Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Electrotécnica e de Computadores

Compensação de movimentos fisiológicos para cirurgia robótica

João Tiago da Silva Marques Pinto

Coimbra, 2011

Compensação de movimentos fisiológicos para cirurgia robótica

Orientador

Doutor Rui Pedro Duarte Cortesão

Júri

Doutor Luís Alberto da Silva Cruz
Doutor Rui Pedro Duarte Cortesão
Doutor Paulo Jorge Carvalho Menezes

João Tiago da Silva Marques Pinto

Dissertação submetida para obtenção do Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Departamento de Engenharia Electrotécnica e de Computadores

Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Setembro 2011

Para a minha família
e para a Stephanie

Agradecimentos

Agradeço ao Professor Doutor Rui Cortesão pela sua disponibilidade e atenção, pela possibilidade que me deu de realizar a minha dissertação de mestrado na área de robótica médica. Agradeço também a motivação e o entusiasmo que me transmitiu na realização do meu trabalho.

Um obrigado aos meus colegas de laboratório por tudo o que me ensinaram, pela paciência, pelo companheirismo e pela amizade.

Aos meus amigos e colegas de curso com os quais partilhei grande parte do meu percurso académico. Um obrigado pela amizade e companheirismo.

Um muito obrigado à minha família pelo apoio e incentivo constantes. Agradeço em especial aos meus pais pelo amor, educação, carinho e por estarem sempre lá quando precisei.

Um especial obrigado à Stephanie por teres estado sempre ao meu lado, principalmente nos momentos em que mais precisei de apoio. Por teres sempre acreditado em mim, por todo o amor, carinho e compreensão que demonstraste. Muito obrigado!

Este trabalho foi suportado em parte pelo projecto da Fundação para a Ciência e Tecnologia (FCT) com referência PTDC/EEA-CRO/110008/2009.

Resumo

Neste documento é apresentado o desenvolvimento de um simulador do manipulador robótico 7-DOF WAM. Neste simulador é feita também a detecção de contactos entre ferramentas acopladas ao *end-effector* do robô e objectos virtuais com os quais o mesmo interage. São também calculadas as forças de contacto resultantes dessa interacção. Para a simulação das forças de contacto é utilizado um método que recorre à voxelização, executada na GPU, dos modelos gráficos dos objectos. A determinação do plano de incidência de uma ferramenta num objecto é feito através do método dos mínimos quadrados totais para planos.

Para efectuar a compensação de movimentos aplicados aos objectos virtuais, é apresentada uma arquitectura de controlo preditivo baseada num modelo linear do manipulador. A linearização do modelo dinâmico do robô é feita através de *feedback* não linear. Foram realizados dois testes com o controlador implementado. No primeiro utilizou-se um plano virtual com um movimento de translação oscilatório. No segundo teste foi utilizada uma esfera virtual com movimento de expansão/contracção, também oscilatório, e o *feedback* de força foi obtido pelo método de simulação de forças de contacto proposto neste trabalho.

É também apresentado o desenvolvimento de uma arquitectura de *software/hardware* que permite controlar remotamente a ferramenta médica cujo modelo virtual foi utilizado no simulador.

Palavras-chave: simulador, forças de contacto, voxelização, GPU, compensação de movimentos, controlo preditivo.

Abstract

This document presents the development of a 7-DOF WAM robot simulator. This simulator also preforms the contact detection between tools coupled to the robot *end-effector* and virtual objects with which it interacts. It's also performed the calculation of the contact forces which result from this interaction. A GPU computed voxelization of the objects graphical models is used for the simulation of contact forces. The method of total least squares is used to determine the plane of incidence.

To perform motion compensation on the virtual objects, a linear predictive control architecture based on a linear model of the manipulator is presented. The linearization of the dynamic model of the robot is done through nonlinear feedback. Two tests were performed with the implemented controller. The first one uses a virtual plane with an oscillating translation. In the second test it was used a virtual sphere with an expansion/contraction oscillating movement, and the force feedback was obtained through the simulation of contact forces presented in this work.

It's also presented the development of a software/hardware architecture which allows the remote control of the medical tool whose virtual model was used in the simulator.

Keywords: simulator, contact forces, voxelization, GPU, motion compensation, predictive control.

Conteúdo

1 Introdução **1**
- 1.1 Estado da Arte . . . 2
- 1.2 Objectivos da Dissertação e Trabalho Desenvolvido . . . 3
- 1.3 Estrutura da Dissertação . . . 4

2 Ferramenta Médica Motorizada **5**
- 2.1 Plataforma Arduino/Ardumoto . . . 5
- 2.2 Arquitectura de software/hardware . . . 6
  - 2.2.1 Módulo de Interface . . . 7
  - 2.2.2 Módulo de Comunicação . . . 8
  - 2.2.3 Módulo de Actuação . . . 8

3 Simulação do manipulador robótico **11**
- 3.1 Modelo do manipulador . . . 11
  - 3.1.1 Modelos geométrico, cinemático e dinâmico . . . 12
  - 3.1.2 Linearização do modelo dinâmico . . . 13
- 3.2 Simulação do manipulador . . . 15
- 3.3 Implementação prática da simulação do manipulador . . . 17
  - 3.3.1 Manipulador 7-DOF WAM . . . 17
  - 3.3.2 Arquitectura do *software* desenvolvido . . . 17

4 Simulação de forças de contacto **23**
- 4.1 Voxelização de modelos *3D* . . . 25
  - 4.1.1 Voxelização implementada na GPU . . . 26
- 4.2 Algoritmo para determinação de forças de contacto . . . 28
- 4.3 Implementação prática da simulação de contactos . . . 30

5 Controlo para compensação de movimentos fisiológicos **34**
- 5.1 Controlo preditivo linear baseado no modelo . . . 34
  - 5.1.1 Formulação . . . 34
  - 5.1.2 Implementação no manipulador WAM . . . 38

6 Resultados experimentais **40**
- 6.1 Voxelização . . . 40
- 6.2 Controlo preditivo linear . . . 42

6.2.1 Compensação de perturbações em força num plano oblíquo virtual 42
6.2.2 Compensação de perturbações em força geradas com a simulação de contactos 43

