x) = a_5w^4 + a_4w^3 + a_3w^2 + a_2w + a_1 = 0 \tag{2.52}$$

Essa equação possui solução fechada e como $w_i > 0$, então $w_1 = \sqrt{q}$. Logo, w_1 e w_2 são determinados unicamente a partir de w_1. Para obter solução única é preciso adicionar um quarto ponto, o que gera um sistema com mais restrições do que incógnitas. Uma possível abordagem é resolver o problema para subconjuntos de três pontos e encontrar a solução comum. No entanto isso não aumenta a precisão do resultado e se houver ruido pode ser difícil encontrar a solução comum.

Conhecidas as distâncias w_i dos pontos do mundo à câmera, essas distâncias são convertidas em coordenadas tridimensionais centradas na câmera através de $P'_i = w_iK^{-1}p_i$, onde K é a matriz de calibração da câmera. O último passo é determinar a orientação, uma transformação de similaridade pode ser obtida através

de dois pares de pontos $P'_i \leftrightarrow P_i$. A solução pode ser obtida através de mínimos utilizando quatérnions. A partir da estimativa da rotação a obtenção da translação e da escala seguem trivialmente.

Esse método auxilia na compreesão do problema, porém não é robusto e é pouco preciso, além de que os dados redundantes (quarto ponto) não aumentam a precisão do resultado. Portanto, outros algoritmos foram propostos.

PNP: Estimação da *pose* com N pontos

Em (Dementhon & Davis 1995) é proposto combinar dois algoritmos. O primeiro *POS (Pose from Orthography and Scaling)* aproxima a projeção de perspectiva com uma projeção ortográfica (e de escala) e encontra a matriz de rotação e o vetor de translação do objeto resolvendo um sistema linear. O segundo algoritmo *POSIT (POS with ITerations)*, usa a *pose* aproximada pelo *POS* em um *loop* para computar melhores projeções ortográficas (e de escala) dos pontos característicos. Então o *POS* é aplicado a essas projeções, invés de às projeções da imagem original. O *POSIT* converge para medidas precisas em poucas iterações e pode utilizar mais pontos para insensibilidade a erros de medição e ruido.

Uma desvantagem do *POSIT* é que ele não é diretamente aplicável a pontos coplanares (Marchand et al. 2016). No entanto, em (Oberkampf et al. 1996) uma extensão ao *POSIT* foi proposta, resolvendo o problema para 4 ou mais pontos coplanares.

2.5.3 Servovisão Baseada em Posição

Em um sistema de servovisão baseado em posição a posição e orientação do alvo com respeito a câmera T_{ct} é estimada. O problema de estimação da posição e orientação foi discutido acima e as implementações disponíveis em código aberto estão listadas no Apêndice B. Como desejamos que alguma ferramenta no efetuador seja capaz de atingir o alvo, é preciso saber a posição e orientação do alvo em relação ao efetuador, dada por

$$T_{et} = T_{ec}T_{ct}. \tag{2.53}$$

Especifica-se uma posição desejada relativa ao sistema de coordenadas do alvo $T_{e^*t} = T_{ec}T_{c^*t}$ e deseja-se determinar o movimento necessário para mover a câmera para a posição desejada, que chamamos de T_Δ.

$$T_{et} = T_\Delta T_{e^*t} \tag{2.54}$$

$$T_\Delta = T_{et}T_{e^*t}^{-1} \tag{2.55}$$

Com isso é possível aplicar uma estratégia de controle cinemático de posição no

referencial do efetuador de modo a atingir a posição e orientação desejada. A lei de controle

$$u = (J_a)_e^{-1} K_v x_\Delta \tag{2.56}$$

é capaz de fazer $x_\Delta(t) \to 0$ quando $t \to \infty$ se $\dot{x}_t = 0$. Na Equação (2.56), x_Δ pode ser obtido a partir de T_Δ. Alternativamente, é possível especificar x_{e^*t} no espaço operacional e x_Δ será dado por

$$x_\Delta = x_{et} - x_{e^*t} \tag{2.57}$$

onde x_{et} é obtido a partir de T_{et} e x_{e^*t} a partir de T_{e^*t}.

2.6 Controle de Força

2.6.1 Problema de Regulação

A estratégia de controle de força baseada em uma ação proporcional e integral tem sido o algoritmo mais utilizado devido a sua robustez com respeito ao atraso de tempo de medição e capacidade de remoção de perturbações de força (Wilfinger et al. 1994). No problema de regulação, o uso de um controlador PI permite evitar instabilidade em uma possível perda de contato.

Considerando o problema de regular a força de contato medida f para uma força desejada constante f_d ao longo da superfície.

$$f \to f_d, \qquad e_f = f - f_d \to 0 \tag{2.58}$$

Como em geral as medidas podem ser contaminadas por ruído utiliza-se um filtro de primeira ordem.

$$\tau \dot{e}'_f = -e'_f + e_f \tag{2.59}$$

onde $e'_f \in \mathbb{R}^3$ é o erro de força filtrado e τ é a constante de tempo do filtro.

Derivando (??), (2.58) e (2.59) com respeito ao tempo, a equação do erro de força é dada por

$$\tau \ddot{e}'_f + \dot{e}'_f = f_d + K_s \bar{u}_f \tag{2.60}$$

onde $\bar{u}_f \in \mathbb{R}^3$ é a lei de controle de posição $\bar{u}_p$ em

$$\bar{u} = \begin{bmatrix} \bar{u}_p \\ \bar{u}_o \end{bmatrix} \tag{2.61}$$

Considerando $\bar{u}_p = \bar{u}_f$, utiliza-se uma lei de controle proporcional

$$\bar{u}_f = -K_s^{-1}(K_f e'_f + K_i \int_0^t e'_f(\tau)d\tau), \tag{2.62}$$

onde $K_f = k_f I$ e $K_i = k_i I$. Portanto a dinâmica do erro é governada por

$$\tau \dddot{e}'_f + \ddot{e}'_f + k_f \dot{e}'_f + k_i e_f = 0 \tag{2.63}$$

Assim, deve-se escolher k_f e k_i como constantes positivas satisfazendo as condições de estabilidade estabelecidas pelo critério de establididade de *Routh–Hurwitz*, ou seja, $k_f > k_i \tau$. Sob essas condições o sistema em malha fechada é exponencialmente estável.

2.7 Controle Híbrido de Força e Posição

É chamado de controle híbrido a estratégia que envolve o uso de restrições artificiais para especificar alvos do sistema e controlar somente as variáveis que não estão sujeitas às restrições naturais (Xaud 2016). Dessa forma as variáveis que não estão restritas pelo ambiente não são afetadas pela lei de controle.

Considerando que força e posição estão em sub-espaços de trabalho complementares é possível dividir o controle de força em duas malhas que não se interferem. Essa separação é feita pela matriz de seleção S. O controlador híbrido utiliza a matriz S para dividir as malhas de força e a posição que atuam sobre o erro computado no sistema de coordenadas E_s da superfície de restrição. A lei de controle híbrido é definida como:

$$u_h = u_{hf} + u_{hp} \tag{2.64}$$

onde u_{hf} e u_{hp} são os sinais de controle atuando sobre os subespaços de força e posição respectivamente.

Suponha que deseja-se aplicar controle de posição em x_s, y_s e controle de força em z_s, normal a superfície de contato. A matriz de seleção de força é definida como

$$S_f = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.65}$$

e cumpre o papel de cancelar os esforços de controle nos graus de liberdade complementares. A matriz de seleção de posição é complementar e pode ser definida

como

$$S_p = (I - S_f) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.66}$$

Se o manipulador está sendo controlado em seu referencial base, utilizando o Jacobiano geométrico, a equação (2.64) deve ser representada como

$$(u_h)_b = (u_{hf})_b + (u_{hp})_b \tag{2.67}$$

Geralmente o erro de posição é definido como $(e_p)_b = (p_d)_b - (p_{be})_b$, enquanto que o erro de força é definido com respeito ao sistema de coordenadas do efetuador $(e_f)_e = f_d - (f_c)_e$. O controle de posição utiliza uma estratégia de controle proporcional com feedforward definida como:

$$(\bar{u}_p)_b = (\dot{p})_b + K_p(e_p)_b \tag{2.68}$$

Para o sinal de controle de força é utilizada um controle proporcional integral cuja lei é dada por

$$(\bar{u}_f)_e = K_{pf}(e_f)_e + K_{if} \int_0^\tau (e_f(\tau))_e d\tau \tag{2.69}$$

onde a parcela feedforward não é utilizada, já que considera-se apenas o problema de *set-point*, para o qual $\dot{f}_d$. Como a matriz de seleção desacopla os subespaços de controle somente no referencial da superfície $\bar{E}_s$, os sinais de controle devem ser representados primeiramente nesse sistema de coordenadas, antes de serem operados po S e, então, representados de volta no sistema de coordenadas da base.

$$(u_{hf})_b = R_{bs} S_f R_{se} (\bar{u}_f)_e \tag{2.70}$$

$$(u_{hp})_b = R_{bs} S_p R_{sb} (\bar{u}_p)_b \tag{2.71}$$

onde R_{bs} é a matriz de rotação do sistema de coordenadas $\bar{E}_b$ da base para o da superfície $\bar{E}_s$, R_{se} é a matriz de rotação do sistema de coordenadas $\bar{E}_s$ da superfície para o sistema de coordenadas do efetuador $\bar{E}_e$ e $R_{sb} = R_{bs}^T$.

O controle híbrido pode ser representado pelo diagrama de blocos da figura 2.7.

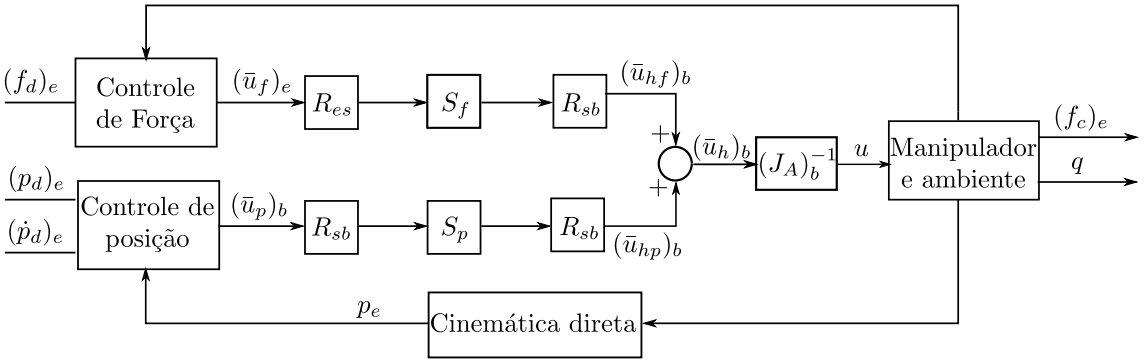


Figura 2.7: Diagrama de blocos: Controle Híbrido Força-Posição

Capítulo 3

TETIS - Modelagem e Controle

3.1 Modelo

O manipulador TETIS consiste basicamente de um braço planar de três elos com uma rotação adicional ao redor do eixo do plano. Também pode ser visto como um braço antropomórfico com uma junta de revolução adicional ao final da cadeia, com o eixo paralelo às duas anteriores. O projeto mecânico faz com que a extremidade do efetuador final não esteja exatamente alinhada com as juntas, o que é levado em conta no último elo. O esquema na figura 3.1 ilustra o modelo de elos e juntas.

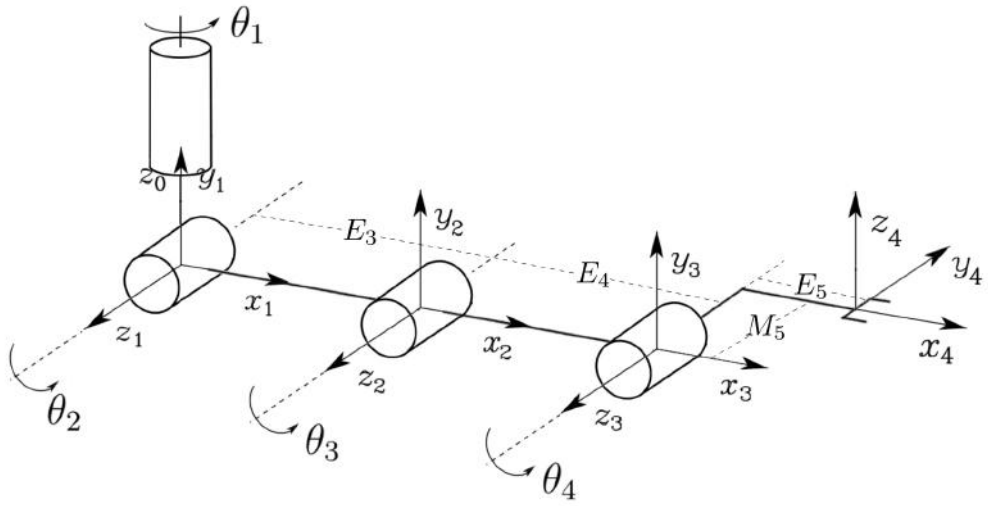


Figura 3.1: Modelagem cinemática do manipulador TETIS e posicionamento dos sistemas de coordenadas

3.2 Cinemática Direta

O primeiro passo ao modelar um manipulador robótico é encontrar a cinemática direta. Será utilizada a convenção de Denavit-Hartenberg para posicionar os sistemas

de coordenadas e obter os parâmetros. Seguindo os passos descritos na Seção 2.2.1 os sistemas de coordenadas foram posicionados da seguinte forma:

- Eixo $\vec{z}_0$ escolhido ao longo da Junta 1. Origem O_0 escolhida arbitrariamente de modo a coincidir com O_1 por simplicidade.
- Eixo $\vec{z}_1$ escolhido ao longo da Junta 2. Origem O_1 na intersecção entre $\vec{z}_0$ e $\vec{z}_1$. Eixo $\vec{x}_1$ na direção normal ao plano definido por $\vec{z}_0$ e $\vec{z}_1$ pois se interceptam. O sentido foi arbitrariamente escolhido na direção de avanço da cadeia por simplicidade.
- Eixo $\vec{z}_2$ escolhido ao longo da Junta 3. Origem O_2 na junta 3 pois $\vec{z}_2$ e $\vec{z}_1$ são paralelas. Eixo $\vec{x}_2$ escolhido arbitrariamente ao longo de $\vec{x}_1$ pois a normal comum entre $\vec{z}_2$ e $\vec{z}_1$ não é unicamente definida.
- Eixo $\vec{z}_3$ escolhido ao longo da Junta 4. Origem O_3 na junta 3 pois $\vec{z}_3$ e $\vec{z}_2$ são paralelas. Eixo $\vec{x}_3$ escolhido arbitrariamente ao longo de $\vec{x}_2$ pois a normal comum entre $\vec{z}_3$ e $\vec{z}_2$ não é unicamente definida.
- Como não existe junta 5, eixo $\vec{z}_4$ definido arbitrariamente. Origem O_4 na extremidade do efetuador final. Eixo $\vec{x}_4$ normal ao eixo $\vec{z}_3$.

A partir dos eixos posicionados conforme a figura 3.1 é possível obter os parâmetros na tabela 4.1.

Tabela 3.1: Parâmetros Denavit–Hartenberg para manipulador Tetis

Elo	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	θ_1
2	E_3	0	0	θ_2
3	E_4	0	0	θ_3
4	E_5	$-\pi/2$	$-M_5$	θ_4

- $a_1 = 0$ e $d_1 = 0$ pois O_0 e O_1 coincidem. $\alpha_1 = \pi/2$ ângulo entre $\vec{z}_0$ e $\vec{z}_1$ ao redor de $\vec{x}_1$.
- $a_2 = E_3$ é a distância entre $\vec{z}_1$ e $\vec{z}_2$ ao longo de $\vec{x}_2$ que corresponde ao comprimento do elo 2. $\vec{z}_1$ e $\vec{z}_2$ são sempre paralelos logo $\alpha_2 = 0$. A distância d_2 entre $\vec{x}_1$ e $\vec{x}_2$ ao longo de $\vec{z}_1$ é zero.
- $a_3 = E_4$ é a distância entre $\vec{z}_2$ e $\vec{z}_3$ ao longo de $\vec{x}_3$ que corresponde ao comprimento do elo 3. $\vec{z}_2$ e $\vec{z}_3$ são sempre paralelos logo $\alpha_3 = 0$. A distância d_3 entre $\vec{x}_2$ e $\vec{x}_3$ ao longo de z_2 é zero.

- $a_4 = E_5$ é a distância entre $\vec{z}_3$ e $\vec{z}_4$ ao longo de $\vec{x}_4$. $\alpha_4 = -\pi/2$ é o ângulo entre $\vec{z}_3$ e $\vec{z}_4$ ao redor de $\vec{x}_4$, sendo portanto negativo. $d_4 = -M_5$ é a distância entre $\vec{x}_3$ e $\vec{x}_4$ ao longo de $\vec{z}_3$.

Considerando a posição inicial da figura 3.1 todos os ângulos θ_i são dados diretamente pelas variáveis de junta, sem *offsets*.

Obtemos a partir da equação (2.26) as transformações homogêneas entre sistemas de coordenadas consecutivos:

$$T_{01} = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{12} = \begin{bmatrix} c_2 & -s_2 & 0 & E_3 c_2 \\ s_2 & c_2 & 0 & E_3 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{23} = \begin{bmatrix} c_3 & -s_3 & 0 & E_4 c_3 \\ s_3 & c_3 & 0 & E_4 s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{34} = \begin{bmatrix} c_4 & 0 & -s_4 & E_5 c_4 \\ s_4 & 0 & c_4 & E_5 s_4 \\ 0 & -1 & 0 & -M_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{04} = T_{01}T_{12}T_{23}T_{34} = \begin{bmatrix} c_1 c_{234} & -s_1 & -c_1 s_{234} & -M_5 s_1 + E_4 c_{23} c_1 + E_3 c_1 c_2 + E_5 c_{234} c_1 \\ s_1 c_{234} & -c_1 & -s_1 s_{234} & M_5 c_1 + E_4 c_{23} s_1 + E_3 c_2 s_1 + E_5 c_{234} s_1 \\ s_{234} & 0 & c_{234} & E_4 s_{23} + E_3 s_2 + E_5 s_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Definindo $T_{b0} = I$ e $T_{4e} = I$, pela equação (2.20) temos

$$T_{be}(q) = T_{04}(q) \tag{3.2}$$

3.3 Espaço das Juntas e Operacional

Antes de tratar de estratégias de controle é necessário definir o espaço operacional e o espaço das juntas, sob os quais serão aplicadas as leis de controle. Como trata-se de um manipulador 4-DOF de temos que o vetor do espaço operacional tem dimensão (4×1) dado por

$$x_e = \begin{bmatrix} p_{be} \\ \phi_{be} \end{bmatrix} \tag{3.3}$$

onde o vetor p_{be} descreve a posição cartesiana representada no referencial da base e ϕ_{be} é o grau de liberdade de orientação *pitch*, dado por

$$\phi_{be} = -(\theta_2 + \theta_3 + \theta_4) \tag{3.4}$$

O espaço das juntas é definido por

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} \tag{3.5}$$

pois todas as juntas são de revolução.