7 Conclusão **49**

Bibliografia **51**

Lista de Figuras

2.1 Placas utilizadas para construção da ferramenta médica motorizada. . 6
2.2 Arquitectura de *software/hardware* usada juntamente com a ferramenta médica motorizada. 7
2.3 Esquema do ciclo principal do programa que é executado no dispositivo AA. 10

3.1 Linearização por *feedback* não linear de um manipulador. 15
3.2 Manipulador 7-DOF *WAM*. Acoplado ao *end-effector* está a ferramenta médica abordada no capítulo 2. 18
3.3 Arquitectura de *software* implementada. 19
3.4 Funcionamento interno da livraria *3ddisplay*. 21
3.5 Interface gráfico do visualiador. 22

4.1 Exemplo de uma hierarquia usando rectângulos como volumes limitadores. 24
4.2 Modelos gráfico (a trás) e voxelizado (à frente) de um objecto. 25
4.3 Esquema de um *slicemap*. 26
4.4 Limitações da voxelização superficial. A primitiva da esquerda encontra-se alinhada com uma coluna e por isso passa completamente despercebida à voxelização. Na primitiva da direita, como o seu alinhamento é muito próximo do do eixo z, a voxelização não é contígua. 27
4.5 Esquema do método usado para o cálculo das forças de contacto. . . . 29
4.6 Simulação de forças para um *buffer* de dados simples e duplo. 31
4.7 Esquema simplificado de um *pipeline* gráfico. 32
4.8 Sequência de codificação/descodificação de um *slicemap* (a relação das durações entre tarefas é meramente ilustrativa, não correspondendo às durações reais). 33

5.1 Exemplo ilustrativo da evolução temporal de algumas variáveis envolvidas no controlo preditivo. 35
5.2 Planta do sistema com comando de força $u(t)$ e saída em força $y(t)$. . 38
5.3 Arquitectura de controlo para compensação de movimentos. F_r e F_{pert} correspondem,respectivamente, à força de referência e à perturbação de força. $\hat{x}_k$ é o vector de estados, estimados pelo observador a partir da leitura da força medida $y(t)$. 39

6.1 Voxelização superficial de uma esfera centrada na origiem do referencial (as linhas vermelha, verde e azul representam os eixos x, y e z, respectivamente). 40
6.2 Voxelização superficial de um modelo 3D de um coração (modelo 3D à esquerda e respectiva voxelização à direita). 41
6.3 Resultados da acção do controlo preditivo sobre um plano virtual oblíquo para uma perturbação de força sinusoidal. A perturbação é iniciada em $t = 5$ s. 44
6.4 Perturbações em força usadas para testar o controlador preditivo implementado. 45
6.5 Resultados da acção do controlo preditivo sobre um plano virtual oblíquo para uma perturbação de força composta por três sinusóides. A perturbação é iniciada em $t = 5$ s. 46
6.6 Medição de força no ponto correspondente à extremidade da ferramenta acoplada ao manipulador. A medição de referência foi obtida com o plano virtual e as restantes com o método de geração de forças, tendo os *voxel*s um tamanho de 2 mm e 0,6 mm, respectivamente. . . 47
6.7 Compensação de uma perturbação sinusoidal, aplicada segundo o eixo z. A perturbação é iniciada em $t = 5$ s. 47
6.8 Compensação de uma perturbação composta por três sinusóides, aplicada segundo o eixo z. A perturbação é iniciada em $t = 5$ s. 48

Lista de Tabelas

2.1 Lista de funções implementadas necessárias à comunicação com a unidade AA. 8

6.1 Tempos de execução do algoritmo de voxelização. (DV) dimensão de um *voxel*. (RGV) resolução da grelha de *voxel*s. (CS) codificação do *slicemap*. (LMG) leitura da memória gráfica. (DS) Descodificação do *slicemap*. 41

Glossário

EEPROM
tipo de memória não volátil que é usada em computadores e outros dispositivos electrónicos para guardar pequenas quantidades de dados, mesmo se os dispositivos forem desligados. 9

OBJ
Formato de ficheiro desenvolvido pela empresa Wavefront Technologies usado para definir a geometria de objectos tridimensionais. É um formato aberto que tem sido adoptado por várias aplicação gráficas *3D*, sendo hoje em dia um formato universalmente aceite. 20, 40

POSIX
Família de normas, que tem como objetivo garantir a portabilidade do código-fonte entre sistemas operativos que implementem as normas POSIX. 19

degrees of freedom
ver (graus de liberdade). viii, 17

duty cycle
fracção de tempo em que o sistema se encontra no estado activo. No contexto da modelação por PWM, corresponde à fracção de tempo em que o sinal se encontra com o valor lógico '1' relativamente ao período da onda quadrada que representa esse sinal. 8

end-effector
Extremidade de um manipulador robótico que interage com o ambiente de forma directa ou através de objectos a ele acoplados. iii, C, 3, 11, 12, 14, 18, 21, 23, 25, 28, 42, 50

frustum
região do espaço tridimensional que é tida em conta para gerar a imagem que aparece no ecrã. 26, 27

on-board
directamente na placa de circuito impresso. 6

pulse-width modulation
técnica de modulação que gera um sinal composto por pulsos de diferentes comprimentos para representar a amplitude de um sinal analógico. viii

thread
Fio de execução de um programa de computador. Podem existir múltiplos fios de execução a correr em paralelo no mesmo programa. 18–20, 33

voxel
Elemento de volume (cubo) que representa um valor numa grelha uniforme tridimensional. Este elemento é caracterizado pelas coordenadas do centro. Em termos de computação gráfica pode-se considerar que um *voxel* corresponde a um *pixel* em três dimensões. iv, v, 24, 25, 27, 28, 32, 40, 41, 44, 45, 47, 49

cirurgia minimamente invasiva
cirurgia médica através de pequenos orifícios no corpo do paciente. viii, 1

dispositivo háptico
dispositivo de interacção homem-máquina por meio de sensações de tacto, por exemplo, por *feedback* de força. 4, 5

força contra electromotriz
força electromotriz que se opõe à corrente principal que percorre um circuito. viii, 6

força electromotriz
força eléctrica produzida pela conversão de qualquer forma de energia em energia elétrica, que gera uma corrente elétrica. vii

fragmento
em computação gráfica um fragmento corresponde à informação necessária para gerar um pixel numa imagem. 26

GLSL
Linguagem de programação de alto nível para *shaders* baseada em C, que foi criada para dar aos programadores um controlo mais directo sobre o *pipeline* gráfico sem ser necessário usar linguagens específicas de hardware. 31

graus de liberdade
Número mínimo de variáveis que descrevem completamente a configuração, postura, posição, orientação, etc. de um sistema. Num robô está associado ao número de juntas. O espaço cartesiano tem 6, 3 de posição mais 3 de orientação.. vi

rasterização
processo que converte uma imagem vetorial numa imagem rasterizada, isto é, composta por pixeis. 26

Acrónimos

AA
Arduino/Ardumoto. iii, v, 6–10

cc
de corrente contínua. 5, 6

CMI
cirurgia minimamente invasiva. 1–3

DOF
degrees of freedom. i, iii, C, 3, 17, 18, 49

f.c.e.m
força contra electromotriz. 6

GPGPU
General-Purpose computation on Graphics Processing Units. 50

GPU
unidade de processamento gráfico. i, C, 25, 26, 40, 49, 50

HVL
hierarquias de volumes limitadores. 24

MRT
Multiple Render Targets. 27, 33

PWM
pulse-width modulation. vi

Capítulo 1

Introdução

A robótica médica, embora seja muitas vezes associada apenas à robótica para cirurgia, é na verdade uma área muito vasta, actualmente com presença em praticamente todos os ramos da medicina [1]. No entanto, devido aos grandes desafios envolvidos na utilização de robôs em ambientes de cirurgia, a robótica cirúrgica tem vindo a despertar cada vez mais o interesse da comunidade científica, com o número de artigos publicados na área a crescer de ano para ano.

O conceito de robótica cirúrgica surgiu pela primeira vez nos anos 80 com a modificação de um manipulador robótico (PUMA 560), utilizado na indústria, de modo a que este fosse usado numa biópsia ao cérebro com uma precisão de cerca de 0,05 mm [2]. Devido ao potencial, desde logo observado, da utilização de robôs em procedimentos cirúrgicos, outras aplicações do género foram surgindo e rapidamente a adaptação de robôs industriais deu lugar ao desenvolvimento de manipuladores específicos para robótica cirúrgica. Estes manipuladores não só aumentam as capacidades técnicas do cirurgião ao nível da precisão e da rapidez de intervenção, como permitem a execução de movimentos mais complexos. As vantagens apresentadas tornam a robótica cirúrgica especialmente indicada para a cirurgia minimamente invasiva (CMI).

A CMI é um tipo de cirurgia em que os instrumentos médicos entram no corpo humano através de pequenos orifícios, sendo o feedback visual do médico obtido através de um endoscópio ou de uma câmera. Este tipo de cirurgia torna as intervenções cirúrgicas mais rápidas e menos invasivas, o que resulta num menor tempo de recuperação para o paciente. A utilização de robôs em CMI permite mitigar algumas das suas principais desvantagens, nomeadamente o desconforto e a falta de coordenação entre a visão e as mãos, por parte do médico cirurgião. Com a implementação de arquitecturas de controlo adequadas é ainda possível restringir os movimentos dos manipuladores, o que se traduz num aumento da segurança, e fazer a compensação de movimentos indesejados como os resultantes do tremor da mão do cirurgião.

Embora a utilização de robôs em ambientes cirúrgicos se revele extremamente vantajosa e seja cada vez mais uma realidade, há ainda determinados tipo de cirurgia que constituem autênticos desafios para a robótica médica. As cirurgias ao cora-

ção, por exemplo, requerem a execução de procedimentos cirúrgicos muito precisos e complexos sob a acção de movimentos fisiológicos induzidos pelo batimento cardíaco. Para que estes procedimentos sejam executados correctamente, é necessário que os movimentos fisiológicos referidos sejam compensados. No entanto, mesmo para os cirurgiões mais experientes, é impossível efectuar manualmente esta compensação sem que resultem erros de fase e amplitude [3]. A resolução deste problema pode passar pelo desenvolvimento de métodos que permitam que a compensação dos movimentos fisiológicos seja feita de forma autónoma pelos manipuladores robóticos usados para cirurgia.

1.1 Estado da Arte

A presença de manipuladores robóticos em salas de cirurgia é uma realidade bastante presente nos dias de hoje. Ao longo dos últimos anos vários têm sido os robôs, desenvolvidos especificamente para aplicações médicas, a entrar nos hospitais. Ao nível da CMI o sistema *da Vinci* [4], da Intuitive Surgical, é o mais conhecido e usado actualmente. Este sistema é composto por quatro manipuladores com instrumentos especializados, um dos quais um endoscópio responsável pela obtenção de imagens 3D que proporcionam um *feedback* visual do que se passa durante a cirurgia. Este sistema permite ainda escalar os movimentos do médico de modo a que os mesmos sejam traduzidos em micro-deslocamentos muito precisos dos instrumentos que operam através de pequenos orifícios no corpo do paciente.

Apesar das vantagens que sistemas como o *da Vinci* trazem à CMI, a sua utilização em cirurgias cardíacas ainda é relativamente problemática devido aos movimentos fisiológicos existentes. Para ultrapassar esta situação recorre-se muitas vezes ao uso de estabilizadores mecânicos passivos que actuam numa determinada zona do coração, reduzindo localmente o movimento do mesmo. Apesar dos melhoramentos verificados desde as primeiras versões (no início dos anos 90), estes estabilizadores apresentam ainda algumas desvantagens [5, 6], nomeadamente os movimentos residuais, devidos a uma imobilização insuficiente, que têm de ser compensados pelo cirurgião. Um método alternativo aos estabilizadores consiste na utilização de uma máquina externa que assegura a circulação e filtragem do sangue enquanto o coração está parado. Contudo, a utilização deste tipo de sistemas implica maiores riscos e uma recuperação mais demorada para o paciente [7].