3.4 Cinemática Diferencial

3.4.1 Jacobiano Analítico

A partir da cinemática direta em (3.1) e da equação 2.29 podemos obter o Jacobiano de posição para o manipulador diferenciando a equação em relação as variáveis de junta.

$$p_{be} = \begin{bmatrix} p_{be_x} \\ p_{be_y} \\ p_{be_z} \end{bmatrix} = \begin{bmatrix} -M_5 s_1 + E_4 c_{23} c_1 + E_3 c_1 c_2 + E_5 c_{234} c_1 \\ M_5 c_1 + E_4 c_{23} s_1 + E_3 c_2 s_1 + E_5 c_{234} s_1 \\ E_4 s_{23} + E_3 s_2 + E_5 s_{234} \end{bmatrix} \tag{3.6}$$

$$(J_{ap})_b = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} & \frac{\partial x}{\partial q_4} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} & \frac{\partial y}{\partial q_4} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} & \frac{\partial z}{\partial q_4} \end{bmatrix} \tag{3.7}$$

onde

$$\begin{aligned}
\frac{\partial x}{\partial q_1} &= -M_5 c_1 - E_4 c_{23} s_1 - E_3 c_2 s_1 - E_5 c_{234} s_1 \\
\frac{\partial x}{\partial q_2} &= -c_1(E_4 s_{23} + E_3 s_2 + E_5 s_{234}) \\
\frac{\partial x}{\partial q_3} &= -c_1(E_4 s_{23} + E_5 s_{234}) \\
\frac{\partial x}{\partial q_4} &= -E_5 s_{234} c_1 \\
\frac{\partial y}{\partial q_1} &= -M_5 s_1 + E_4 c_{23} c_1 + E_3 c_1 c_2 + E_5 c_{234} c_1 \\
\frac{\partial y}{\partial q_2} &= -s_1(E_4 s_{23} + E_3 s_2 + E_5 s_{234})
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial y}{\partial q_3} &= -s_1(E_4 s_{23} + E_5 s_{234}) \\
\frac{\partial y}{\partial q_4} &= -E_5 s_{234} s_1 \\
\frac{\partial z}{\partial q_1} &= 0 \\
\frac{\partial z}{\partial q_2} &= E_4 c_{23} + E_3 c_2 + E_5 c_{234} \\
\frac{\partial z}{\partial q_3} &= E_4 c_{23} + E_5 c_{234} \\
\frac{\partial z}{\partial q_4} &= E_5 c_{234}
\end{aligned}
$$

A partir de 2.30 e de 3.4 podemos calcular o Jacobiano de orientação

$$J_{a\phi}(q) = \frac{\partial \phi_e}{\partial q} = \begin{bmatrix} 0 & -1 & -1 & -1 \end{bmatrix} \tag{3.8}$$

Em alguns modos como controle por servovisão e no controle manual com *joystick* é interessante fazer o controle no referencial do efetuador, portanto precisamos representar o Jacobiano de posição no referencial do efetuador como $(J_{ap})_e = R_{be}^T (J_{ap})_b$.

$$(J_{ap})_e = \begin{bmatrix} -M_5 c_{234} & E_3 s_{34} + E_4 s_4 & E_4 s_4 & 0 \\ E_4 c_{23} + E_3 c_2 + E_5 c_{234} & 0 & 0 & 0 \\ M_5 s_{23} & E_5 + E_3 c_{34} + E_4 c_4 & E_5 + E_4 c_4 & E_5 \end{bmatrix} \tag{3.9}$$

Note que para a escolha de sistemas de coordenadas do TETIS tem-se que

$$(J_{a\phi})_e = (J_{a\phi})_b. \tag{3.10}$$

3.5 Malha Aberta no Espaço Operacional

Este é um modo em malha aberta, onde o sinal de entrada no controlador é de velocidade no referencial da base ou do efetuador, ou seja a velocidade linear $\dot{p}_d$. Utiliza-se a equação (2.29) para calcular a velocidade de cada junta.

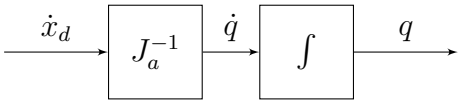


Figura 3.2: Diagrama de Blocos: Modo de velocidade do Espaço Operacional

3.5.1 Sistema de coordenadas da Base

Quando o controle é feito no referencial da base $(\dot{x}_d)_b$ é velocidade desejada expressa no referencial da base e utiliza-se $(J_a)_b$.

$$u = \dot{q}_d = (J_a)_b^{-1}(\dot{x}_d)_b \tag{3.11}$$

3.5.2 Sistema de coordenadas do Efetuador

Quando o controle é feito no referencial do efetuador $(\dot{x}_d)_e$ é velocidade desejada expressa no referencial do efetuador e utiliza-se $(J_a)_e$.

$$u = \dot{q}_d = (J_a)_e^{-1}(\dot{x}_d)_e \tag{3.12}$$

onde

$$\dot{x}_d = \begin{bmatrix} \dot{p}_d \\ \dot{\phi}_d \end{bmatrix} \tag{3.13}$$

3.6 Controle de Posição no Espaço das Juntas

O controle de posição no espaço das juntas consiste em uma realimentação com controle proporcional individual para cada junta. Considerando que as juntas sejam modeladas como integradores, o diagrama 3.3 mostra a malha de controle.

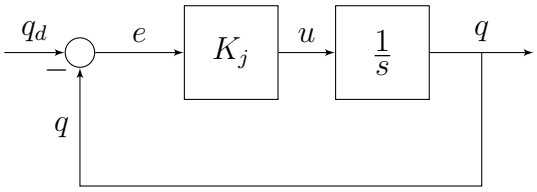


Figura 3.3: Diagrama de Blocos: Modo de Posição no Espaço das Juntas

A lei de controle é dada por

$$u = K_j(q_d - q) \tag{3.14}$$

onde $K_j = k_j I > 0$.

3.7 Controle de Posição no Espaço Operacional

Para este modo considera-se o problema de *set-point* para o controle de posição no espaço operacional conforme definido em 2.27. Para referências constantes, ou seja,

quando $\dot{x}_d = 0$ a lei de controle dada por

$$u = (J_a)_b^{-1} K_p (x_d - (x_{be})_b) \tag{3.15}$$

é capaz de levar o $e \to 0$ quando $t \to \infty$.

3.8 Controle Proporcional com Feedforward

Para o rastreamento de trajetória considera-se o problema de seguir um referência no referencial da base $x_d(t)$ que é função do tempo, conhecidos $x_d(t)$ e $\dot{x}_d(t)$. Para levar o erro assintoticamente a zero, utiliza-se uma lei de controle proporcional com *feedforward* utilizando a inversa do Jacobiano Analítico. Conforme mostrado na Seção 2.4, a lei de controle dada por

$$u = (J_a)_b^{-1} [\dot{x}_d + K_t (x_d - (x_{be})_b)] \tag{3.16}$$

é capaz de levar o erro assintoticamente a zero pois a dinâmica fica

$$\dot{e} + Ke = 0 \tag{3.17}$$

O algoritmo de controle pode ser resumido por:

$$e = x_d - (x_{be})_b \tag{3.18}$$

$$\bar{u} = K_t e + (\dot{x}_d)_b \tag{3.19}$$

$$u = (J_a)_b^{-1} \bar{u} \tag{3.20}$$

3.9 Controle por Servovisão

O controle por servovisão no manipulador TETIS é feito através de uma configuração *eye-in-hand*, onde a câmera é montada no efetuador final do manipulador, utilizando a abordagem de Servovisão Baseada em Posição (PBVS). Na Seção 4.3 é descrita a câmera utilizada. Trata-se de uma câmera estereoscópica, no entanto será utilizada apenas um dos sensores. Assume-se que é possível aproximar a câmera pelo modelo *pin-hole* descrito na Seção 2.5.1. Consideram-se desprezíveis as distorções geométricas causadas pelas lentes.

O objetivo é deste modo de controle é rastrear um alvo, que é representado por um QR code como o da Figura 3.5. Deseja-se que o efetuador rastreie sempre a orientação normal ao plano do alvo.

A matriz de calibração K da equação (2.47) obtida através do processo descrito em (Spindler 2014) é:

$$K = \begin{bmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 877.62 & 0 & 306.53 \\ 0 & 880.32 & 210.12 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

Observando que o referencial da câmera e do efetuador são dados pela Figura 3.4, podemos chegar a seguinte transformação homogênea

$$T_{ec} = \begin{bmatrix} 0 & 0 & 1 & -30 \\ 0 & -1 & 0 & 101 \\ 1 & 0 & 0 & -43 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

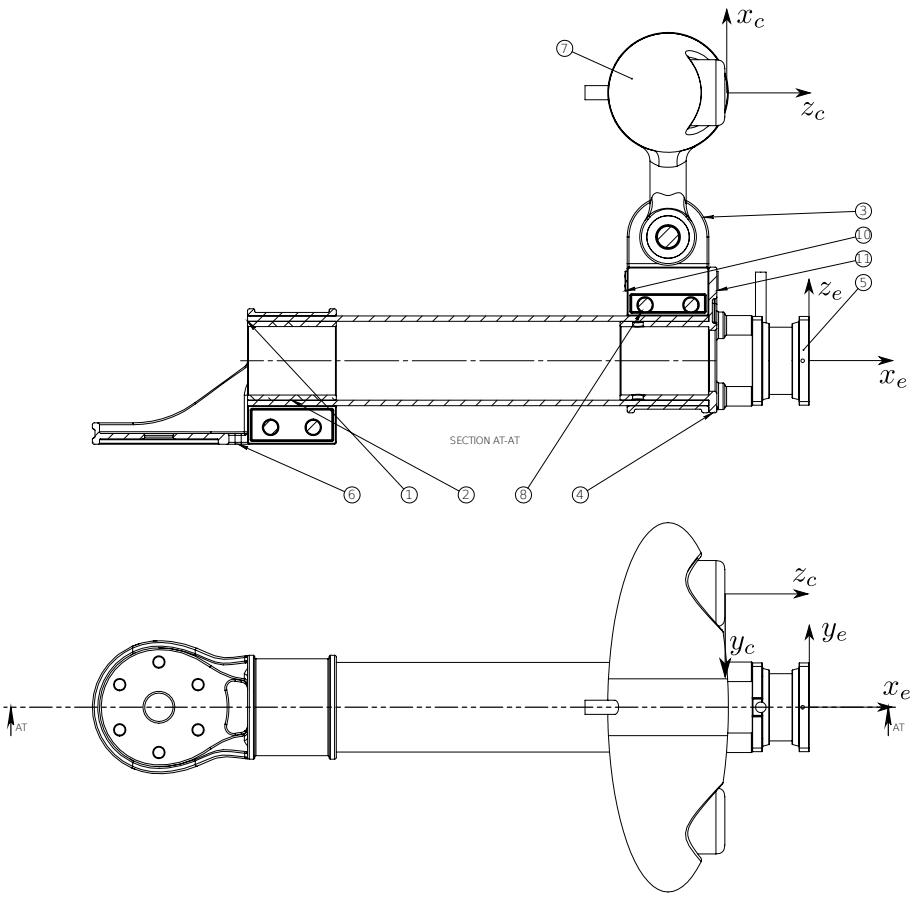


Figura 3.4: Sistemas de coordenadas da câmera e do efetuador

Para a abordagem PBVS, foi utilizada a lei de controle proporcional descrita na equação (2.56)

$$u = (J_a)_e^{-1} K_v x_\Delta$$

onde $x_\Delta = (x_{et})_e - (x_{e^*t})_e$. O vetor do espaço operacional x_{e^*t} representa uma posição e orientação relativa ao alvo, enquanto que $(x_{et})_e$ é obtida a partir de T_{et}.

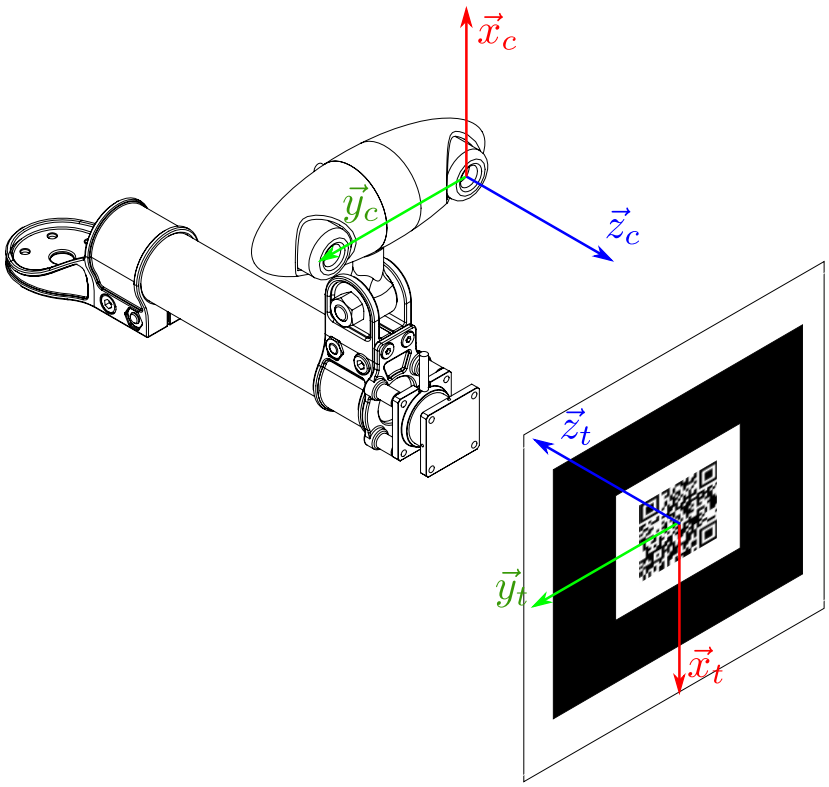


Figura 3.5: Sistemas de coordenadas da câmera e do alvo

$$T_{et} = T_{ec}T_{ct} \tag{3.23}$$

onde T_{ec} é dado por (3.22) e T_{ct} obtemos através de algoritmos de estimação de posição e orientação a partir de visão computacional. Ficamos então com

$$(x_{et})_e = \begin{bmatrix} (p_{et})_e \\ (\phi_{et})_e \end{bmatrix} \tag{3.24}$$

onde $(p_{et})_e$ é o vetor de translação que pode ser obtido diretamente de T_{et} e $(\phi_{et})_e$ é a orientação, que pode ser obtido de duas formas.

A primeira alternativa é obter a rotação em torno do eixo x ($pitch$) em relação ao referencial do efetuador, extraído de T_{et}. No entanto essa opção não permite que o alvo seja rotacionado em torno de z_c, pois a rotação em torno de x_c não mais representará a inclinação do plano do alvo em relação a posição inicial, que faz o efetuador rastrear a direção normal ao alvo (vide Figura 3.6).

Portanto, optou-se pela segunda alternativa. Projeta-se o eixo do alvo $(z_t)_c$, representado do referencial da câmera, no plano $(z_{t_0})_c \times (x_{t_0})_c$, onde $(z_{t0})_c$ indica o eixo de coordenadas do alvo z_t em sua posição inicial conforme a Figura 3.5. Em seguida encontra-se o ângulo entre os vetores $(z_{t_0})_c$ e $(z_{t_0})_c$.

Dada a matriz de rotação R_{ct_0}, do sistema de coordenadas da câmera ao sistema

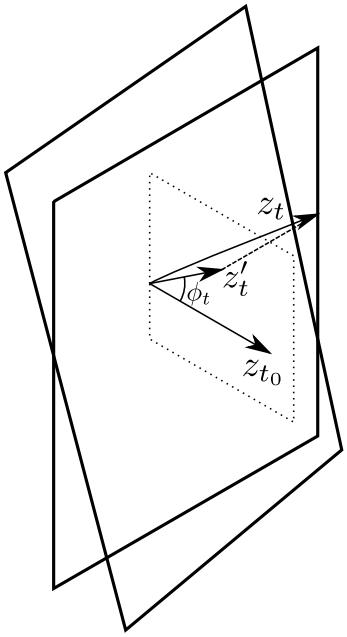


Figura 3.6: Caso em que a rotação em torno de x_c não representa a inclinação do plano do alvo.

de coordenadas do alvo em sua posição inicial, mostrada na Figura 3.4

$$R_{ct_0} = \begin{bmatrix} (x_{t_0})_c & (y_{t_0})_c & (z_{t_0})_c \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.25}$$

é conhecida a normal ao plano do alvo é $n = x_{t_0} \times z_{t_0} = y_{t_0}$. A matriz de projeção linear (Strang 2005) de um vetor em um plano é dada por

$$P = I - nn^T, \tag{3.26}$$

logo para $n = y_{t_0}$ temos

$$P = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.27}$$

O vetor projetado no plano é dado por

$$z'_t = Pz_t, \tag{3.28}$$

assim basta apenas encontrar o ângulo entre os vetores z'_t e z_{t_0}, que pode ser calculado pela fórmula do Cosseno (Strang 2005)

$$\phi_{et} = \cos^{-1}\left(\frac{z'^T_t z_{t_0}}{||z'_t||\ ||z_{t_0}||}\right) \tag{3.29}$$

Após obter os valores de $(p_{et})_e$ e ϕ_{et}, sabemos $(x_{et})_e$ e controle se resume a:

$$x_\Delta = (x_{et})_e - (x_{e^*t})_e \tag{3.30}$$

$$\bar{u} = K_v x_\Delta \tag{3.31}$$

$$u = (J_a)_e^{-1}\bar{u} \tag{3.32}$$

3.10 Controle de Força

Utilizando o sensor de força descrito em 4.4, montado no efetuador final do manipulador, deseja-se fazer o controle da força aplicada sobre uma superfície.

A matriz de rotação do referencial do sensor para o referencial do efetuador, como definido em 3.1 é dada por

$$R_{es} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.33}$$

3.10.1 Float

Esse mode de controle de força consiste em configurar uma referência de controle de força $f_d = 0$, de modo que o efetuador do manipulador fique "flutuando", se movendo de forma sensível ao toque.

Primeiramente a força representada no referencial do sensor $(f)_s \in \mathbb{R}^3$ é representada na base por $(f)_e = R_{es}(f)_s$. Com $f_d = 0$, o erro fica

$$(e_f)_e = -(f)_e \tag{3.34}$$

Utilizando um controle proporcional:

$$\bar{u}_f = K_f(e_f)_e \tag{3.35}$$

$$\bar{u} = \begin{bmatrix} \bar{u}_f & 0 \end{bmatrix}^T \tag{3.36}$$

$$u = (J_a)_b^{-1}\bar{u} \tag{3.37}$$

3.10.2 Approach

Considera-se o problema de controle de força na direção de *approach* para o manipulador robótico 4-DOF em questão. O objetivo de controle é resolver o problema de *set-point*, ou seja, rastrear uma referência de força constante.

É possível modelar o ambiente (força de contato), ou seja, a placa de poliestireno,

como uma mola linear, através da *Lei de Hooke*:

$$f = -k_s(x - x_s) \tag{3.38}$$

onde x é a posição do ponto de contato com a superfície e x_s um ponto na superfície.

Considerando que o controle de força seja ativado somente após a etapa de contato com o ambiente onde será aplicada a força e que o controle de força é feito apenas na direção de approach (segundo o sistema de coordenadas $\bar{E}_e$, na direção x), pode-se utilizar a malha de controle mostrada na figura 3.7. É utilizada uma lei de controle com ação proporcional e integral conforme a equação (2.62).

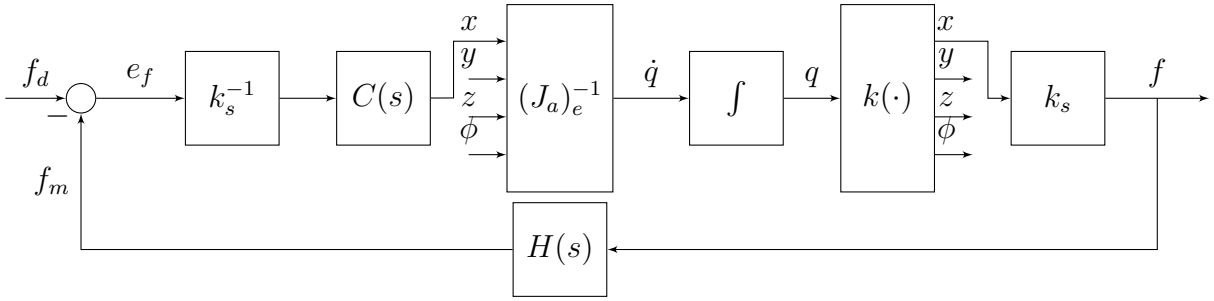


Figura 3.7: Diagrama de Blocos: Malha de Controle de Força.

O diagrama da Figura 3.7 pode ser simplificado se abstrairmos as outras dimensões que não a de approach, resultando em 3.8.