Numa tentativa de ultrapassar os problemas existentes nos métodos clássicos apresentados, têm sido investigadas, nos últimos anos, soluções em que a compensação dos movimentos fisiológicos é feita autonomamente por manipuladores robóticos, através da aquisição de dados sensoriais (p. ex. visão e/ou força). Em [8], é proposto um método que utiliza um sistema de visão de alta velocidade para medir os deslocamentos de determinadas regiões do coração, usadas como marcadores naturais. Estes deslocamentos são depois comunicados a um manipulador robótico que, desta forma, consegue compensar o batimento cardíaco durante a cirurgia. Mais recentemente, Bachta e co-autores [9] propuseram um método de estabilização activa que pretende melhorar os métodos clássicos utilizados. Neste método é usada uma

câmara de alta velocidade para criar uma malha de controlo que permita eliminar os deslocamentos residuais verificados nos estabilizadores mecânicos convencionais. As soluções baseadas apenas em informação visual têm no entanto desvantagens [10]. Em CMI, a área de manuseamento dos instrumentos médicos é geralmente muito reduzida e a oclusão dos marcadores (naturais ou artificiais) ocorre com muita frequência. Além disso, o contacto com os tecidos provoca a sua deformação, o que afecta drasticamente a calibração dos marcadores.

Recentemente têm sido apresentadas soluções que utilizam arquitecturas de controlo baseadas em informações de força. Com este tipo de arquitecturas de controlo não só é possível fazer a compensação de movimentos como se consegue proporcionar ao médico cirurgião algum *feedback* háptico. Em [11] é feita uma análise dos efeitos do *feedback* de força na execução de procedimentos complexos como a sutura-ção. Os resultados experimentais mostraram que a existência de *feedback* de força melhora a prestação do médico em procedimentos que são executados com o auxílio de robôs. Em [12] é proposta uma arquitectura de controlo, baseada em *feedback* de força capaz de compensar movimentos periódicos. Este método usa um módulo de controlo avançado que rejeita perturbações periódicas sem ser, para isso, necessário o conhecimento de nenhum modelo específico do robô ou do meio que o envolve. Contudo, o método referido baseia-se na periodicidade das perturbações que tenta compensar, o que pode ser problemático em cirurgias cardíacas devido à natureza aleatória dos movimentos resultantes do batimento do coração [8].

1.2 Objectivos da Dissertação e Trabalho Desenvolvido

Este trabalho teve como objectivo o desenvolvimento de um simulador que permita reproduzir o comportamento dinâmico real do manipulador 7-DOF WAM, bem como modelar as forças externas resultantes da sua interacção com outros objectos. O simulador desenvolvido tem como principal finalidade o teste e a afinação de algoritmos de controlo baseados em *feedback* de força, nomeadamente os que permitem fazer a compensação dos movimentos fisiológicos existentes em cirurgias cardíacas. Para tal é necessário determinar as forças resultantes do contacto das ferramentas médicas, acopladas ao *end-effector* do manipulador, com objectos deformáveis cuja expansão e/ou contracção sejam semelhantes às de um coração.

Para a simulação das forças de contacto foi desenvolvido um método que tem por base o trabalho apresentado em [13]. A característica principal do método desenvolvido, quando comparado a outros já existentes, é a sua rapidez, fundamental em aplicações que envolvem *feedback* háptico. Esta rapidez deve-se ao facto de grande parte do processamento ser executado pelo GPU, como é descrito no capítulo 4.

No âmbito deste trabalho foi também implementada uma arquitectura de controlo baseada em controlo preditivo linear que permite, através de *feedback* de força, a compensação de movimentos fisiológicos, não sendo para isso necessária qualquer

informação à priori sobre os movimentos em causa. O *feedback* de força é assegurado pelas medidas do esforço de contacto aplicado pelo manipulador robótico num determinado objecto. Uma vez que o controlo implementado necessita de um modelo matemático linear, foi necessário recorrer à linearização por *feedback* não linear, descrita em 3.1.2, do modelo dinâmico do manipulador WAM.

Foi ainda desenvolvida uma arquitectura de *software/hardware* que permite a motorização de ferramentas médicas, actualmente existentes no mercado, de modo a que as mesmas possam ser controladas remotamente através, por exemplo, de um computador ou de um dispositivo háptico.

1.3 Estrutura da Dissertação

A Dissertação apresentada encontra-se organizada da seguinte forma:

- Capítulo 1 - Breve introdução à robótica médica/cirúrgica e ao estado da arte relativo à compensação de movimentos fisiológicos em cirurgia robótica. Descrição dos objectivos propostos para o trabalho desenvolvido.
- Capítulo 2 - Apresentação da estrutura de *software*/*hardware* desenvolvida que permite a utilização de ferramentas médicas motorizadas em ambiente de cirurgia assistida por robôs.
- Capítulo 3 - Introdução da teoria de suporte ao desenvolvimento de um simulador de um manipulador robótico e apresentação da estrutura de *software* usada para o seu desenvolvimento.
- Capítulo 4 - Descrição do método proposto para a modelação das forças de contacto resultantes da interacção de um manipulador com o meio envolvente.
- Capítulo 5 - Descrição do algoritmo de controlo preditivo utilizado para a compensação de movimentos fisiológicos em cirurgia assistida por robôs.
- Capítulo 6 - Análise e discussão dos resultados experimentais obtidos.
- Capítulo 7 - Apresentação das conclusões relativas ao trabalho desenvolvido, bem como de propostas para trabalho futuro.

Capítulo 2

Ferramenta Médica Motorizada

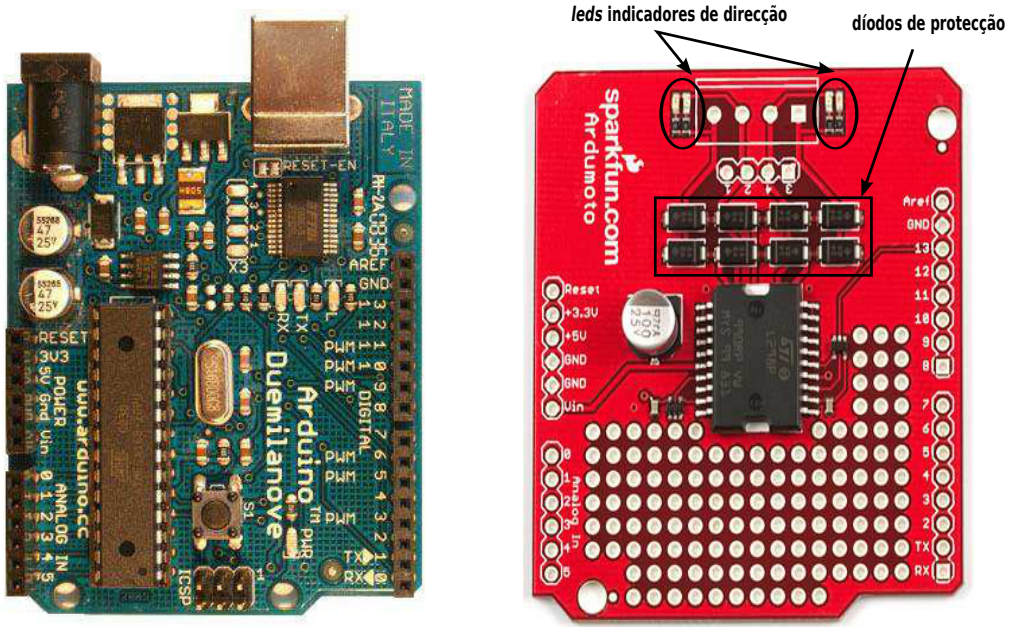
Este capítulo aborda o desenvolvimento de uma estrutura que permite motorizar ferramentas cirúrgicas já existentes no mercado, de forma a que estas possam ser aplicadas em manipuladores robóticos e integradas num ambiente de cirurgia assistida roboticamente. Na secção 2.1 é apresentada a plataforma que serviu de suporte ao módulo desenvolvido para controlar a ferramenta. Na secção 2.2 é feita uma descrição detalhada da arquitectura implementada que permite a utilização desta ferramenta juntamente com os outros dispositivos normalmente utilizados em cirurgia assistida por manipuladores robóticos.

2.1 Plataforma Arduino/Ardumoto

Para efectuar o controlo do motor da ferramenta médica desenvolvida foi utilizada a plataforma *Arduino* que, para além de ser uma plataforma livre, é constituída por um micro-controlador de placa única, muito compacta, cuja linguagem de programação é baseada em C/C++[1]. Apesar das suas dimensões reduzidas esta plataforma possui várias linhas de entrada/saída digitais e algumas analógicas (necessárias ao controlo do motor referido) bem como um módulo de comunicação por *USB* ou *RS-232*, o que permite que o controlo da ferramenta seja efectuado por um dispositivo remoto. Este módulo de comunicação tem especial importância pois permite ao cirurgião controlar a ferramenta médica com o mesmo dispositivo háptico usado para controlar o manipulador robótico no qual a ferramenta é acoplada. Na figura 2.1(a) encontra-se representada a placa *Arduino Duemilanove* utilizada para controlar a ferramenta médica desenvolvida. As especificações desta placa, bem como todas as informações necessárias à sua linguagem de programação podem ser consultadas em [14].

Devido ao facto de as placas *Arduino* não estarem preparadas para efectuar directamente o controlo de motores, houve necessidade de utilizar uma das várias placas de expansão, geralmente designadas de *shields*, disponíveis para esta plataforma. A placa referida é a *Ardumoto Motor Driver Shield* que permite o controlo de dois motores (de corrente contínua) em simultâneo e possibilita ainda a implementação

[1] http://arduino.cc/en/Main/Policy



(a) Placa *Arduino Duemilanove* (b) Placa *Ardumoto Motor Driver Shield*

Figura 2.1: Placas utilizadas para construção da ferramenta médica motorizada.

on-board de um pequeno circuito eléctrico. Esta placa é colocada na parte superior da *Arduino Duemilanove* e, como se pode observar na 2.1(b), a correspondência de pinos entre as duas placas permite que os mesmos possam ser acedidos a partir de qualquer circuito eléctrico *on-board* implementado. Os *leds* presentes na parte superior indicam qual a "direcção" em que cada motor se está a deslocar e os díodos localizados nas linhas de actuação de cada um dos motores protegem as mesmas da força contra electromotriz (f.c.e.m) gerada pelo funcionamento desses mesmos motores.

2.2 Arquitectura de software/hardware

A ferramenta médica desenvolvida neste projecto consiste num instrumento da série *Karl Storz Clickline*[2] cuja parte de actuação manual (semelhante a uma tesoura) lhe foi retirada e a parte da pinça acoplada a uma estrutura que permite o seu encaixe a um motor de corrente contínua (cc) linear *Firgelli PQ12*[3]. De modo a integrar esta ferramenta num ambiente de cirurgia assistida por manipuladores robóticos, foi implementada uma arquitectura de *software/hardware* que inclui uma unidade *Arduino/Ardumoto* (AA), um computador ligado a esta por uma comunicação série (*RS-232*) e um dispositivo háptico opcional. A unidade AA é constituída por três módulos distintos: comunicação, interface e actuação. Os dois primeiros permitem

[2]http://www.karlstorz.de/cps/rde/xchg/SID-27BEF720-88064AC5/karlstorz-en/hs.xsl/8906.htm

[3]http://www.firgelli.com/products.php?id=10

uma utilização remota ou local da ferramenta, respectivamente, enquanto que o terceiro é responsável por actuar directamente no motor da mesma.

O computador encontra-se ligado ao módulo de comunicação e é o dispositivo responsável pelo controlo remoto da ferramenta. Este controlo pode ser efectuado directamente através de comandos enviados pelo computador ou através do dispositivo háptico a ele ligado.