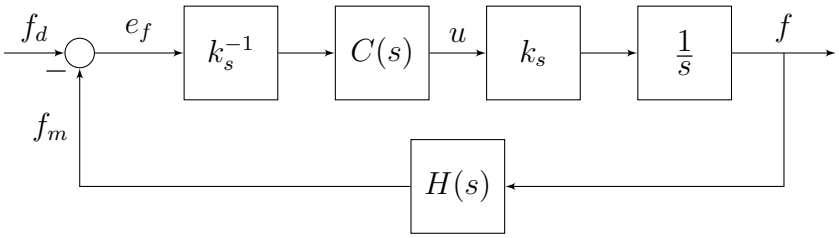


Figura 3.8: Diagrama de Blocos: Malha de Controle de Força Simplificada.

Considerando $H(s) = 1$

$$G(s) = \frac{k_p s + k_i}{s^2 + k_p s + k_i} \tag{3.39}$$

Como o sinal vindo do sensor é bastante ruidoso, utiliza-se um filtro de primeira ordem com $f_c = 1$.

$$H(s) = \frac{1}{\tau s + 1} \tag{3.40}$$

onde $\tau = 1/(2\pi f_c) \approx 0.16$.

Ficamos com a função de transferência a seguir a partir da qual é possível sinto-

nizar os parâmetros do controlador.

$$G(s) = \frac{k_p \tau s^2 + (k_p + \tau k_i)s + k_i}{\tau s^3 + s^2 + k_p s + k_i} \tag{3.41}$$

Considerando que obtemos do sensor de força um vetor $(f)_s \in \mathcal{R}^3$, representado no referencial do sensor. O controle de força pode ser implementado a partir das seguintes equações:

$$(f)_e = R_{es}(f)_s \tag{3.42}$$

$$(e_f)_e = f_d - (f)_e \tag{3.43}$$

$$\tag{3.44}$$

A estratégia de controle PI é dada por

$$\bar{u}_f = -K_s^{-1}(K_f(e_f)_e + K_i \int_0^t (e_f)_e(\tau)d\tau) \tag{3.45}$$

Como $u \in \mathcal{R}^4$ e desejamos controlar somente a direção de *approach*:

$$\bar{u} = \begin{bmatrix} S_f \bar{u}_f \\ 0 \end{bmatrix} \tag{3.46}$$

onde

$$S_f = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{3.47}$$

portanto:

$$u = (J_a)_e^{-1}\bar{u} \tag{3.48}$$

Capítulo 4

Descrição do Hardware

Nesse capítulo serão descritos elementos do hardware do robô DORIS associados ao funcionamento do manipulador TETIS. A descrição completa dos dispositivos do DORIS, pode ser obtida em (Freitas et al. 2015, Galassi et al. 2014, Nunes et al. 2013, Xaud 2016).

4.1 Eletrônica Embarcada

A eletrônica embarcada é composta de um computador central e subsistemas de comunicação, atuação, aquisição de dados e o *Vehicle Suport System (VSS)* (Freitas et al. 2015).

No projeto do DORIS, o computador consite em um módulo PCIe/104 com Intel®Core i7™e disco de estado sólido (SSD). Atualmente ele está operando com um módulo PCIe/104 Intel®Atom D525.

O sistema de comunicação trata do tráfego de dados entre o robô e a base remota. É composto de:

- Rede Local *Gigabit Ethernet*. È responsável pela comunicação de alta largura de banda entre o DORIS e seus diversos periféricos. A essa rede estão conectados o computador embarcado, câmeras *Ethernet*, access points Wi-Fi, roteadores e o *Vehicle Support System* (VSS).

- *Controller Area Network (CAN)*. É responsável pelo controle em tempo real do sistema de atuação através de uma rede robusta. A rede CAN integra os drivers do sistema de atuação e os computadores embarcados em uma topologia de rede em barramento.

- Tecnologias sem fio: O DORIS pode ser remotamento operado por meio de rede *Wi-Fi IEEE 802.11n* ou através de um rádio 2.4/5.0 GHz caso ocorra alguma falha na conexão.

O sistema de aquisição de dados coleta imagem, video e áudio do ambiente. É composto de uma camera fixa, uma câmera térmica, uma câmera *fish-eye*, duas câmeras estereoscópicas e uma unidade de medição inercial (IMU).

4.2 Atuadores e Drivers

O sistema de atuação é responsável pela atuação dos motores do DORIS. É composto pelo subsistema de tração e pelo subsistema do manipulador. As juntas de revolução do TETIS necessitam de atuadores com as seguintes características:

- Alto taxa de redução, para permitir controle a nível de velocidade nas juntas.
- Folga desprezível, para evitar a adição de não linearidades ao sistema.
- Eixo oco para permitir o roteamento de cabos através dele.
- Alta acurácia e repetibilidade

A tecnologia *Harmonic gear*, registrada pela *Harmonic Drive*, consiste em engrenagens do tipo *strain wave*. Esse mecanismo melhora certas características em comparação com sistemas de transmissão tradicionais como engrenagens helicoidais, ou planetárias. As vantagens incluem a inexistência de folgas, ser mais compacto e leve, altas taxas de redução, excelente resolução e repetibilidade e alto torque.

As juntas do manipulador TETIS utilizam atuadores *FHA Mini Servo* da *Harmonic Drive AG*, mostrados na Figura 4.1a. Segundo o estudo em (Xaud 2016), esse atuador cumpre os requisitos acima. Ele conta também com *encoder*, montado no eixo do motor, antes da redução, e sensores *hall* para os três enrolamentos. Com base nos requisitos de torque, para as duas primeiras juntas é utilizado o modelo FHA-11C-100-D200-EKMI, enquanto que para as duas últimas o modelo FHA-8C-100-D200-EKMI.

Para atinjir um controle satisfatório a nível de junta, são utilizados drivers da Maxon Motor, compatíveis com o barramento CAN. Cada junta é controlada por um driver modelo Maxon Motor EPOS2 70/10 P/N 375711, mostrado na figura 4.1b.

4.3 Câmera Minoru

O robô DORIS conta com diversos tipos de câmeras, como câmera de alta definição, térmica, *fisheye* e estereoscópica.

A Figura 4.2 mostra a câmera utilizada para o controle por servovisão e o efetuador com a câmera montada. A câmera em questão é a *Minoru 3D Webcam*, uma

(a) Harmonic Drive FHA Mini Servo

(b) Maxon Motor EPOS2 70/10

Figura 4.1: Atuadores e Drivers

câmera esterescópica que possui dois sensores VGA CMOS. É um dispositivo leve que pode ser acoplado ao efetuador final.

(a) Efetuador com câmera

(b) Minoru Camera

Figura 4.2: Efetuador e câmera Minoru

4.4 Sensor de Força

No efetuador do manipulador TETIS há um sensor de força da *OptoForce Kft.*, modelo OMD-20-FE-200N. Os sensores Optoforce 3D medem a magnitude e direção de forças F_x, F_y e F_z utilizando princípios ópticos (*OMD-20-FE-200N DATASHEET V2.1* 2016). O sensor utiliza uma interface USB.

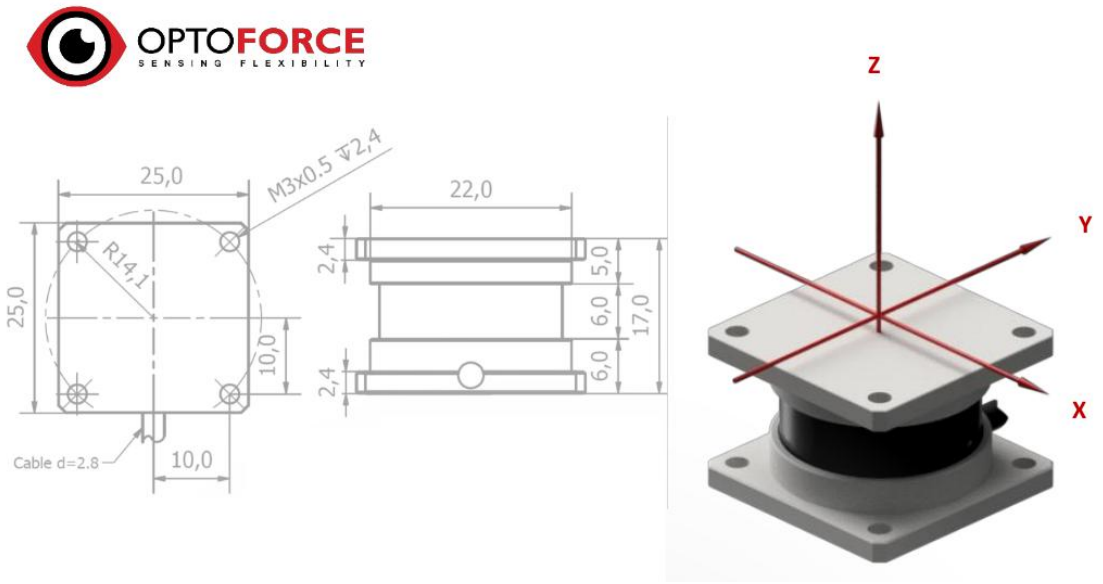


Figura 4.3: Optoforce OMD-20-FE-200N (*OMD-20-FE-200N DATASHEET V2.1* 2016)

Tabela 4.1: Capacidade do sensor Optoforce OMD-20-FE-200N

	Capacidade Nominal	Deformação Típica
F_{xy}	$\pm 20N$	$\pm 1.5mm$
F_z - Compressão	$200N$	$1.2mm$
F_z - Tensão	$100N$	$1mm$

4.5 Integração dos dispositivos

A malha de controle do manipulador TETIS é integrada através de *Controller Area Network* (CAN) e da interface USB do computador embarcado PCIe/104. O computador é o nó principal, recebendo dados de sensores e enviando comandos de controle para os drivers.

Através da interface USB, o computador recebe realimentação do sensor de força e vídeo da câmera. A realimentação dos *encoders* é feita através do barramento CAN, passando pelos *drivers* EPOS2. A Figura 4.4 ilustra a integração do manipulador ao sistema do robô.

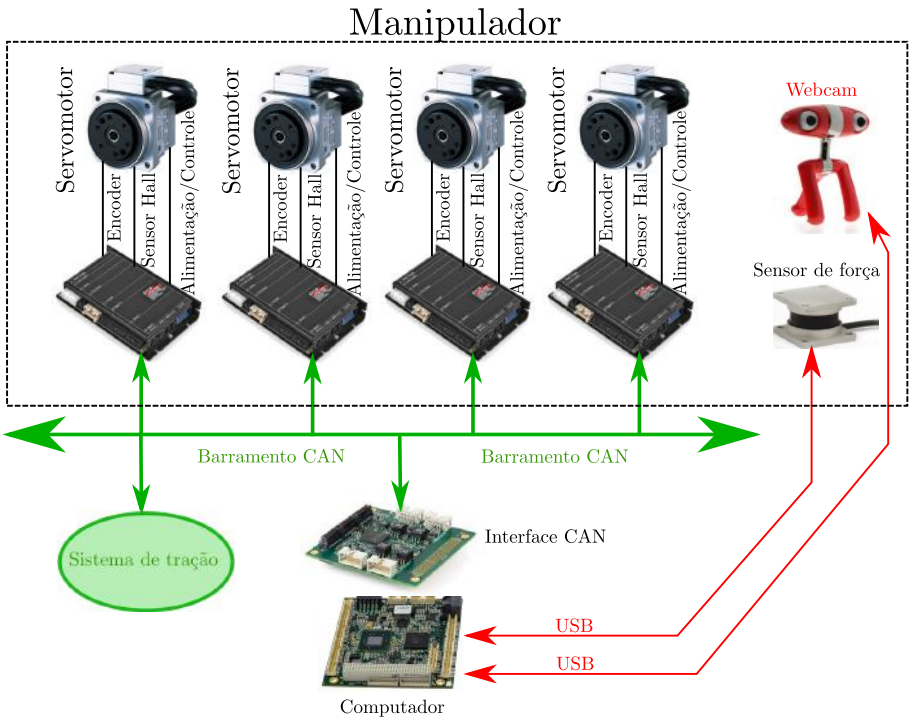


Figura 4.4: Esquema de integração dos dispositivos.

Capítulo 5

Implementação de Software para Controle de Sistemas Robóticos

A implementação dos algoritmos de controle detalhados nos capítulos anteriores for feita como extensão ao *software* RobotGUI desenvolvido pela equipe do LEAD-GSCAR (Nunes et al. 2013).

Proporciona uma infraestrutura para carregar componentes modulares de *software* para robôs, juntamente com classes altamente reutilizáveis. Esses sistema permite inicializar módulos de software com uma interface gráfica associada, de modo a interagir com eles. Por exemplo, ao iniciar componente que obtêm dados de um sensor, é iniciada na interface gráfica uma *Tool* que permite visualizar os dados e configurar parâmetros. Essa conexão é dinâmica, permitindo que componentes sejam iniciados de forma independente e em momentos diferentes.

5.1 Ferramentas

Os seguintes programas e *frameworks* foram utilizados:

- Linux (Ubuntu 14.04/16.04) como Sistema Operacional.
- C++ como linguagem de programação.
- Robot Operating System (Quigley et al. 2009) como *framework* principal utilizado pelo RobotGUI, fornecendo comunicação entre nós através de mensagens e serviços. Será descrito mais detalhadamente na próxima seção.
- Qt (*Qt Documentation* 2017) como *framework* para elaboração da interface gráfica.
- Julia Language como linguagem auxiliar na elaboração de algoritmos de controle modificáveis em tempo de execução.

5.2 Robot Operating System (ROS)

Escrever *software* para robôs tem se tornado particularmente difícil conforme a escala e o escopo dos projetos de robótica continua a crescer. Robôs podem ser de diferentes tipos e ter *hardware* completamente distinto, o que torna a reutilização de código difícil. Além disso, a quantidade de código necessário para um projeto de robótica pode ser intimidadora, indo desde o nível de *driver* até algoritmos de percepção e raciocínio, por exemplo. Como o nível de conhecimento e experiência necessário é maior do que um pesquisador sozinho pode ter, arquiteturas de *software* para robótica devem suportar integração.

Em (Quigley et al. 2009) é descrito o *framework* ROS, que busca enfrentar esses desafios. A enfase dada a projetos integrados de grande escala é útil em uma variedade de aspectos conforme sistemas robóticos tornam-se mais complexos. Os objetivos de projeto do ROS podem ser resumidos como:

- **Peer-to-peer ou ponto-a-ponto:** Um sistema construído utilizando ROS consiste de um conjunto de processos, potencialmente em diferentes máquinas hospedeiras e conectados entre si em uma topologia ponto-a-ponto. Tipicamente existe um ou mais computadores embarcados conectados via *ethernet*, que se comunicam via rede *wireless* com máquinas com maior poder computacional, capazes de realizar tarefas como visão computacional e reconhecimento de fala.

- **Multi-Linguagem:** Muitos possuem preferência em relação a determinadas linguagens de programação devido a fatores como tempo para escrita do código, facilidade de *debugging*, sintaxe ou eficiência de execução. ROS suporta as linguagens C++, Python, Octave e LISP. O ROS faz uma especificação a nível de mensagens. A configuração e negociação da conexão ponto a ponto ocorre através de XML-RPC [1], para o qual existem implementações na maioria das linguagens.

  É utilizada uma linguagem de definição de interface (IDL) que descreve as mensagens enviadas entre módulos. O código nativo de cada linguagem é gerado a partir dessas mensagens.

- **Baseado em ferramentas:** Para dar conta da complexidade do ROS seus desenvolvedores optaram por um *design microkernel*, em contraste a um *design* monolítico. Essas ferramentas cumprem tarefas como navegar pela árvore de

[1] RPC, do inglês, Remote Procedure Call é um termo da computação distribuída usado para quando um programa de computador causa a execução de um procedimento (sub-rotina) em outro espaço de endereçamento (outro computador na mesma rede).

código fonte, configurar parâmetros, visualizar a topologia dos nós, publicar e exibir mensagens, entre outros.

- **Fino:** Muitos projetos de *software* para robótica possuem *drivers* e algoritmos que poderiam ser reutilizados mas estão altamente "amarrados"ao programa em questão, de modo que é difícil extrair e reutilizar essas funcionalidades.

  No ROS, encoraja-se a encapsular todos os *drivers* e algoritmos em bibliotecas independentes.

- **Gratuito e código aberto:** Todo o código fonte do ROS está disponível publicamente. Isso é útil para facilitar a descoberta e correção de erros. É distribuído sob os termos da licença BSD.

5.2.1 Conceitos Básicos

Esta seção se baseia em (*ROS/Concepts - ROS Wiki* 2016) e busca detalhar conceitos do funcionamento e arquitetura do ROS necessários ao entendimento do *software* desenvolvido neste trabalho.

5.2.2 Nível de Grafo de Computação

Os conceitos a nível de grafo de computação descrevem elementos da rede ponto-a-ponto de processos ROS que estão em execução juntos. Esses elementos são descritos abaixo com seus nomes mantidos em inglês por fidelidade ao original.

Nodes: Nós são processos que executam algum tipo de computação. ROS foi projetado para ser modular, assim um sistema de controle de um robô geralmente tem múltiplos nós.

Master: Devido a topologia ponto-a-ponto é necessário um mecanismo de busca que os processos se encontrem. O mestre é um servidor de nomes, ou seja, fornece registro e busca de nomes.

Parameter Server: O servidor de parâmetros permite que dados sejam armazenados em uma localização central. Faz parte do *Master*.

Messages: Nós comunicam entre si passando mensagens. Uma mensagem consiste apenas de uma estrutura de dados. Mensagens suportam tipos primitivos, *arrays* de tipos primitivos ou outras estruturas aninhadas.

Topics: Um nó envia uma mensagem publicando a um dado tópico. O nome do tópico é utilizado para identificar o conteúdo da mensagem. Um outro nó que esteja interessado nesse conteúdo pode subscrever ao tópico. Podem existir

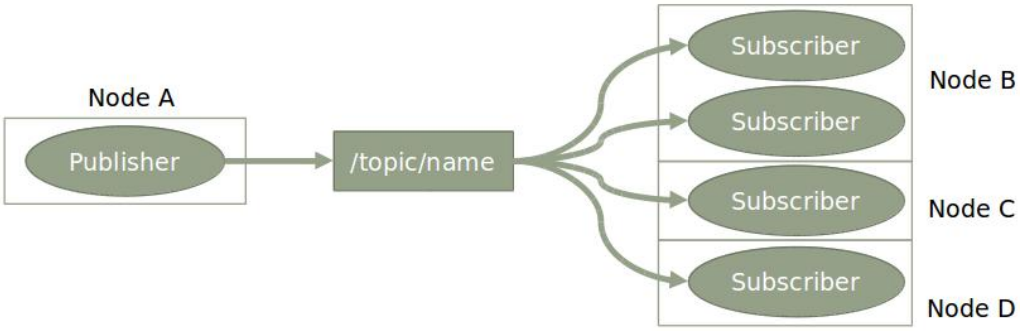


Figura 5.1: Topics

múltiplos *publishers* e *subscribers* para o mesmo tópico e um nó pode publicar e/ou subscrever a múltiplos tópicos.

Services: Em muitos casos é necessário ter um mecanismo de comunicação um para um de modo a lidar com interações do tipo pedido/resposta. Isso é feito através de serviços, que são definidos por um par de estruturas como as mensagens, uma para o pedido e outra para a resposta. Um nó oferece um serviço sob um nome e um cliente utiliza o serviço mandando um mensagem de pedido e esperando uma resposta. As bibliotecas do ROS apresentam essa interação na forma de um chamada de procedimento remoto (RPC).

Figura 5.2: Services

Bags: Bags são um formato para salvar e reproduzir dados de mensagens de ROS ao longo do tempo. São úteis para salvar dados de sensores, por exemplo, que são difíceis de coletar mas necessários para o desenvolvimento de algoritmos.

O *ROS Master* funciona como um servidor de nomes. Ele armazena informação de registro para os nós, que se comunicam com o Master para reportar sua informação de registro. Conforme os nós se comunicam com o *Master*, eles podem receber informação sobre outros nós registrados e fazer as conexões apropriadas.

Nós conectam-se diretamente a outros nós, o *Master* somente fornece informação de busca, como um servidor DNS. Nós que subscrevem a um tópico vão solicitar conexão aos nós que publicam naquele tópico e irão estabelecer a conexão de acordo com um protocolo de conexão concordado. O protocolo mais comum é chamado de TCPROS, que utiliza sockets TCP/IP padrão.