O dispositivo háptico ligado ao computador permite ao médico cirurgião manipular a ferramenta motorizada com o mesmo dispositivo com que controla o manipulador robótico onde a ferramenta é acoplada. Para tal é necessário apenas associar os controlos existentes neste dispositivo com comandos que possam ser enviados pelo computador. Na figura 2.2 é apresentado um esquema da arquitectura descrita. As setas presentes na figura indicam a forma como a informação, relativa ao comando a aplicar ao motor e à sua posição, é transferida entre os vários dispositivos e o utilizador.

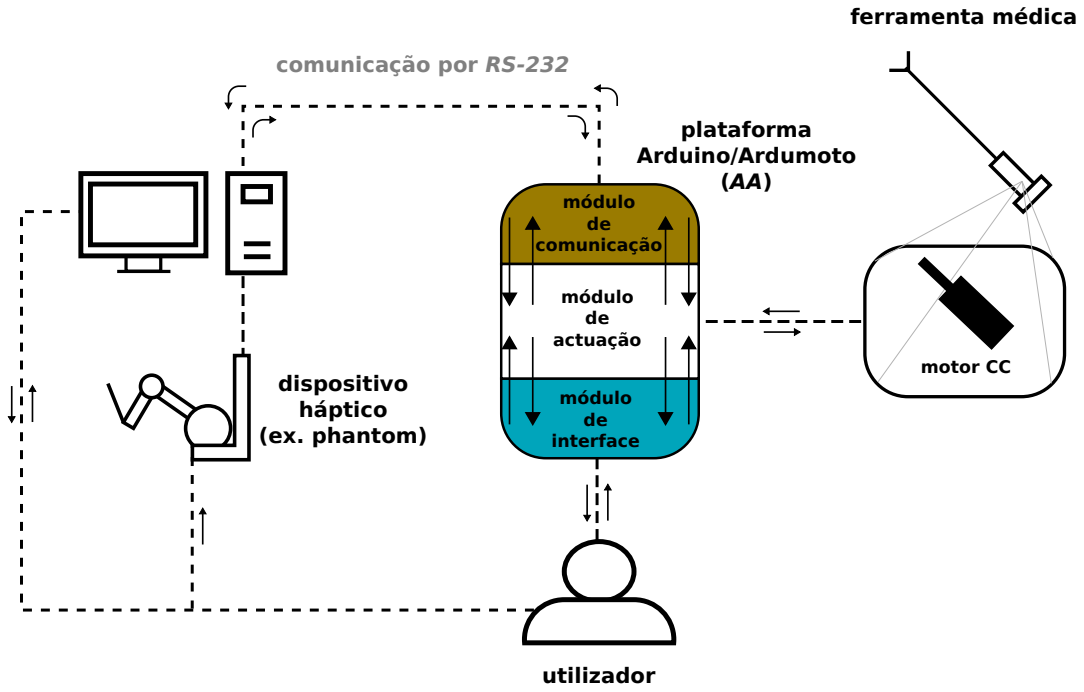


Figura 2.2: Arquitectura de *software/hardware* usada juntamente com a ferramenta médica motorizada.

2.2.1 Módulo de Interface

O módulo de interface consiste num pequeno circuito electrónico que foi desenvolvido na placa *Ardumoto* para permitir ao utilizador controlar o motor linear directamente no dispositivo AA e para possibilitar a aplicação de limites no deslocamento do mesmo. Este deslocamento está directamente relacionado com a abertura da pinça cirúrgica. O circuito electrónico desenvolvido contém dois botões que movimentam

o motor linear nos dois sentidos e um terceiro botão para mudar o modo de funcionamento do módulo de actuação. Para além dos botões estão ainda disponíveis dois *leds* que consoante a frequência com que acendem e apagam, indicam a proximidade do motor relativamente a cada um dos limites. Estes dois *leds* juntamente com um terceiro indicam ainda qual modo de funcionamento em que se encontra o módulo de actuação. O esquemático do circuito desenvolvido encontra-se em anexo.

2.2.2 Módulo de Comunicação

Como foi referido anteriormente, este módulo é responsável por permitir a ligação de um dispositivo remoto à unidade AA através do interface *USB/RS-232* existente na placa *Arduino*. É um módulo de *software* constituído por um uma livraria desenvolvida na linguagem de programação *C*, que é executada a partir do computador remoto, e por um conjunto de rotinas, implementadas na unidade AA, que interpretam os comandos existentes nessa livraria. Na tabela 2.1 são apresentadas as funções da livraria que foram implementadas para permitir o controlo remoto da ferramenta motorizada.

Tabela 2.1: Lista de funções implementadas necessárias à comunicação com a unidade AA.

Comando	**Descrição**
openSerialPort	Estabelece a comunicação série com o módulo de comunicação.
setCommandMotor	Define o valor do *duty cycle* (0-1) a aplicar à saída analógica da placa *Arduino*, ou seja, o valor médio de tensão a aplicar ao motor.
getCommandMotorPosition	Obtém a posição actual do motor bem como o seus actuais limites de abertura.

2.2.3 Módulo de Actuação

Como se pode observar na figura 2.2, é a este módulo que é conectado o motor linear da ferramenta e é através dele que é feita a actuação desse mesmo motor, assim como a leitura da sua posição actual. A actuação do motor é efectuada tendo em conta os comandos recebidos dos outros dois módulos e os limites impostos para a posição do mesmo.

O módulo de actuação inclui o ciclo principal do programa que é executado na plataforma AA e pode-se encontrar, durante a execução desse ciclo, num de três estados:

- ***lock*** - neste estado a actuação do motor é feita tendo em conta os limites impostos para a sua posição e, por isso, a abertura da pinça cirúrgica está compreendida entre um valor máximo e mínimo, definidos pelo utilizador.

- ***unlock*** - neste estado a actuação do motor é feita ignorando os limites impostos para a sua posição e, por isso, a única restrição para a abertura da pinça cirúrgica é ao nível dos seus limites físicos de fabrico.

- ***set*** - neste estado a actuação no motor é desactivada e o utilizador pode definir novos valores para os limites de abertura da pinça cirúrgica.

Sempre que é feita uma alteração dos limites da posição do motor (modo *set*) os seus valores são guardados na memória *EEPROM* da placa *Arduino* de modo a que, se for necessário desligar a unidade AA, esses mesmos limites estejam activos quando a unidade for novamente ligada.

O módulo de actuação é também responsável por, a cada iteração do ciclo principal, executar a parte do módulo de comunicação implementada na unidade AA, assim como por actualizar o estado dos componentes que constituem o módulo de interface. Na figura 2.3 é apresentado um esquema da execução do ciclo principal e, como se pode observar, no caso de existir um sinal de comando proveniente do módulo de comunicação e outro do módulo de interface, em simultâneo, apenas o sinal do segundo é tido em conta. Neste esquema os elementos entre *[]* correspondem às variáveis que são calculadas, quando é necessário, nos módulos correspondentes, e os que estão entre *()* correspondem a variáveis que se encontram armazenadas em memória.

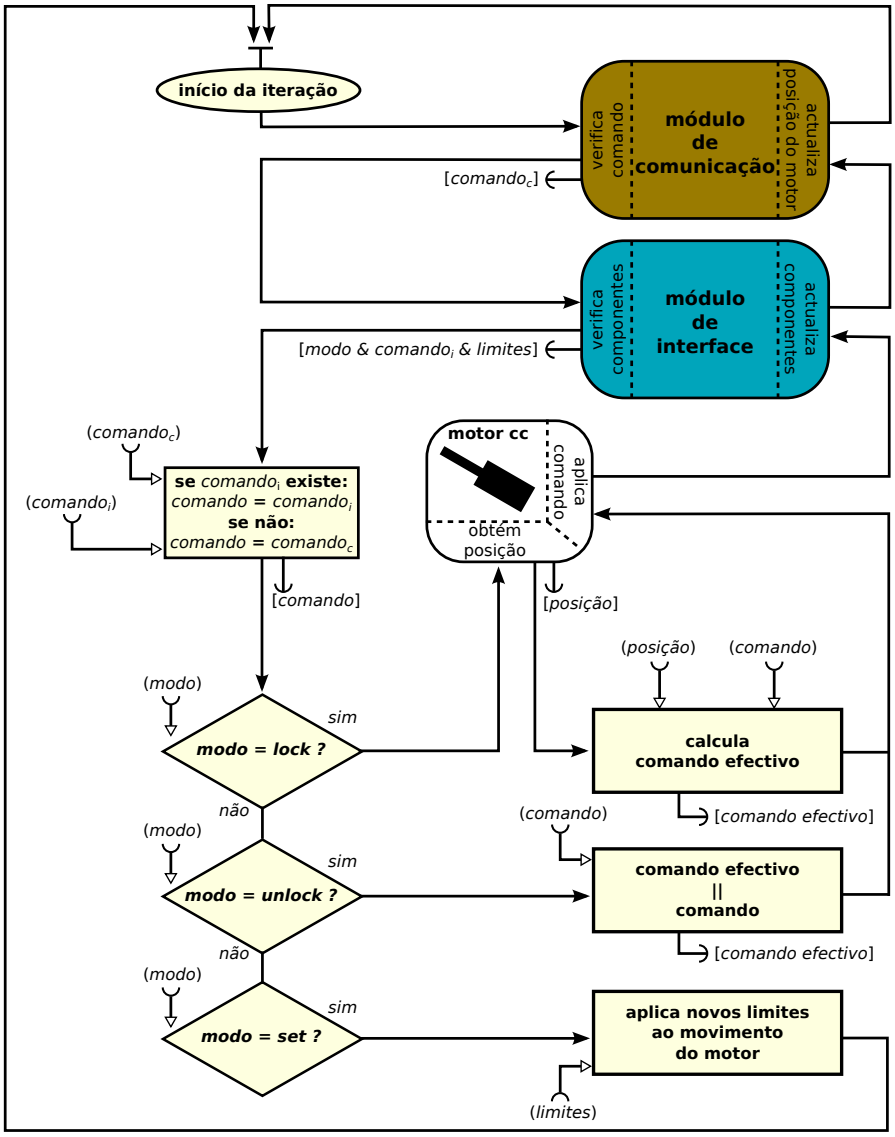


Figura 2.3: Esquema do ciclo principal do programa que é executado no dispositivo AA.

Capítulo 3

Simulação do manipulador robótico

Neste capítulo é apresentada toda a teoria de suporte ao desenvolvimento de um simulador do manipulador sobre o qual incidiu este projecto. Na secção 3.1 é feita uma pequena introdução à notação usada neste capítulo bem como uma descrição dos modelos que caracterizam um manipulador robótico. É ainda apresentado um método de linearização do modelo dinâmico de um manipulador que é implementado no controlo discutido no capítulo 5. Na secção 3.2 é feita uma descrição mais detalhada de como simular a dinâmica de um manipulador robótico e é apresentado o método de integração numérico que foi usado para a simulação da mesma. Na secção 3.3 são apresentados todos os módulos de *software* desenvolvidos para a implementação do simulador referido bem como a arquitectura utilizada para a interligação desses mesmos módulos.

3.1 Modelo do manipulador

A configuração de um manipulador, num dado instante temporal, é descrita por um vector q que contém o valor de posição[1] de cada junta nesse mesmo instante. Os vectores de velocidade e aceleração das juntas são representados por $\dot{q}$ e $\ddot{q}$, respectivamente[2].

Sabendo o modelo geométrico do manipulador é possível obter, a partir de q, o vector X que descreve a posição e a orientação cartesianas do *end-effector* e vice-versa. A velocidade e aceleração do *end-effector* são descritas pelos vectores $\dot{X} = [\dot{p}\ \omega]^T$ e $\ddot{X} = [\ddot{p}\ \dot{\omega}]^T$ respectivamente, sendo p a componente de posição de X e ω a componente de velocidade angular. Estes vectores, assim como o vector X, serão, ao longo do texto, sempre referenciados na base do manipulador.