Essa arquitetura permite uma operação desacoplada. Nós, tópicos, serviços e parâmetros possuem nomes. Por meio deles sistemas maiores e mais complexos

podem ser construídos. É possível remapear nomes, de modo que um programa compilado pode ser reconfigurado para operar com uma topologia diferente.

5.2.3 Nodelets

O pacote Nodelet foi projetado de forma a permitir a execução de múltiplos algoritmos no mesmo processo sem *overhead* [2], devido a comunicação entre nós (não há custo de cópia). Esse pacote fornece a classe Nodelet, utilizada para implementar um nodelet e a classe nodelet::Loader utilizada para instanciar nodelets.

5.3 Arquitetura do RobotGUI

Primeiramente define-se como computador base aquele que será utilizado pelo operador para controlar e visualizar dados do robô. Define-se computador embarcado, ou do robô, o computador especificação PCIe/104, descrito no Capítulo 4. O programa executa nós diferentes no robô e na base.

A arquitetura do RobotGUI baseia-se nos seguintes conceitos principais (Nunes et al. 2013):

- **Components**: Lidam com a comunicação e processam dados no Computador Base. São essencialmente *Nodelets* de ROS, podendo utilizar funcionalidades como comunicação através de mensagens e serviços, utilizar parâmetros e bibliotecas de ROS. Permitem maior modularidade pois componentes podem ser inicializados independentemente do *RobotGUI*.

- **Tools:** São elementos gráficos da interface para o usuário. São utilizadas para interagir com o robô e visualizar informação. São criadas como *plugins* para o ROS *pluginlib* e independentes de qualquer biblioteca do ROS.

- **Devices:** São *Components* utilizados para representar um dispositivo do robô. Utilizam o mecanismo de *Item* e *Variable* e se conectam a *Tools* padrão fornecidas pelo RobotGUI, que são *DataTable*, *Device Management* e *Plotter*.

- **Item e Variable:** *Variables* contém valores de interesse, sempre guardando um valor desejado e um valor real. Por exemplo em uma variável controlada o valor desejado será a referência dada pelo operador e o valor real é obtido de algum sensor. Para mostrar esses valores na interface utiliza-se *Items*. Uma *Variable* pode estar associada a diversos *Items*, de forma que através desse mecanismo o valor exibido em todos eles é atualizado.

[2]Processamento ou armazenamento em excesso, seja de tempo de computação, de memória, de largura de banda ou qualquer outro recurso que seja gasto para executar uma determinada tarefa.

- **Interação entre Compontents e Tools:** Tools e Components podem se conectar, quando isso ocorre eles interagem entre si a nível de "ponteiro para objeto". Essa conexão permite que o desenvolvimento da interface através do *Qt* seja quase que independente do desenvolvimento do código que lida com *hardware*, lógica e comunicação. Essas classes disponibilizam uma estrutura para conectar *Components* e *Tools* de tal forma que quano um *Component* é iniciado ele busca por *Tools* associadas a ele que estejam em execução e vice versa.

- **RobotGUI:** É uma GUI [3] em nó de ROS. Permite que múltiplas janelas sejam criadas. O usuário pode decidir o que é exibido. Ao clicar nos botões na parte superior (Figura 5.5), uma janela referente a *Tool* aparece. Essa janela pode ser encaixada na janela principal. O RobotGUI executa *roslaunch*'s, nós incluídos pelo *roslaunch* em questão podem ser *Components*.

- **NodeletManager e RobotGUI:** o RobotGUI é um Nodelet Manager com pequenas modificações. Um NodeletManager utiliza a classe nodelet::Loader para instanciar Nodelets. Utilizando uma versão modificada dessa classe é possível verificar se um Nodelet a ser carregado é um *Component*. Se esse for o caso, a *Tool* correspondente será carregada e as conexões são feitas. Quando um componente é descarregado, a conexão com as *Tools* associadas a ele é encerrada.

5.3.1 Topologia

No diagrama da Figura 5.3 são mostrados os nós executados pelo programa para utilizar o manipulador e como se comunicam. Elipses representam Nodes, elipses tracejadas representam Nodelets e retângulos representam tópicos. Os retângulos que contém outros elementos representam os pacotes aos quais aqueles nós pertencem. Quando as setas saem de um nó, representam que o nó publica informação em um tópico, e quando chegam, representam que ele subscreve ao tópico em questão. As linhas pontilhadas representam que os *Nodelets* aos quais elas estão conectadas são executadas pelo *Node* na outra extremidade. Neste diagrama, são omitidos os tópicos de configuração de parâmetros.

Os *ROS Nodes* `manipulator_controller` e `usb_cam` são executados no computador embarcado. Os nós `robot_gui`, `visp_auto_tracker` são executados no computador base. Os *Nodelets* `manipulator_controller_device` e os `eposX_device` são carregados pelo `robot_gui`, que é um *Nodelet Manager*.

[3] Guided User Interface

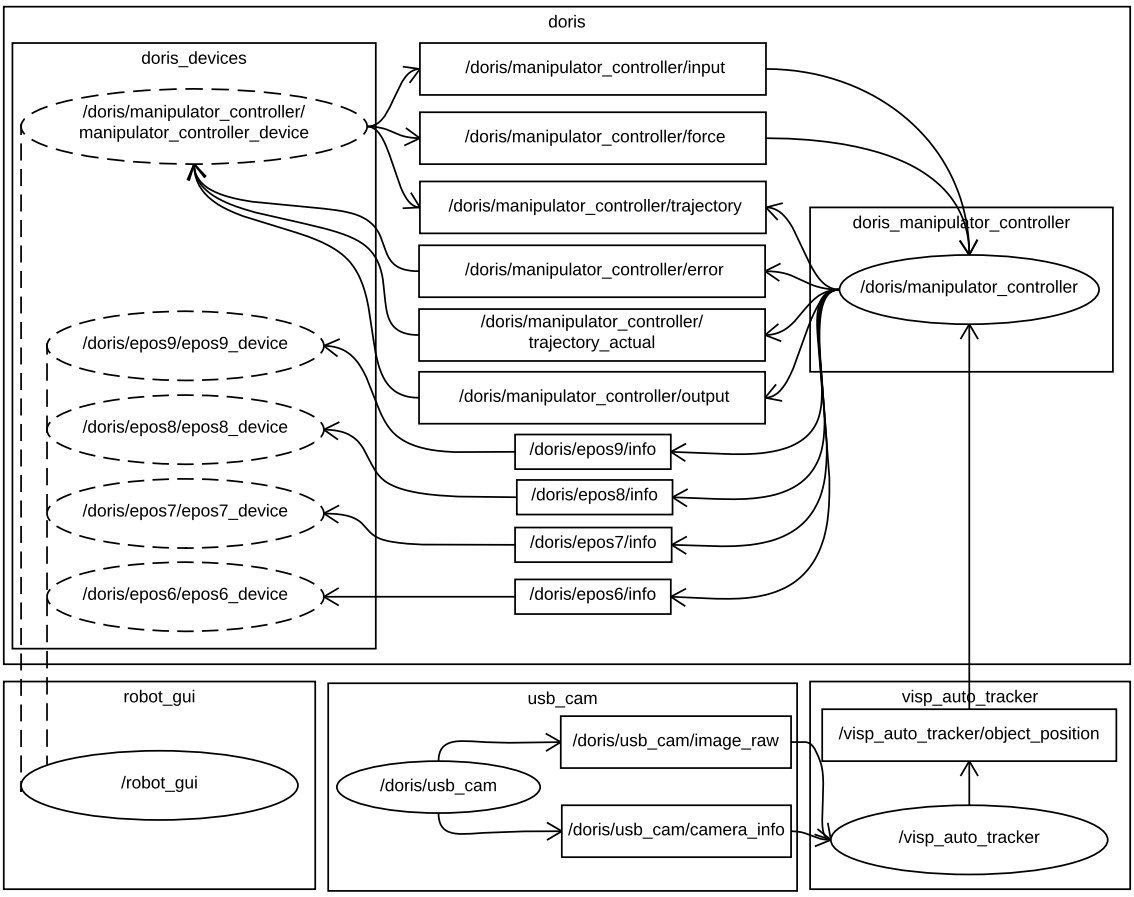


Figura 5.3: ROS Nodes

5.3.2 ManipulatorControllerDevice

A classe *ManipulatorControllerDevice* deriva de Device. Possui as *Variables* mostradas na tabela 5.1. Essa classe faz a interface entre *Tools* e os tópicos de ROS. Ao receber uma mensagem de ROS, um *callback* referente ao tópico em questão é executado atualizando a variável associada àquela informação. A atualização da variável resulta na atualização do *Item*.

5.3.3 DorisManipulatorController

A classe DorisManipulatorController deriva da classe Controller, que implementa a estrutura necessária para a criação de um controlador. Ela permite que o controle seja executado com um período fixo, em uma *thread* interna, ou que o ciclo de controle seja executado por uma chamada externa. Nesse caso é utilizada a primeira opção. Possui a funcionalidade de receber entradas de múltiplas fontes. Por exemplo, o manipulador pode ser controlado por uma janela na interface e por um *joystick*. O Controller é capaz de tratar isso, quando uma fonte fica inativa, outra é capaz de controlar.

Tabela 5.1: *Variables* do *ManipulatorControllerDevice*

Variable	Descrição
`coast`	Comando de parar o controle das juntas.
`home`	Comando de ir para posição inicial.
`record`	Comando de iniciar a gravação de Logs.
`zeroForce`	Comando de *reset* do sensor de Força.
`ktGain`	Ganho K_t para rastreamento de trajetória.
`kjGain`	Ganho K_j para controle no espaço das juntas.
`kvGain`	Ganho K_v para controle por servovisão.
`kfGain`	Ganho K_f para controle de força.
`pidParameters`	Parâmetros do controlador PID para controle de força.
`x`	Vetor $\boldsymbol{x_e}$ conforme definido em 3.3.
`error`	Erro conforme definido em 3.18.
`output`	Saída $\boldsymbol{u}$ do controlador.
`controlMode`	Modo de controle selecionado (detalhes em 5.3.4).
`individualMode`	Modo de controle individual selecionado (detalhes em 5.3.4).
`positionValue`	Valor de posição cartesiana (referência/medido).
`forceValue`	Valor de força (referência/medido).
`individualValue`	Valores para controle individual.
`openLoopVelocityEfct`	Valores para malha aberta de velocidade em E_e.
`openLoopVelocityBase`	Valores para malha aberta de velocidade em E_b.
`qValue`	Valor de $\boldsymbol{q}_d$ e $\boldsymbol{q}$ para controle no espaço das juntas.
`trajectory`	Trajetória na forma de código em *Julia Language*

A entrada do Controller é uma mensagem de controle com a seguinte forma.

Código 5.1: Control.msg

```
1 uint8 originId
2 uint8[] modes
3 float64[] data
```

O primeiro campo identifica a fonte da mensagem. O segundo é um *array* de modos de cotrole. O primeiro elemento é o modo principal, enquanto os demais, se existirem, são submodos. O terceiro elemento `data` é um *array* de elementos em ponto flutuante que contém a referência de controle. Pode ser vazio caso o modo de controle seja autônomo ou receba dados de outra fonte.

Seguindo os princípios do ROS, busca-se separar em classes a interação com ROS de algoritmos de controle. As classes com terminação em *Nodelet* herdam dessa classe e, portanto, implementam toda a comunicação (mensagens, serviços, etc.). A classe DorisManipulatorController implementa o ciclo de controle, chaveamento entre modos de controle, assim como os algoritmos de controle propriamente ditos. OptoforceDAQ é uma classe utilizada para fazer a aquisição de dados do sensor de força Optoforce descrito na Seção 4.4. A classe TetisData encapsula dados específicos do manipulador TETIS como dimensões dos elos e ângulos de juntas. A classe

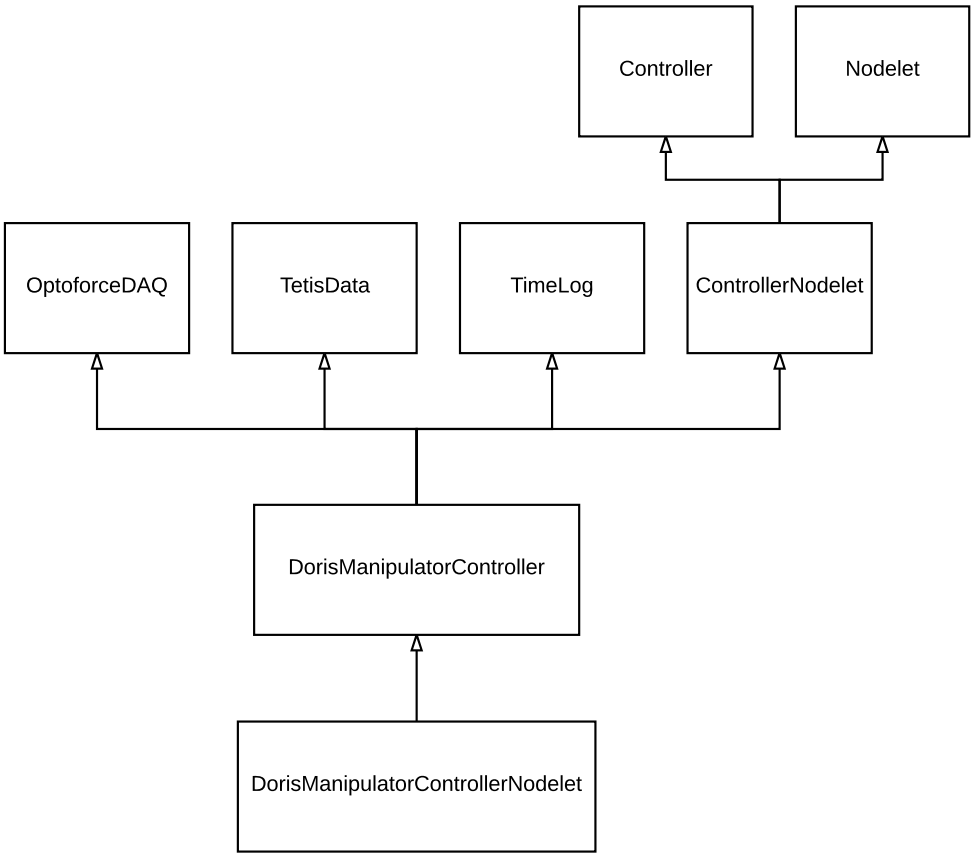


Figura 5.4: Diagrama de classes para o DorisManipulatorController

TimeLog é uma utilidade para salvar dados e gerar logs de forma automatizada.

5.3.4 Modos de controle

A classe *DorisManipulatorController* implementa os modos de controle descritos na tabela 5.2.

Tabela 5.2: Modos de controle

Modo de controle	Descrição
`NotControlled`	Nenhum sinal de controle é enviado ao manipulador.
`Brake`	Sinal de controle $u = 0$.
`VelocityEfct`	Implementa o controle descrito em 3.5.2.
`VelocityBase`	Implementa o controle descrito em 3.5.1.
`JointSpace`	Implementa o controle descrito em 3.6.
`OperationalSpace`	Implementa o controle descrito em 3.7.
`PplusFF`	Implementa o controle descrito em 3.8.
`Vision`	Implementa o controle descrito em 3.9.
`Force`	Implementa o controle descrito em 3.10.2.
`Individual`	Implementa o controle individual de juntas por gerador de função.
`MasterSlave`	Modo Master/Slave com o Sensable®Phantom Omni.

No Apêndice C é mostrada a implementação dos modos de controle `PplusFF`, `Vision` e `Force`.

5.3.5 DorisManipulatorTool

Para interagir com o manipulador é utilizada a *Tool* chamada DorisManipulatorTool. Ela permite alternar entre os modos de controle listados acima, ajustar parâmetros e ganhos dos controladores e alterar referências. A Figura 5.5 mostra a interface com DorisManipulatorTool a direita, Plotter no canto esquerdo superior, e DeviceManagement no canto esquerdo inferior.

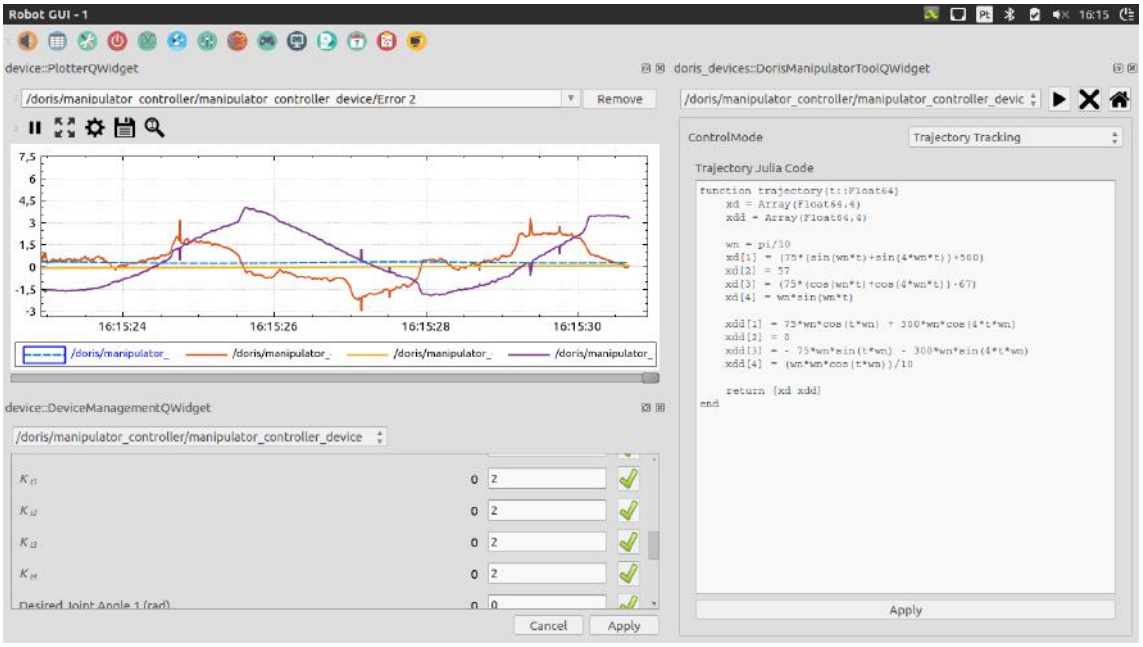


Figura 5.5: RobotGUI com três *Tools*: DorisManipulatorTool, Plotter e DeviceManagement

5.4 Julia Language

Julia é uma linguagem dinâmica de alto nível e alta performance, projetada para computação técnica. Possui uma sintaxe familiar para usuários de outras linguagens orientadas a computação científica. Dentre as funcionalidades que a linguagem proporciona, destacam-se para a aplicação neste projeto:

- Boa performance, aproximando-se de linguagens estaticamente compiladas como C.
- Tipagem dinâmica: tipos podem ser utilizados para documentação e otimização, mas podem ser omitidos.

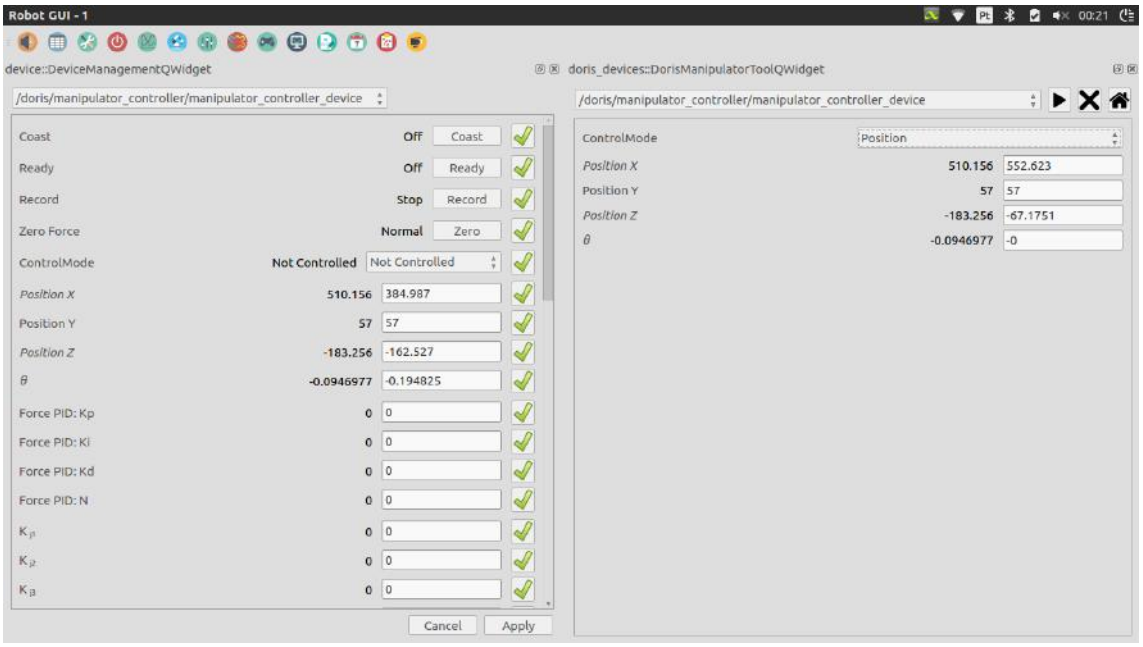


Figura 5.6: RobotGUI com DorisManipulatorTool e DeviceManagement

- Compilador *just-in-time* [4] de alta performance.
- Possibilidade de integrar Julia a projetos C/C++.
- Gratuito e código aberto (Licença MIT).

A compilação *just-in-time* é baseada no projeto LLVM (Lattner & Adve 2004, *The LLVM Compiler Infrastructure* n.d.), que consiste em uma coleção de tecnologias e ferramentas de compilação modulares e reutilizáveis. O uso dessas tecnologias, juntamente com a forma como a linguagem foi projetada permite que em muitos casos ela tenha uma performance comparável com a linguagem C e ao mesmo tempo ter uma sintaxe comparativamente simples.

Portanto, a linguagem Julia mostrou-se uma opção interessante e mais dinâmica para a implementação de controle no *software* do TETIS, que requer desempenho mas pode se beneficiar de uma linguagem dinâmica. Uma prova de conceito foi feita para a configuração das equações da trajetória a ser rastreada. Na interface uma janela permite a escrita de equações na forma de uma função Julia. Ao aplicar as alterações esse código é enviado para o computador embarcado através do ROS. O programa principal, em C++, chama a API em C do Julia e compila essa função no momento da primeira execução.

Dessa forma a trajetória pode ser alterada em tempo de execução sem prejudicar a performance e sem necessitar acesso a dados externos constantemente. O Código 5.2 mostra a trajetória de teste em *Julia Language*.

[4] Compilação just-in-time (JIT) refere-se a compilação feita no momento de execução do programa, em contraste a compilação anterior a execução.

Código 5.2: trajectory.jl

```
function trajectory(t::Float64)
        xd = Array(Float64,4)
        xdd = Array(Float64,4)

        wn = pi/10
        xd[1] = (75*(sin(wn*t)+sin(4*wn*t))+500)
    xd[2] = 57
    xd[3] = (75*(cos(wn*t)+cos(4*wn*t))-67)
    xd[4] = wn*sin(wn*t)

    xdd[1] = 75*wn*cos(t*wn) + 300*wn*cos(4*t*wn)
    xdd[2] = 0
    xdd[3] = - 75*wn*sin(t*wn) - 300*wn*sin(4*t*wn)
    xdd[4] = (wn*wn*cos(t*wn))/10

        return [xd xdd]
end
```

Capítulo 6

Resultados de Simulação e Experimentais

Neste capítulo são exibidos os resultados de simulação numérica e ensaios experimentais. A simulação mostrada é de um rastreamento de trajetória no espaço operacional, pois é representativa do comportamento do sistema. Para experimentos com o manipulador TETIS foram feitos testes em varias etapas, em crescente nível de complexidade. Iniciando com uma análise da malha interna de controle, seguida de controle no espaço das juntas, controle proporcional de posição no espaço operacional, controle proporcional com feedforward e controle de força. As trajetórias utilizadas nos testes são definidas a seguir.

Trajetória 1

$$x_d = \begin{bmatrix} 75\sin(\omega_n t) + \sin(4\omega_n t) + 500 \\ 57 \\ 75\cos(\omega_n t) + \cos(4\omega_n t) - 67 \\ \omega_n \sin(\omega_n t) \end{bmatrix} \dot{x}_d = \begin{bmatrix} 75\omega_n \cos(t\omega_n) + 300\omega_n \cos(4t\omega_n) \\ 0 \\ -75\omega_n \sin(t\omega_n) - 300\omega_n \sin(4t\omega_n) \\ \omega_n^2 \cos(t\omega_n) \end{bmatrix} \tag{6.1}$$

onde $\omega_n = \pi/10$

Trajetória 2

$$x_d = \begin{bmatrix} \frac{100\cos(t)}{\sin^2(t)+1} + 500 \\ 57 \\ \frac{100\cos(t)\sin(t)}{\sin^2(t)+1} - 50 \\ 0 \end{bmatrix} \qquad \dot{x}_d = \begin{bmatrix} \frac{100\sin(t)(\sin^2(t)-3)}{(\sin^2(t)+1)^2} \\ 0 \\ \frac{-(300\sin^2(t)-100)}{(\sin^2(t)+1)^2} \\ 0 \end{bmatrix} \tag{6.2}$$

6.1 Simulação

Utilizando a Robotics Toolbox (Corke 2011) para MATLAB®, foram feitas simulações para o controle cinemático utilizando controle proporcional com feedforward e a trejetória 1 (6.1) como referência. Para tornar os resultados compatíveis com o sistema real, foi utilizado um ZOH na saída do controlador com período de amostragem de $10ms$ (ciclo de controle do software de controle).

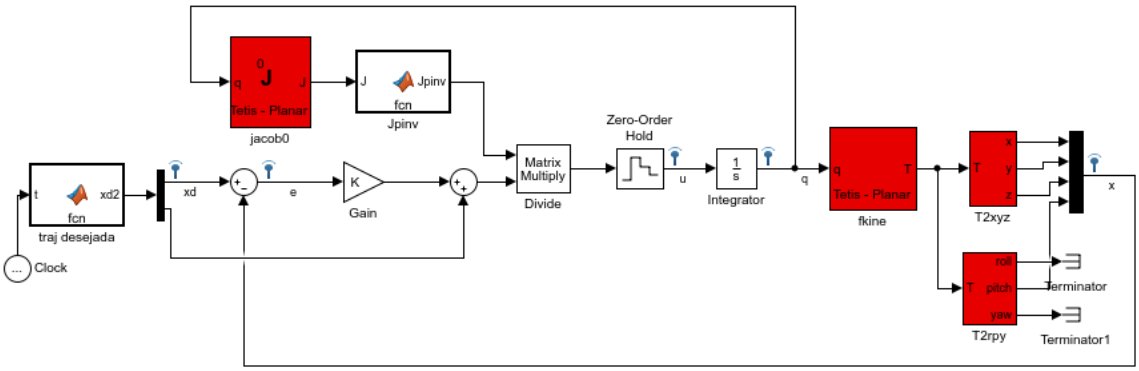


Figura 6.1: Diagrama de blocos Simulink utilizado para simulação.

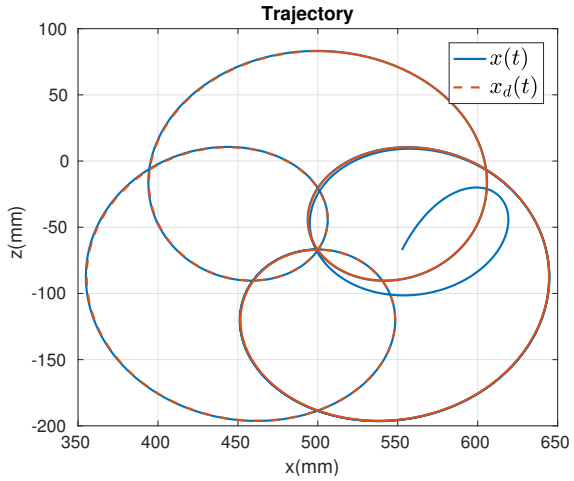
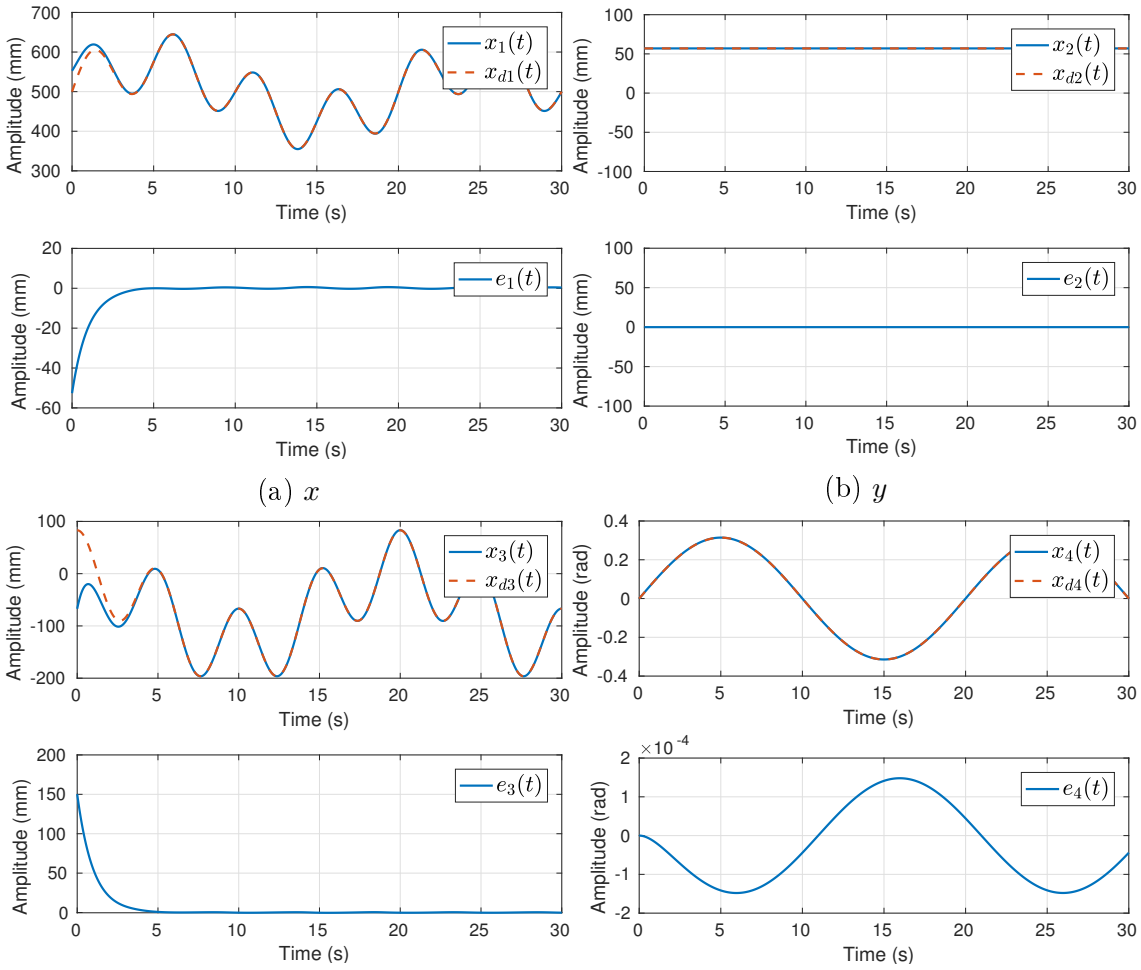


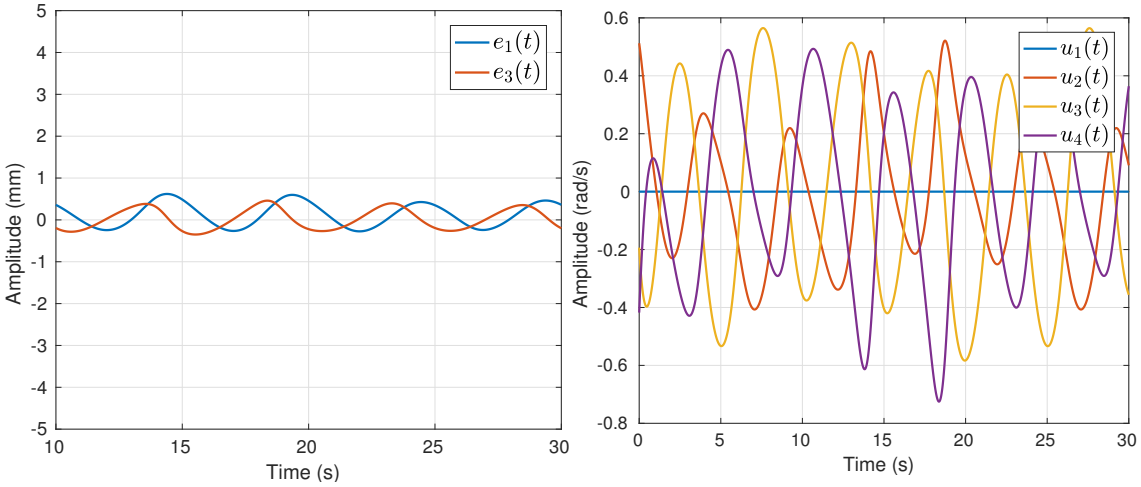
Figura 6.2: Simulação: Trajetória 1 no plano x-z



(a) x (b) y

(c) z (d) ϕ

Figura 6.3: Simulação - Rastreamento da trajetória 1 para $K_t = I$



(a) e_1 e e_3 (b) $u(t)$

Figura 6.4: Simulação - Destaque para o erro e_1 e e_3 com $K_t = I$

6.2 Análise da Malha de Controle a nível de Juntas

Para verificar se de fato são válidas as premissas assumidas na Seção 2.4 para aplicação de uma estratégia de controle cinemático foi levantada para cada uma das juntas a resposta a uma onda quadrada tal que:

$$u = A\,\mathrm{sgn}(\sin(2\pi t/f))$$

onde o período $T = 1/f = 2s$ e a amplitude $A = 0.5rad/s$

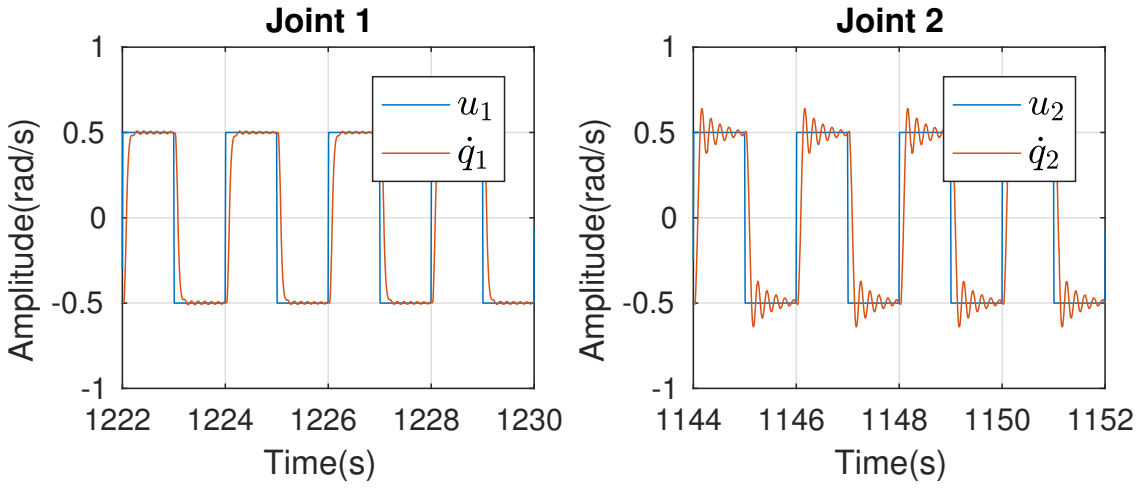


Figura 6.5: Resposta de velocidade das juntas 1 e 2

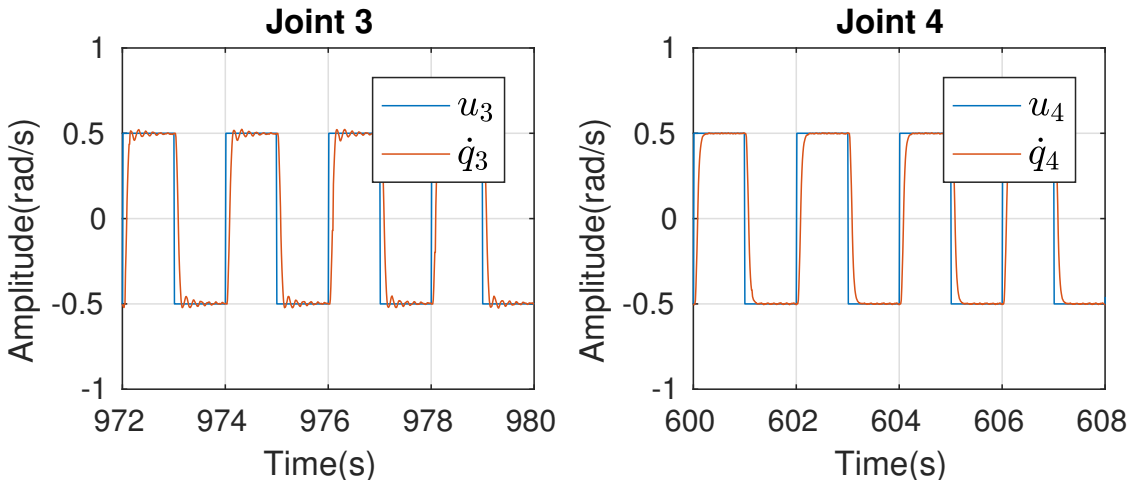


Figura 6.6: Resposta de velocidade das juntas 3 e 4

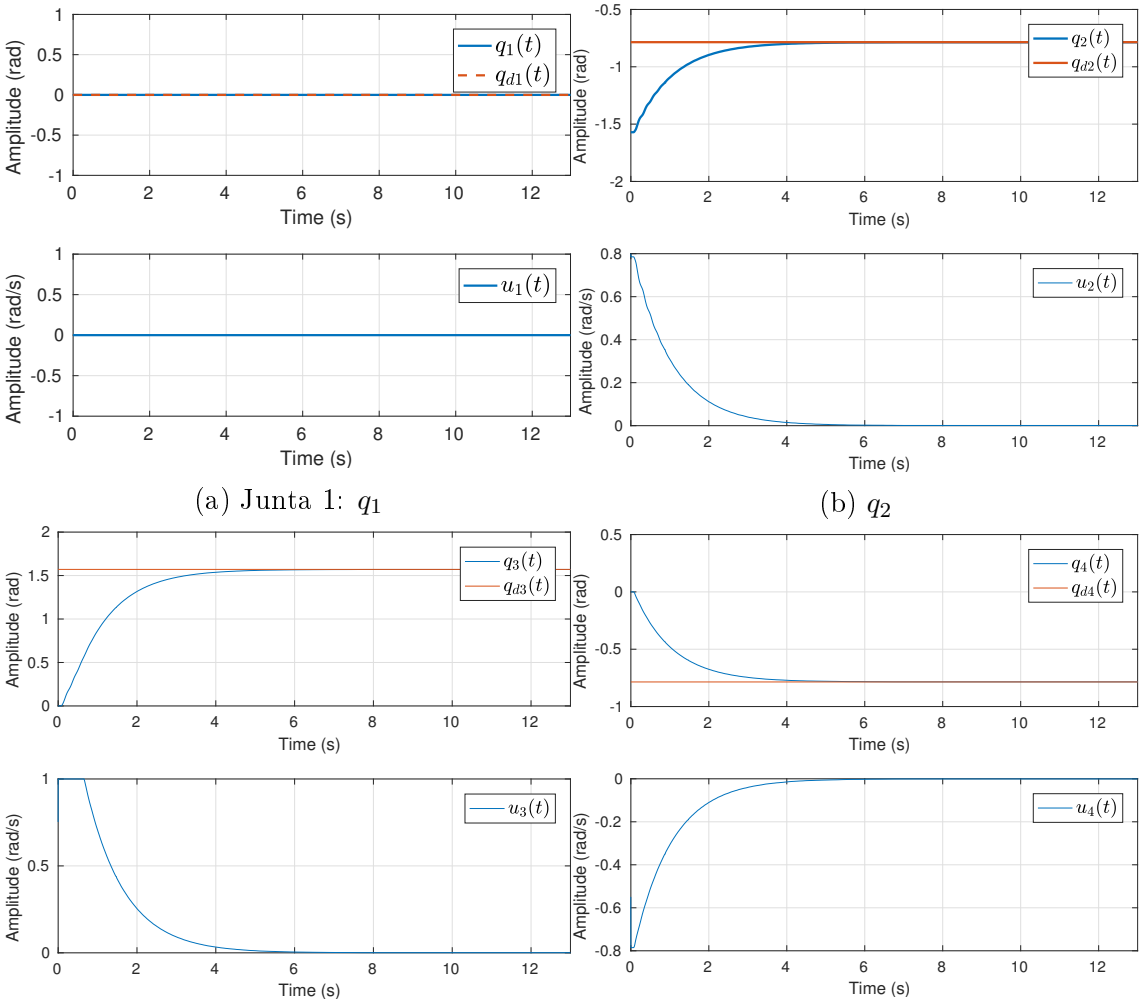
Observa-se que para as juntas 1, 3 e 4 a resposta é de primeira ordem e que são capazes de reproduzir com precisão os comandos de velocidade. A junta 2 observa-se que existe uma dinâmica. Isso se deve ao fato de a junta 2 necessitar de um torque maior para vencer a ação da força gravitacional. O controle cinemático produz resultados satisfatórios somente quando não são necessários movimentos muito rápidos

ou acelerações muito grandes.

6.3 Controle de Posição no Espaço das Juntas

Neste experimento foi testado controle de posição no espaço das juntas, conforme definido em 3.6. Partindo da posição $q = [0 \quad -\pi/2 \quad 0 \quad 0]^T$, deseja-se atingir a posição $q = [0 \quad -\pi/4 \quad \pi/2 \quad -\pi/4]^T$. Foi utilizado um ganho $K_j = I$.

Os resultados foram conforme esperados, levando o erro assintoticamente a zero em todas as juntas.



(a) Junta 1: q_1 (b) q_2

(c) q_3 (d) Junta 4

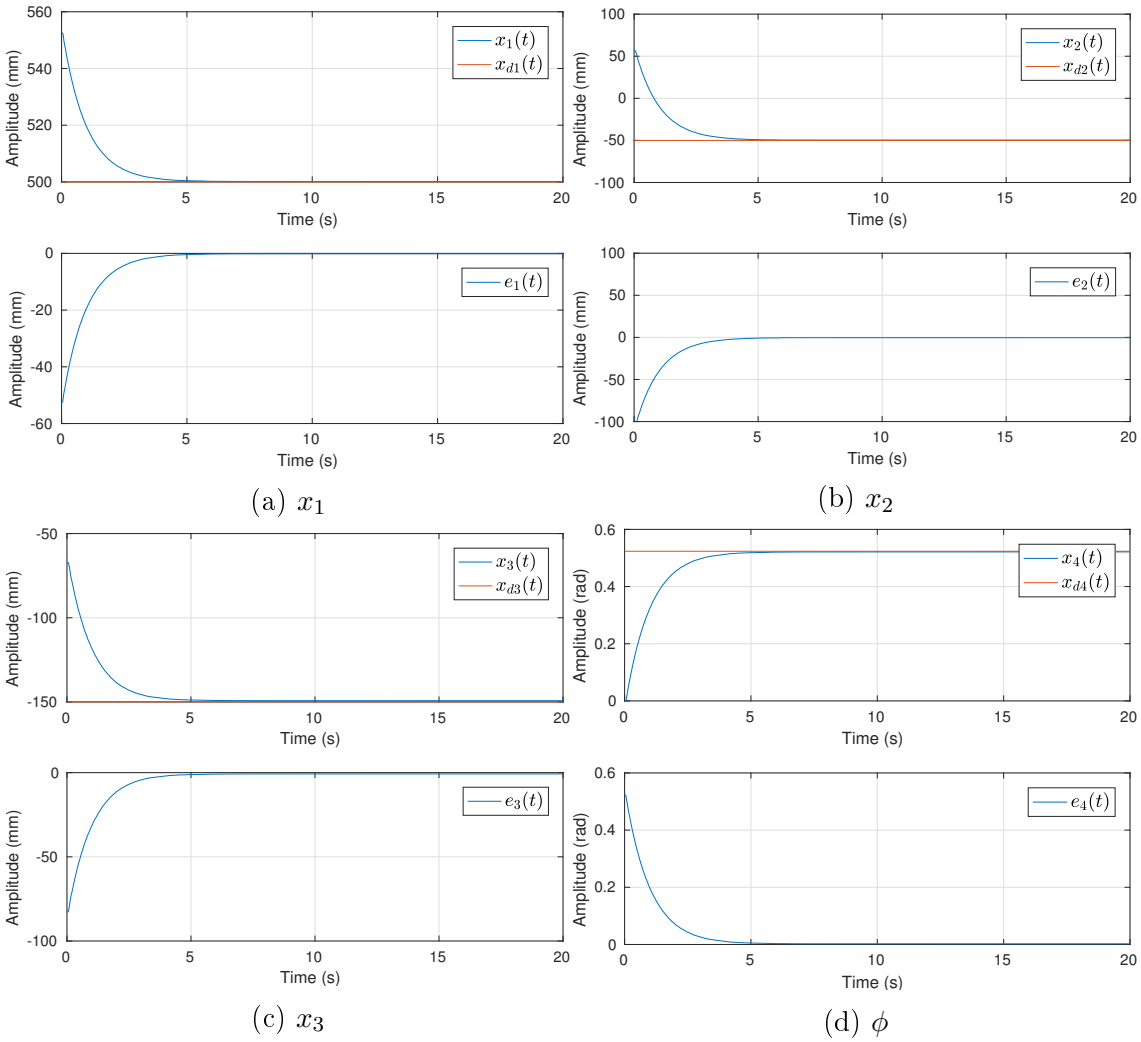
Figura 6.7: Resultados experimentais - Controle de Posição no Espaço das Juntas

6.4 Controle de Posição no Espaço Operacional

Nesta seção, demonstra-se o controle no espaço operacional. Utilizando a estratégia de controle proporcional conforme definido na Seção 3.7.

6.4.1 Experimento 1

Neste experimento a posição inicial é $x = [552.62\ 57\ -67.17\ 0]^T$ e deseja-se alcançar $x_d = [500\ -50\ -150\ 0.5235]^T$. Utiliza-se um ganho $K_p = I$. A figura 6.8 mostra que para o resultado para o problema de *set-point*, o sistema foi capaz de rastrear assintoticamente um sinal de referência constante.



(a) x_1 (b) x_2

(c) x_3 (d) ϕ

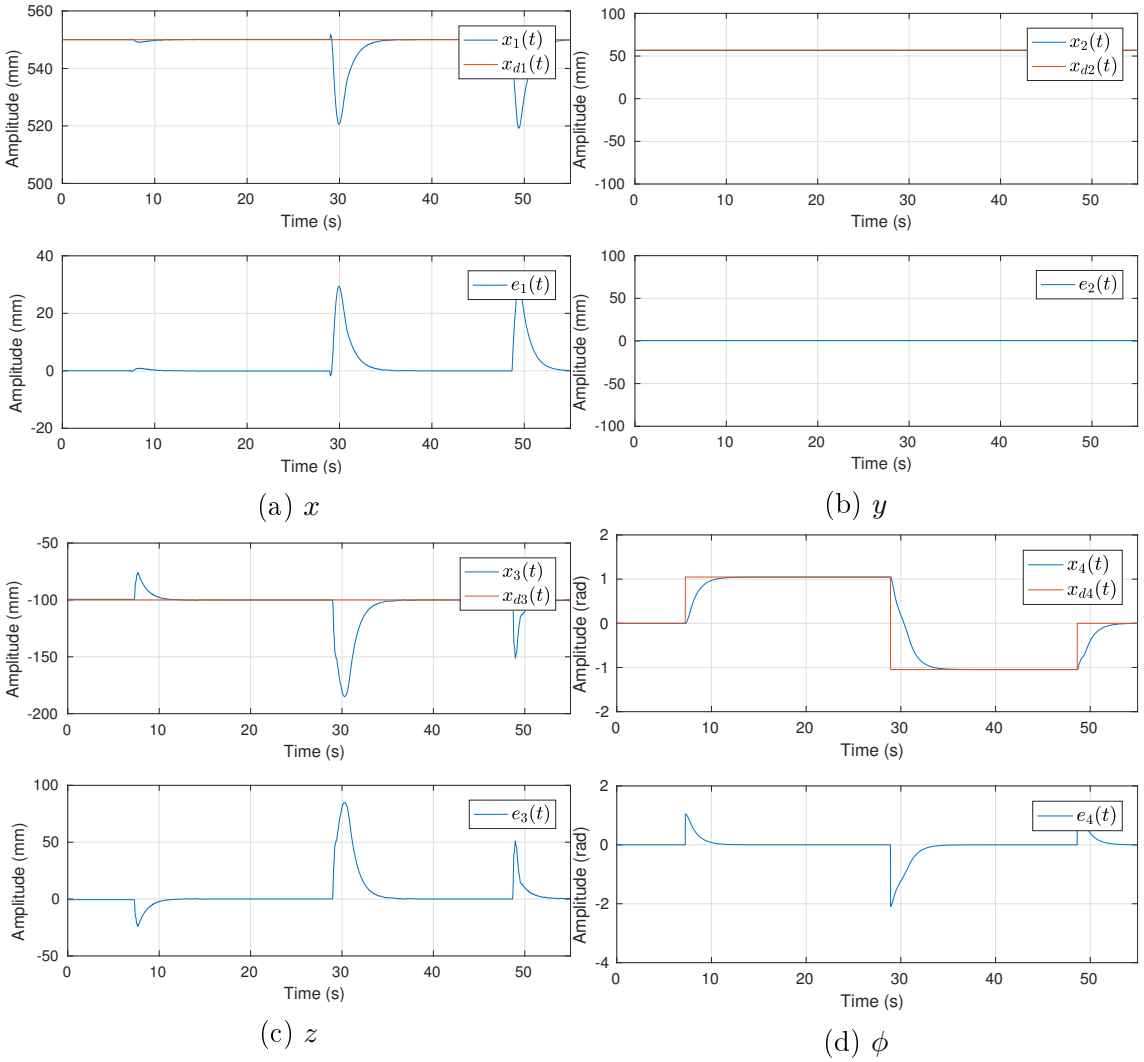
Figura 6.8: Resultados experimentais - Controle de Posição no Espaço Operacional: Experimento 1

6.4.2 Experimento 2

Neste experimento, o objetivo é testar a alteração da orientação, mantendo a posição constante. A posição inicial é $x = [550\ 57\ -100\ 0]^T$. Como mostra o gráfico na Figura 6.9d, a orientação é alterada para 60^o, ou seja $x_d = [550\ 57\ -100\ 1.0472]^T$. Em seguida, a orientação é alterada para -60^o, ou seja $x_d = [550\ 57\ -100\ -1.0472]^T$.

Nas Figuras 6.9a e 6.9c é possível identificar que a mudança de orientação gerou distúrbios nas dimensões x e z. No entanto, os distúrbios foram rejeitados assinto-

ticamente.



(a) x (b) y

(c) z (d) ϕ

Figura 6.9: Resultados experimentais - Controle de Posição no Espaço Operacional: Experimento 2

6.5 Rastreamento de Trajetória

Nesta seção são exibidos os resultados experimentais para o rastreamento das duas trajetórias definidas em (6.1) e (6.2). Utiliza-se uma estratégia de controle proporcional com *feedforward*, conforme definido na Seção 3.8.

6.5.1 Trajetória 1

Ganho $K_t = I$

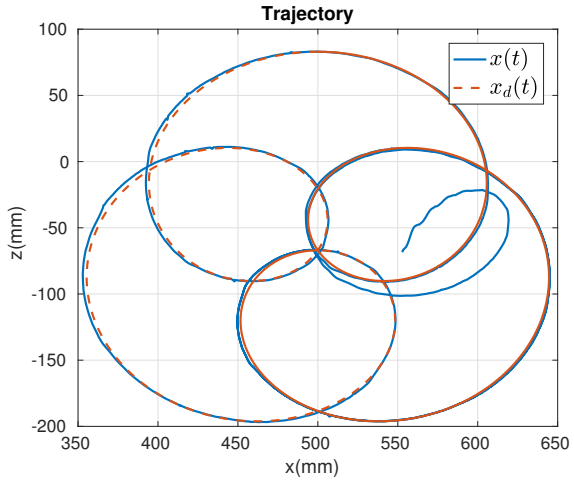
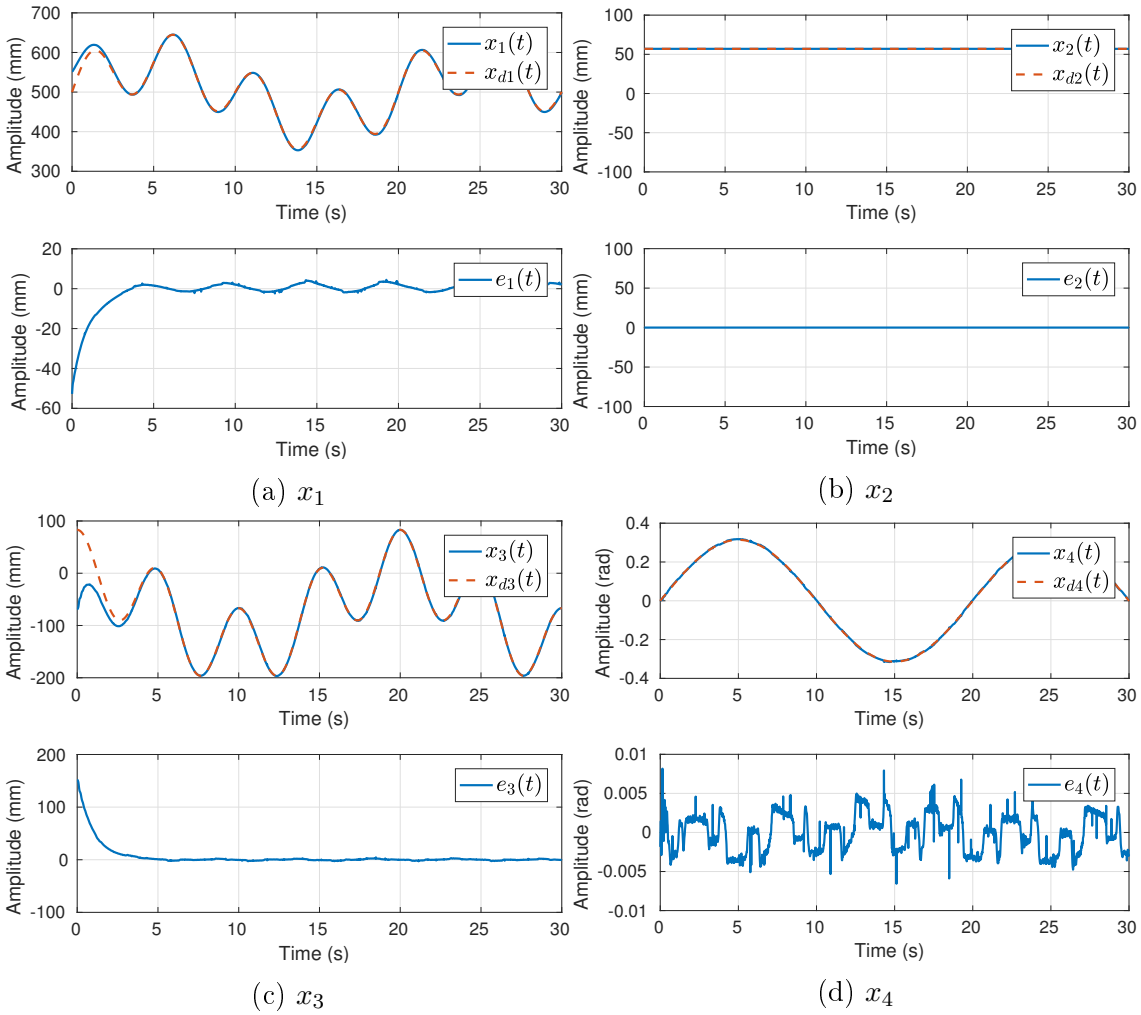


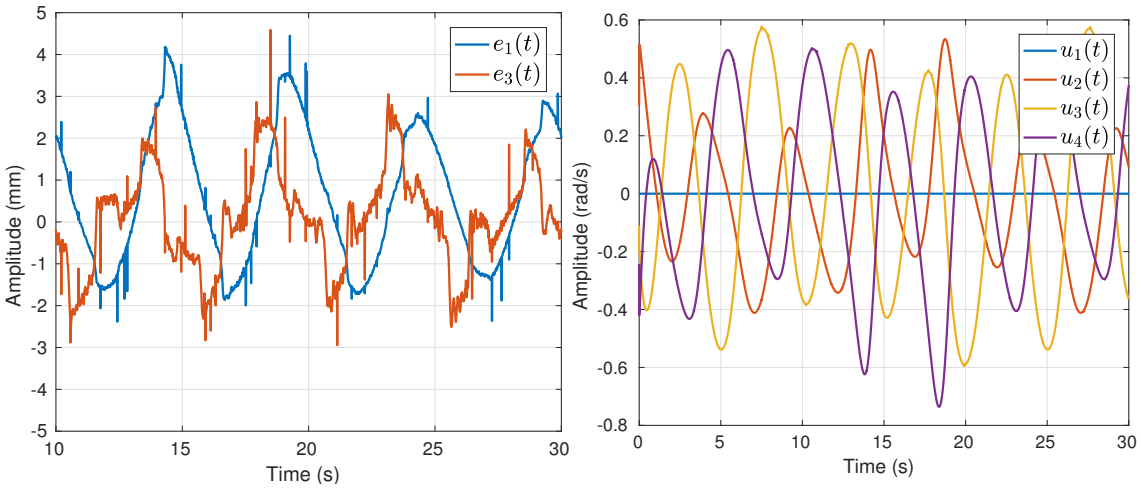
Figura 6.10: Trajetória 1 no plano x-z: Resultados experimentais com $K_t = I$



(a) x_1 (b) x_2

(c) x_3 (d) x_4

Figura 6.11: Resultados experimentais - Rastreamento da trajetória 1 para $K_t = I$



(a) e_1 e e_3 (b) $u(t)$

Figura 6.12: Resultados experimentais - Trajetória 1: destaque para o erro e_1 e e_3 com $K_t = I$

Podem-se observar diversos picos no sinal de erro na Figura 6.18 que se devem ao fato de, por não se tratar de um sistema em tempo-real, o período do laço de controle pode sofrer variação caso aconteça alguma interrupção no sistema operacional. Assim, o comando não seria atualizado, resultando em um erro maior na próxima iteração.

6.5.2 Trajetória 2

Ganho $K_t = I$

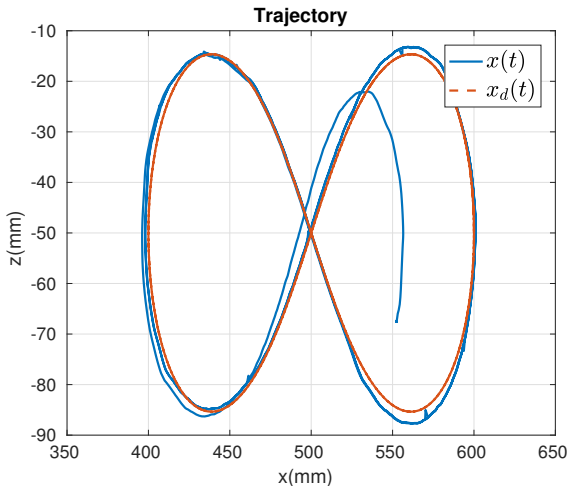
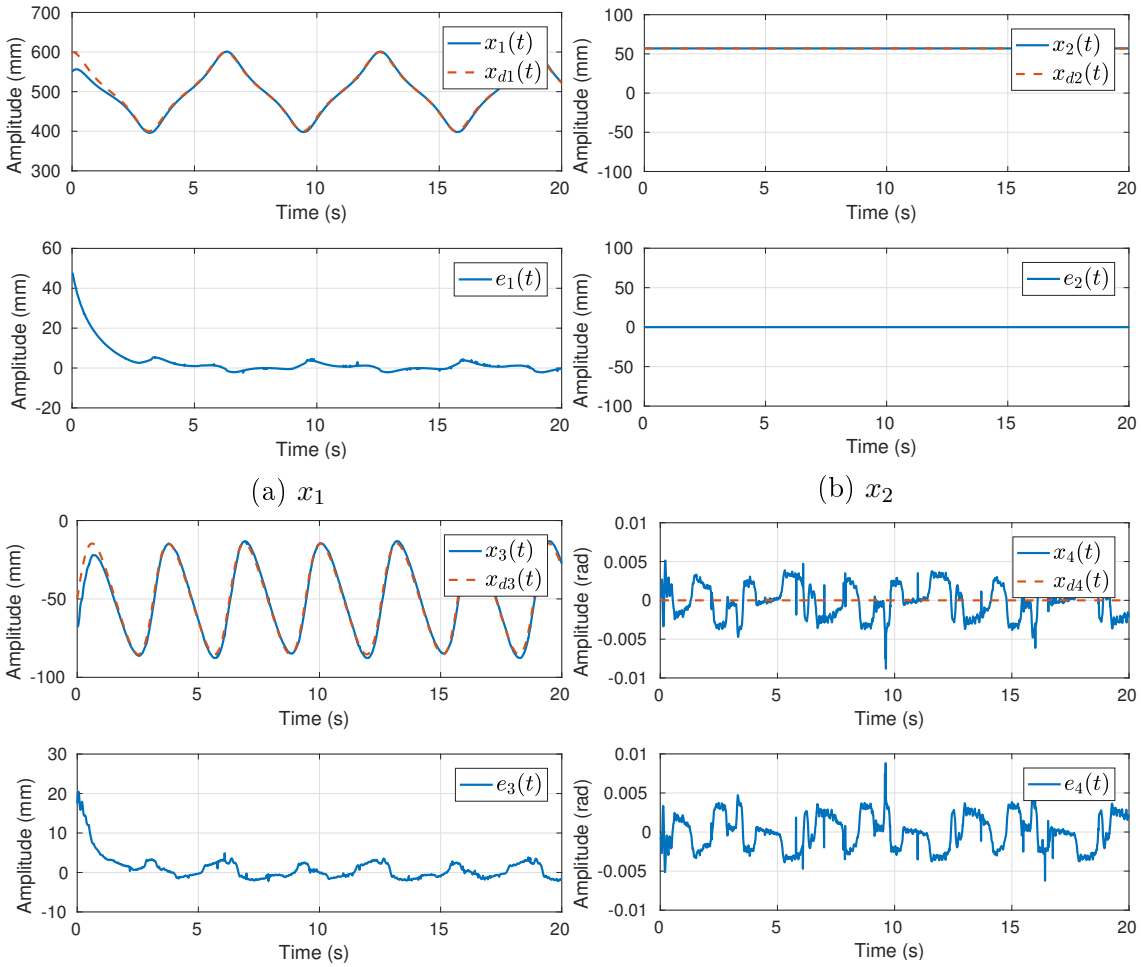


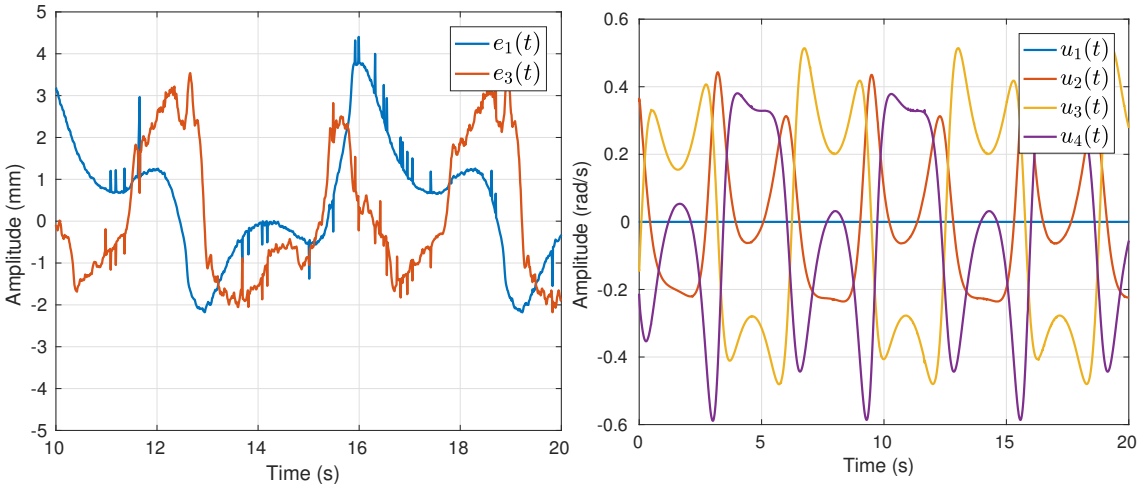
Figura 6.13: Resultados experimentais - Trajetória 2 no plano x-z



(a) x_1 (b) x_2

(c) x_3 (d) x_4

Figura 6.14: Resultados experimentas - Rastreamento da trajetória 2 para $K_t = I$



(a) e_1 (b) e_3

Figura 6.15: Resultados experimentais - Trajetória 2: Destaque para o erro e_1 e e_3 com $K_t = I$

Ganho $K_t = 5I$

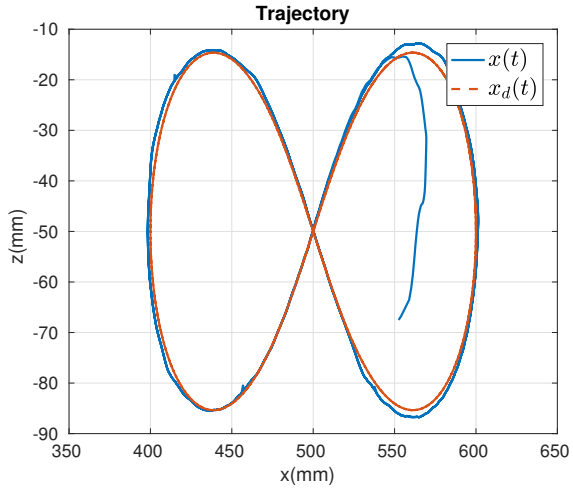
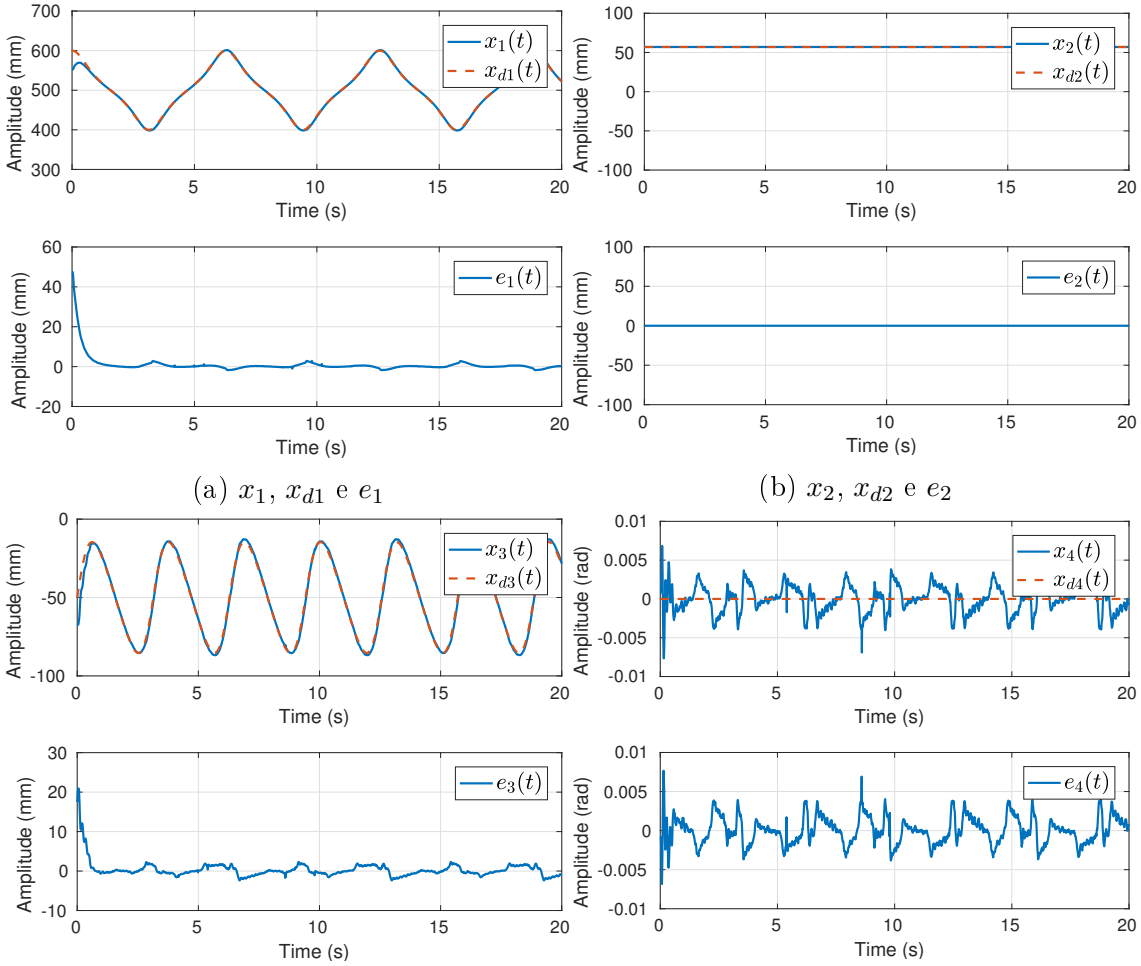


Figura 6.16: Resultados experimentais - Trajetória 2 no plano x-z para $K_t = 5I$



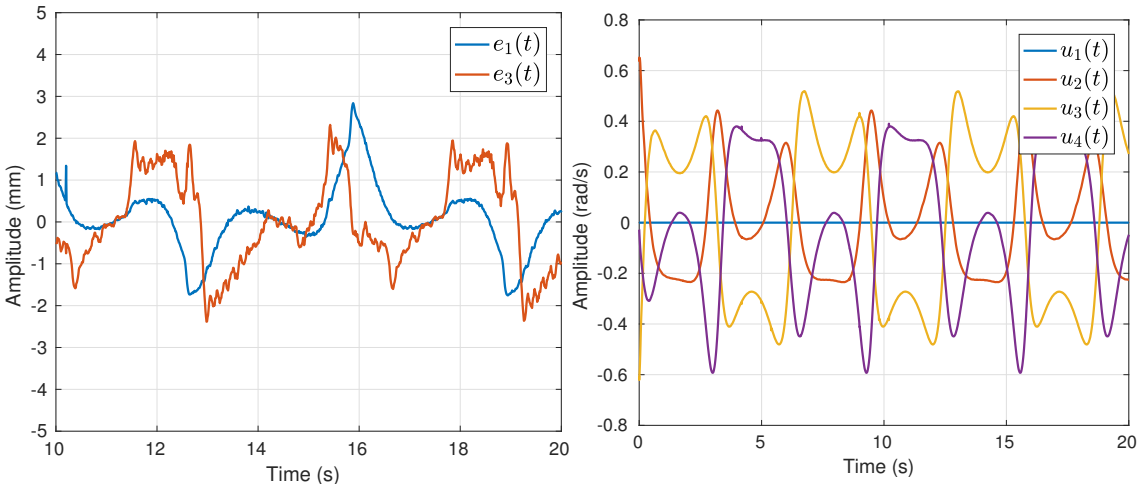
(a) x_1, x_{d1} e e_1

(b) x_2, x_{d2} e e_2

(c) x_3, x_{d3} e e_3

(d) x_4, x_{d4} e e_4

Figura 6.17: Resultados experimentais - Rastreamento da trajetória 2 para $K_t = 5I$



(a) e_1 (b) e_3

Figura 6.18: Resultados experimentais - Trajetória 2: destaque para o erro e_1 e e_3

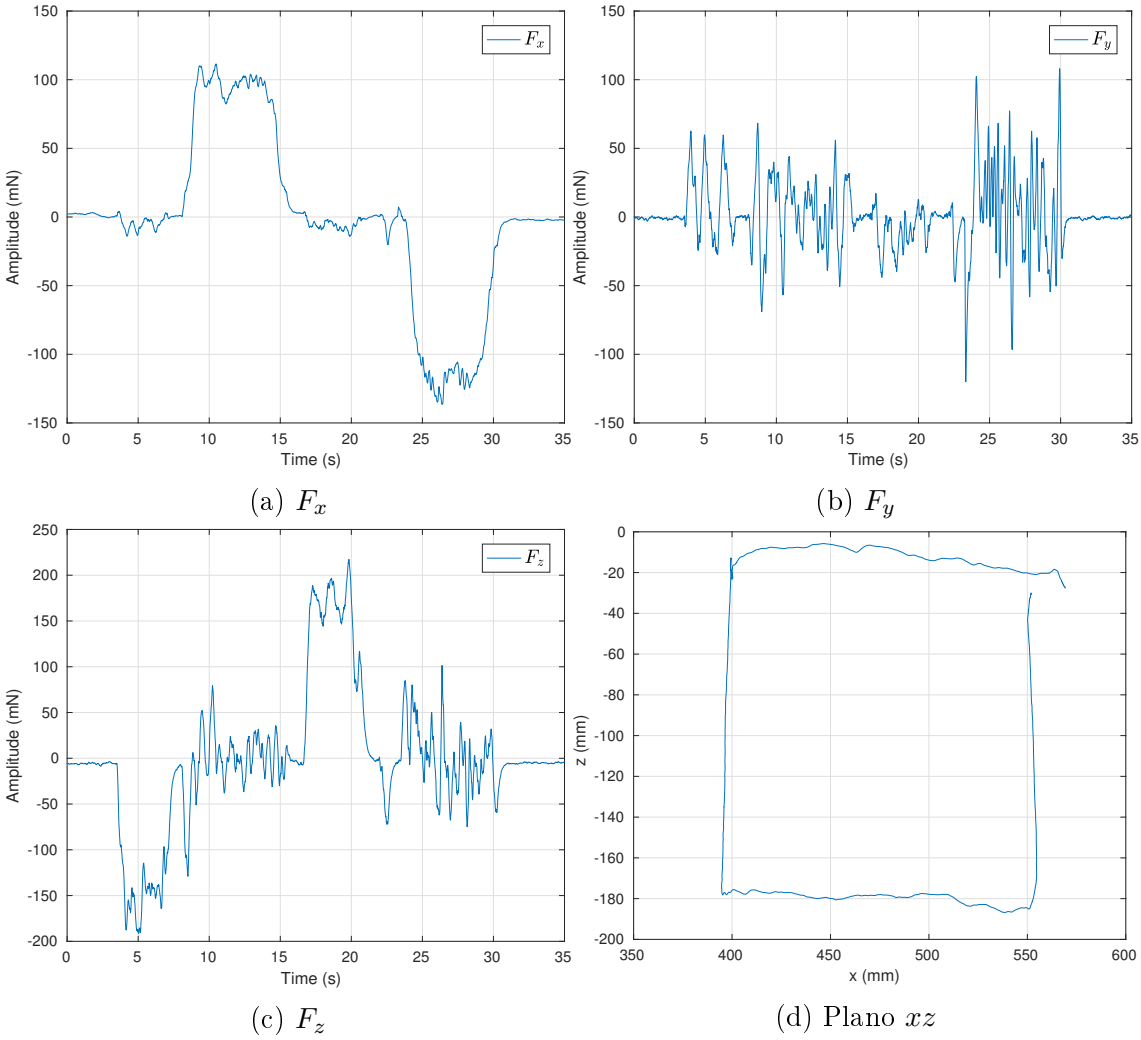
A partir da variação do ganho de controle foi possível observar que acima de $K_t = 5I$ não resultaram em erro menor. Portanto, conclui-se que o erro em regime permanente se deve aos seguintes fatores (i) não é possível garantir que o período de controle é constante pois não se trata de um sistema em tempo real, (ii) a junta 2 não é capaz de reproduzir com tanta precisão comandos de velocidade e (iii) período máximo que o computador embarcado consegue executar o ciclo de controle é de 10ms.

6.6 Controle de Força

Nesta seção são mostrados os resultados de experimentos de controle de força, segundo os estratégias definidas na Seção 3.10.

6.6.1 Float

Neste experimento a referência de força é configurada para zero, logo é possível mover o efetuador aplicando forças sobre o sensor de força. O operador tentou traçar um retângulo no plano x-z, movimentando o efetuador com as mãos.



(a) F_x (b) F_y

(c) F_z (d) Plano xz

Figura 6.19: Resultados experimentais - Controle de Força: Float

6.6.2 Approach

Considerando o problema de aplicar uma força em uma placa de poliestireno fixada em um suporte plano na vertical. Conforme mostrado na Seção 3.10.2 a força de contato pode ser modelada pela Lei de Hooke, portanto é necessário saber a constante elástica do ambiente.

Foi feito um ensaio para encontrar a constante k_s variando a distância x_s e medindo a força resultante. Após o contato com a superfície, foi variada a distância x_s de x_{si}, a distância inicial de contato, até $x_{si} + 10mm$, em incrementos de $1mm$. Os dados experimentais e a reta obtida por regressão linear pode ser vista na Figura 6.20. A constante encontrada foi $k_s = 379.7642N/m$.

Foi aplicado o controle de força com uma referência constante de $1000mN$ na direção de *approach*, sobre uma placa de poliestireno utilizando ganho do controlador PI de $k_p = 2$ e $k_i = 0.05$.

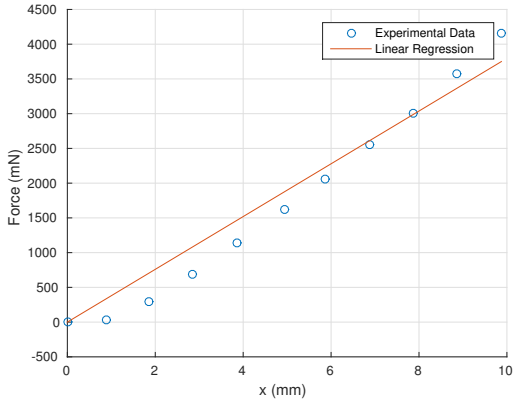


Figura 6.20: Estimação da constante elástica k_s por regressão linear.

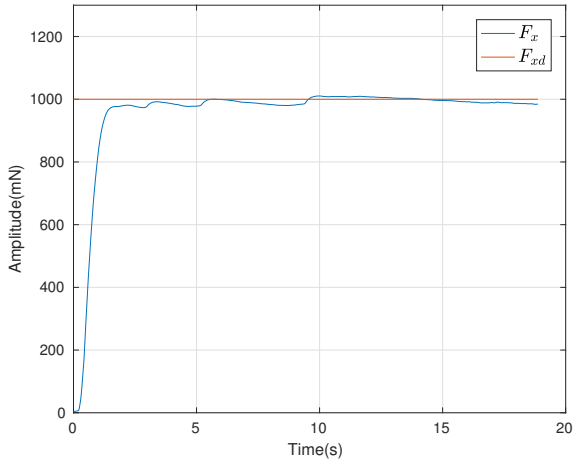


Figura 6.21: Resultados experimentais - Controle de força: Approach Fx

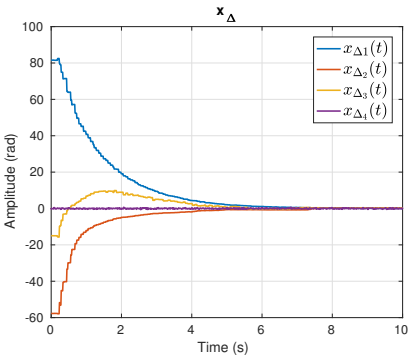
Os resultados mostraram a existência de não linearidades que não foram consideradas no modelo. Por exemplo, a relação entre força e distância no ambiente testado não é exatamente linear, no entanto a aproximação foi aceitável. O fato de o sensor de força ter uma alta relação sinal-ruído para a ordem de grandeza do sinal de referência também contribuiu para os resultados observados.

6.7 Controle por Servovisão

Considerando o controle por servovisão conforme definido na Seção 3.9, foi realizado um experimento em que o operador posicionou o alvo a frente do efetuador, movimentando-o nos primeiros segundos de forma a aproximá-lo do efetuador e em seguida parando o movimento. Foi utilizada uma posição relativa em relação ao alvo $(x_{e^*t})_e = [400 \;\; 0 \;\; 0 \;\; 0]^T$.

Conforme esperado, quando o alvo está parado, ou seja, $\dot{x}_t = 0$, $x_\Delta(t) \to 0$ quando $t \to \infty$ como mostra a Figura 6.22b.

(a) Operador posicionando o alvo a frente do efetuador.



(b) x_Δ

Figura 6.22: Resultados experimentais - Controle por Servovisão

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

Neste projeto de graduação foi apresentada a modelagem cinemática, estratégias de controle baseadas na abordagem de controle cinemático, assim como a implementação de um *software* para controle de sistemas robóticos, utilizando como estudo de caso o manipulador TETIS, do sistema DORIS.

A modelagem cinemática consiste em representar o manipulador como uma cadeia cinemática de corpos rígidos, de modo a obter relações geométricas que governam o sistema. A cinemática direta do manipulador TETIS foi obtida através da convenção Denavit–Hartenberg. Diferenciando as equações resultantes com respeito as variáveis das juntas foi possível obter a cinemática diferencial, relacionando a velocidade no espaço operacional com as variáveis de juntas, na forma do Jacobiano analítico. Em simulação os resultados se mostraram compatíveis com o esperado.

A abordagem de controle cinemático assume que é possível considerar um sistema cinemático como entrada do sistema, tipicamente um valor de velocidade. Isso é possível quando existe uma malha de controle de baixo nível, que idealmente é capaz de impor uma velocidade especificada de referência. Esse é o caso do TETIS, que utiliza atuadores *FHA Mini Servo* com drivers EPOS2 70/10, com uma malha de baixo nível de velocidade. Assim, utilizou-se como modelo simplificado do robô um integrador, supondo que os servos são capazes de reproduzir comandos de velocidade de forma razoavelmente precisa. No caso da Junta 2, essa reprodução não é tão precisa devido necessidade de maior torque, por estar no início da cadeia.

O rastreamento de trajetória com controle proporcional com *feedforward* apresentou erro em regime permanente pelos seguintes fatores: (i) ciclo de controle máximo que o computador embarcado suporta é de $10ms$; (ii) a Junta 2 não é capaz de reproduzir com tanta precisão comandos de velocidade, por estar mais sujeita à torques maiores; (iii) O laço de controle está sujeito a atrasos devido ao *software* e

hardware não respeitarem restrições de tempo-real.

Os componentes de *software* para controle, interação e configuração de parâmetros foram desenvolvidos e integrados ao RobotGUI, que já era utilizado para controle dos demais sistemas do robô. A interface gráfica permite a visualização de dados de forma gráfica *online* dos sinais, assim como a reconfiguração de parâmetros no computador embarcado com aquelas desejadas. A adição de novas funcionalidades e modos de controle é imediata, devido a arquitetura modular e ao paradigma orientado a objetos. O ROS segue os princípios de um microkernel, onde as funcionalidades são implementadas separadamente em módulos bem definidos que se comunicam, em contraste com um "monólito"que realiza todas as funções. Isso permite que nós possam ser substituídos ou modificados sem influenciar outros, desde que sigam o mesmo protocolo (tópicos/serviços). Por exemplo, caso seja de interesse substituir o módulo de visão computacional basta publicar/subscrever aos mesmos tópicos.

A implementação de *software* para controle com computação em tempo real é custosa em tempo de implementação e pouco expansível, necessitando tratar de algoritmos de *scheduling* e latência de entrada/saída (Nilsson et al. 1998). A abordagem utilizada neste trabalho permite uma implementação mais desacoplada e menos interdependente. No entanto, existem algumas desvantagens do uso do ROS para um *software* completamente desacoplado. Existem *overheads* de comunicação entre *ROS Nodes*, que pode se tornar crítico no caso de separar o nó de controle do nó de atuação. Essa separação introduziria atrasos significativos, tornando-se evidente para o caso em que o objetivo de controle é rastrear uma trajetória que é função do tempo. Em casos como nó de visão computacional que realiza o processamento de imagem a separação é inevitável e não representa um problema. Conclui-se ser mais apropriado juntar os nós de atuação e controle, o que melhorou significativamente o erro de rastreamento.

Com o uso do Julia Language foi possível alterar em tempo de execução a trajetória a ser rastreada escrevendo as equações em uma janela na interface, com uma sintaxe simples. O *script* é compilado *just-in-time* e pode ser executado sem perda de performance.

7.2 Trabalhos futuros

- Computar o modelo dinâmico do manipulador TETIS.
- Utilizar ambos os sensores da câmera estereoscópica Minoru de modo a melhorar a estimação da posição e orientação de um alvo, assim como ampliar o campo de visão.

- Extender o uso do Julia Language, de modo a tornar o controle ainda mais dinâmico. Atualmente é utilizado somente na definição de uma trajetória a ser rastreada, no entanto é possível implementar algoritmos de controle inteiramente no Julia, o que permitiria ajustes e experimentos em tempo de execução.

- Estudar implementações para sistemas de controle em tempo real e como tratar atrasos de entrada/saída.

- Implementar a estratégia de controle híbrido posição-força, de forma a permitir que o manipulador TETIS execute tarefas de interação com *touchscreens*.

- Integrar o *Sensable Phantom Omni*, um dispositivo háptico, em modo Master/Slave para permitir que um operador controle manualmente o manipulador com *feedback* da força aplicada em uma superfície.

Referências Bibliográficas

Ackerman, E. & Guizzo, E. (2015), 'Sawyer: Rethink robotics unveils new robot'. Disponível em: <http://spectrum.ieee.org/automaton/robotics/industrial-robots/sawyer-rethink-robotics-new-robot>

Bengel, M. & Pfeiffer, K. (2007), 'Mimroex mobile maintenance and inspection robot for process plants', *Fraunhofer Institute for Manufacturing Engineering and Automation IPA* pp. 1–2.

Camera Calibration and 3D Reconstruction (2016). Disponível em: <http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>

Corke, P. (2011), *Robotics, Vision & Control*, Springer.

Dementhon, D. F. & Davis, L. S. (1995), 'Model-based object pose in 25 lines of code', *International journal of computer vision* **15**(1-2), 123–141.

Freitas, R. S., Xaud, M. F., Marcovistz, I., Neves, A. F., Faria, R. O., Carvalho, G. P., Hsu, L., Nunes, E. V., Peixoto, A. J., Lizarralde, F. et al. (2015), 'The embedded electronics and software of doris offshore robot', *IFAC-PapersOnLine* **48**(6), 208–213.

Galassi, M., Røyrøy, A., de Carvalho, G. P., Freitas, G. M., From, P. J., Costa, R. R., Lizarralde, F., Hsu, L., de Carvalho, G. H., de Oliveira, J. F. et al. (2014), Doris-a mobile robot for inspection and monitoring of offshore facilities, *in* 'Anais do XX Congresso Brasileiro de Automtica (CBA)', pp. 3174–3181.

GmbH, D. (2011), 'Artis - autonomous railguided tank inspection system.'. Acessado em Dezembro de 2016. Disponível em: <http://robotik.dfki-bremen.de/en/research/robot-systems/artis-1.html>

Lattner, C. & Adve, V. (2004), Llvm: A compilation framework for lifelong program analysis & transformation, *in* 'Proceedings of the international symposium

on Code generation and optimization: feedback-directed and runtime optimization', IEEE Computer Society, p. 75.

Marchand, E., Uchiyama, H. & Spindler, F. (2016), 'Pose estimation for augmented reality: a hands-on survey', *IEEE transactions on visualization and computer graphics* **22**(12), 2633–2651.

Nilsson, J. et al. (1998), 'Real-time control systems with delays'.

Nunes, E., Peixoto, A., Xaud, M., From, P. R., Carvalho, G. H., Oliveira, J. F., Lizarralde, F., Neves, A., Motta-Ribeiro, G., Royroy, A. R. et al. (2013), Doris-monitoring robot for offshore facilities, *in* 'OTC Brasil', Offshore Technology Conference.

Oberkampf, D., DeMenthon, D. F. & Davis, L. S. (1996), 'Iterative pose estimation using coplanar feature points', *Computer Vision and Image Understanding* **63**(3), 495–511.

OMD-20-FE-200N DATASHEET V2.1 (2016). Disponível em: <`http://optoforce.com/wp-content/uploads/2016/12/OMD-20-FE-200N-DATASHEET-V2.1.pdf`>

Qt Documentation (2017). Disponível em: <`http://doc.qt.io/`>

Quan, L. & Lan, Z. (1999), 'Linear n-point camera pose determination', *IEEE Transactions on pattern analysis and machine intelligence* **21**(8), 774–780.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. & Ng, A. Y. (2009), Ros: an open-source robot operating system, *in* 'ICRA workshop on open source software', Vol. 3, Kobe, p. 5.

ROS/Concepts - ROS Wiki (2016). Disponível em: <`http://wiki.ros.org/ROS/Concepts`>

Siciliano, B., Sciavicco, L. & Villani, L. (2009), *Robotics : modelling, planning and control*, Advanced Textbooks in Control and Signal Processing, Springer, London.

Skourup, C. & Pretlove, J. (2009), 'The robotized field operator', *ABB Review* **1**, 68–73.

Spindler, F. (2014), 'Calibration of an external camera using the visp library'. Disponível em: <`http://wiki.ros.org/visp_camera_calibration/Tutorials/CalibrationExternalCamera`>

Strang, G. (2005), *Linear Algebra and Its Applications*, 4th edn, Brooks Cole.

The LLVM Compiler Infrastructure (n.d.). Disponível em: <`http://llvm.org/`>

Tomasi, C. & Kanade, T. (1991), 'Detection and tracking of point features'.

Wilfinger, L., Wen, J. & Murphy, S. H. (1994), 'Integral force control with robustness enhancement', *IEEE Control Systems* **14**(1), 31–40.

Xaud, M. (2016), Modeling, control and electromechanical design of a modular lightweight manipulator for interaction and inspection tasks, Master's thesis, Universidade Federal do Rio de Janeiro.

Apêndice A

Coordenadas Homogêneas

Um ponto n-dimensional no espaço Euclidiano $p \in \mathbb{R}^n$ é representado por suas coordenadas como $p = [p_1 \;\; p_2 \;\; \cdots \;\; p_n]$. O ponto é representado em coordenadas homogêneas, ou seja, no espaço projetivo como $\tilde{x} \in \mathbb{P}^n$ com coordenadas $\tilde{x} = [\tilde{p}_1 \;\; \tilde{p}_2 \cdots \tilde{p}_{n+1}]$. As coordenadas Euclidianas estão relacionadas com as coordenadas homogêneas por

$$p_i = \frac{\tilde{p}_i}{\tilde{p}_{n+1}} \qquad i = 1 \cdots n \tag{A.1}$$

Um vetor em coordenadas homogêneas pode ser obtido a partir de suas coordenadas Euclidianas por

$$\tilde{x} = [x_1 \;\; x_2 \;\; \cdots \;\; x_n \;\; 1] \tag{A.2}$$

O fato de as coordenadas homogêneas (ou projetivas) possuírem uma dimensão a mais, oferece algumas vantagens. Permite que pontos e retas no infinito sejam representados utilizando apenas números reais. Também elimina a relevância da escala, pois $\tilde{p}_1$ e $\tilde{p}_2 = \alpha\tilde{p}_1$ representam o mesmo ponto em coordenadas Euclidianas para qualquer $\alpha \neq 0$. Pontos em coordenadas homogêneas também podem ser multiplicados por uma matriz de transformação linear de dimensão $(n+1) \times (n+1)$ de forma a sofrer rotação e translação.

Apêndice B

Estimação *online* da *pose*

B.1 ViSP - Visual Servoing Platform

Para a implementação do modo de controle por servovisão, foi utilizada a biblioteca ViSP. A ViSP contém um módulo de visão computacional que permite computar a *pose* de um objeto a ser reconhecido por meio de um padrão pre-determindado. São utilizados algoritmos robustos e fornecidos mecanismos para calibração da câmera. Esse módulo em alguns casos utiliza a biblioteca OpenCV, com aplicação em problemas de robótica.

B.2 Rastreamento Baseada em Modelo

O ViSP propoe um rastreador baseado em modelo 3D que permite simultaneamente rastrear um objeto utilizando conhecimento de seu modelo CAD e fornecer a sua posição e orientação, no sistema de coordenadas da câmera.

O rastreamento é dividido em duas etapas. A primeira consiste em rastrear as características do objeto. A segunda etapa consiste em determinar a *pose* da câmera para realizar a correspondência das características rastreadas com a projeção do modelo 3D.

Dependendo das características visuais, três rastreadores estão disponíveis.

- Rastreador que considera bordas móveis. Esse rastreador é apropriado para objetos não texturizados.
- Rastreador pelo algoritmo Kanade–Lucas–Tomasi (Tomasi & Kanade 1991), conhecido como KLT. Esse modo é projetado para objetos texturizados cujas boradas não são bem visíveis.
- Versão híbrida que considera bordas móveis e pontos chave KLT.

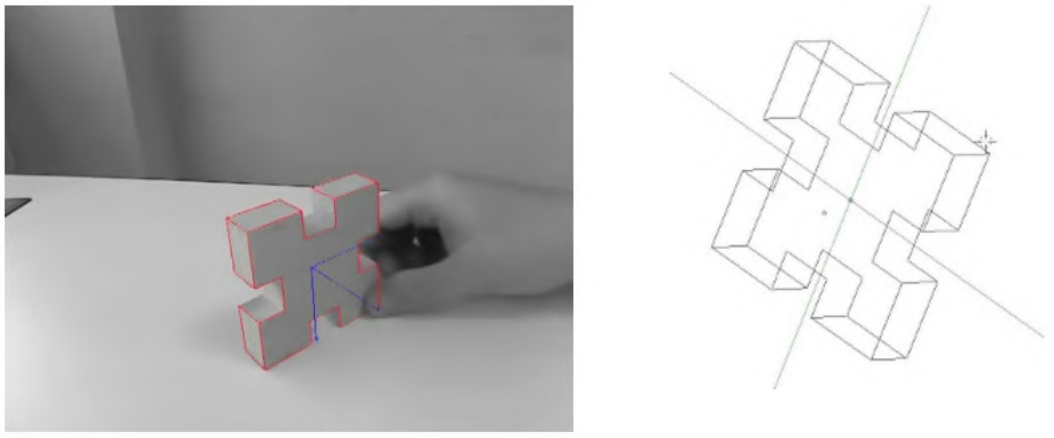


Figura B.1: Rastreamento baseado em modelo

B.3 Perspective-n-Point

O problema Perspective-n-Point é essencial para problemas de servovisão pois permite a obtenção da pose de um objeto através de um conjunti de pontos característicos. As seguintes abordagens ao problema PnP estão disponíveis na biblioteca ViSP são:

- Abordagem Linear Lagrange. É realizado um teste para verificar aplicação da versão planar ou não-planar do algoritmo.
- Abordagem Linear Dementhon (Dementhon & Davis 1995, Oberkampf et al. 1996). É realizado um teste para verificar aplicação da versão planar ou não-planar do algoritmo.
- Abordagem Lowe baseada em um esquema de minimização não linear de Levenberg Marquartd. Necessita de inicialização por meio de Lagrange ou Dementhon.
- Abordagem de Lowe: Baseia-se em uma minimização não linear, inicializada pela aboradagem de Lagrange.

B.4 Pacote ROS visp_auto_tracker

O pacote de ROS chamado `visp_auto_tracker` consiste em um rastreador de objetos baseado em padrões. Ele funciona como um envoltório de ROS para rastreadores baseados em modelo fornecidos pela biblioteca de servovisão ViSP. O objeto a ser rastreado deve possuir um *QRCode* ou um padrão *Flash*. Baseado nesse padrão, o objeto é detectado automaticamente, permitindo a inicialização de rastreadores baseados em modelo. Quando o rastreamento se perde, uma nova detecção é feita e os rastreador é re-inicializado.

O algoritmo utilizado para a solução do problema Perspective-n-Point é a abordagem descrita em (Dementhon & Davis 1995, Oberkampf et al. 1996), que permite computar a posição e orientação rápido o suficiente para permitir rastreio *online* utilizando uma câmera.

Figura B.2: Rastreamento de um QRCode

Apêndice C

Código

Neste Apêndice são mostrados alguns trechos chave do código utilizado na implementação do controle no software.

Código C.1: Controle cinemático com lei proporcional com feedforward.

```
void doris_manipulator_controller::DorisManipulatorController::
     PplusFFMat()
{
    double tnow = GetTimeMicro();
    double t = (tnow - start)/1000000.0;
    Eigen::Vector4d xd, xdd, error, ubar, u;

    double* p = JuliaFunctionWrapper(t);
    xd = Eigen::Map<Eigen::Vector4d>(p);
    xdd =  Eigen::Map<Eigen::Vector4d>(p+4);

    Eigen::Matrix4d Ji = Ja_b.inverse();

    error = xd - x;
    ubar = kt*error + xdd;
    u = Ji*ubar;

    for (int i = 0; i < outputs.size(); i++)
        outputs[i] = u(i);
    SaturateOutput();
}
```

Código C.2: Controle por servovisão.

```
void doris_manipulator_controller::DorisManipulatorController::
     VisionControl()
{
    if (trackingStatus == TRACKER_STATE_TRACK_MODEL)
    {
        Eigen::Vector4d x_delta, x_et, u, ubar;
        Eigen::Vector3d p_ct(x_target(0), x_target(1), x_target(2));

        Eigen::Matrix3d Rec;
        Rec << 0,  0, 1,
               0, -1, 0,
               1,  0, 0;

        Eigen::Vector3d p_et = Rec * p_ct;
        x_et  << p_et(1), p_et(2), p_et(3), x_target(3);
        x_delta = x_et - x_elt;

        ubar = kv * x_delta;
        Eigen::Matrix4d Ji = Ja_e.inverse();
        u = Ji*ubar;

        for (int i = 0; i < outputs.size(); i++)
            outputs[i] = u(i);
        SaturateOutput();
    }
    else
    {
        for (int i = 0; i < outputs.size(); i++)
            outputs[i] = 0;
    }
}
```

Código C.3: Controle de Força.

```
void doris_manipulator_controller::DorisManipulatorController::
     ForceControl()
{
    Eigen::Vector4d ubar, u;
    Eigen::Vector3d error, Fd, ubar_f;
    Fd = Eigen::Vector3d(inputs[0],inputs[1],inputs[2]);

    Eigen::Matrix3d Res, Sf;
    Res << 0,  0, 1,
           0, -1, 0,
           1,  0, 0;

    Sf <<  1, 0, 0,
           0, 0, 0,
           0, 0, 0;

    Eigen::Vector3d Fflt_e = Res*Fflt_s;
    error = Fflt_e - Fd;
    ubar_f = (pid->calculate(error))/KS;
    ubar << Sf * ubar_f, 0;
    Eigen::Matrix4d Ji = Ja_e.inverse();
    u = Ji*ubar;

    for (int i = 0; i < outputs.size(); i++)
        outputs[i] = u(i);
    SaturateOutput();
}
```