O vector de binários que actuam nas juntas quer por actuação dos motores quer por acção de forças externas é denominado por τ.

[1] No caso do manipulador considerado, a posição da junta corresponde a uma rotação.

[2] O ponto colocado por cima de q representa a sua derivada, ou seja, $\dot{q} = \frac{dq}{dt}$.

O vector das forças e momentos de força cartesianos associados ao *end-effector* é representado por $F = [f\ \mu]^T$, com f a componente de força e μ a componente de binário. Os vectores F e τ estão relacionados pela transposta da matriz Jacobiana que é calculada a partir do modelo cinemático do manipulador.

Para caracterizar completamente um manipulador é necessário determinar, para além dos modelos geométrico e cinemático já referidos, o seu modelo dinâmico. O modelo dinâmico permite, a partir de q e $\dot{q}$, relacionar a aceleração das juntas $\ddot{q}$ com o binário total τ a elas aplicado.

3.1.1 Modelos geométrico, cinemático e dinâmico

O modelo geométrico de um robô permite relacionar o vector q de posições das juntas com o da posição do *end-effector*. A transformação de q em X,

$$X = \mathrm{dh}(q)\ , \tag{3.1}$$

pode ser calculada usando a convenção de Denavit-Hartenberg [15].

A obtenção do vector X é essencial para modelar qualquer tipo de contactos que o manipulador tenha com objectos existentes no ambiente simulado. Só conhecendo a posição do *end-effector* (ou do objecto a ele acoplado[3]) relativamente aos objectos em causa é que é possível calcular as forças e binários resultantes da interacção entre eles. Para além da modelação de contactos, o vector X tem especial importância quando se pretende implementar arquitecturas de controlo em que o *feedback* de força é determinado não por um sensor mas a partir do erro de posição.

Tanto no caso da simulação de contactos como na implementação do controlo, é necessário converter a força F associada ao *end-effector* num vector de binários τ a aplicar às juntas de forma a que o efeito desta seja sentido na dinâmica do manipulador. Para tal recorre-se ao modelo cinemático já referido anteriormente. Com este modelo é possível calcular o Jacobiano que é usado para calcular o vector de velocidades linear e angular do *end-effector* sabendo a velocidade de cada uma das juntas:

$$\dot{X} = J(q)\dot{q}\ . \tag{3.2}$$

Utilizando o Jacobiano transposto é possível, sabendo F, calcular o vector τ:

$$\tau = J^T(q)F\ . \tag{3.3}$$

Simular a dinâmica de um manipulador robótico consiste em determinar, iterativamente no decorrer da simulação, a posição, velocidade e aceleração de todas as juntas em função dos binários aplicados nas mesmas. Estes binários podem resultar de comandos aplicados intencionalmente ou da acção de outras forças como por exemplo a da gravidade. Para efectuar esta simulação é necessário determinar o modelo dinâmico do manipulador em causa que é descrito pela seguinte equação:

$$M(q)\ddot{q} + c(\dot{q}, q) + g(q) + d(\dot{q}, q) = \tau\ , \tag{3.4}$$

[3]A posição do objecto acoplado é calculada através de uma transformação de corpo rígido conhecida feita à posição p do *end-effector*.

em que $M(q)$ é a matriz das massas, $c(\dot{q}, q)$ é o vector das forças de *Coriolis* e centrípetas, $g(q)$ é o termo da gravidade, e $d(\dot{q}, q)$ é o vector das forças de atrito. A matriz $M(q)$ representa a inércia do corpo do robô, e o termo $g(q)$ representa o efeito da força gravítica sobre o mesmo. Os termos apresentados podem ser calculados computacionalmente a cada iteração através de dois métodos: o método recursivo *Newton-Euler* (RNE) e o método *Euler-Lagrange* (EL) [15], ou uma combinação dos dois. Uma vez que o vector das forças de atrito é geralmente muito difícil de modelar costuma ser ignorado e tratado como uma perturbação.

O conhecimento do modelo dinâmico de um manipulador possibilita ainda o desenvolvimento de arquitecturas de controlo mais robustas que incluam a linearização por *feedback*[4] desse mesmo modelo.

3.1.2 Linearização do modelo dinâmico

Utilizando as equações (3.2) e (3.3) juntamente com as seguintes [16]:

$$\Lambda(q) = (JM^{-1}J^T)^{-1} \tag{3.5}$$
$$C(\dot{q}, q) = (J^+)^T c(\dot{q}, q) - \Lambda \dot{J}\dot{q} \tag{3.6}$$
$$g_t(q) = (J^+)^T g(q) \ , \tag{3.7}$$

é possível rescrever a equação (3.4) da dinâmica no espaço de tarefa[5], ou espaço cartesiano, obtendo-se:

$$\Lambda(q)\ddot{X} + C(\dot{q}, q) + g_t(q) = F \ . \tag{3.8}$$

Como o Jacobiano nem sempre é caracterizado por uma matriz quadrada, ou seja, o número de juntas do manipulador é diferente do número de graus de liberdade do espaço de tarefa (seis), nem sempre é simples a obtenção da sua inversa. Por este motivo, em vez da inversa, é usada a inversa generalizada J^+ correspondente à solução que minimiza a energia cinética instantânea do manipulador [16]. Esta matriz é dada por:

$$J^+ = M^{-1}J^T\Lambda \ . \tag{3.9}$$

Como foi referido em 3.1.1, o binário τ aplicado às juntas do manipulador pode ter origem em comandos intencionais, aplicados por exemplo pelo controlo, ou forças externas ao manipulador. Pela equação (3.3) verifica-se que também F pode ser dividida em duas componentes, isto é, uma componente F_c proveniente de forças de comando e uma outra componente F_e proveniente de forças externas. A equação (3.8) pode então ser rescrita da seguinte forma:

$$\Lambda(q)\ddot{X} + C(\dot{q}, q) + g_t(q) = F_c + F_e \ . \tag{3.10}$$

[4]Entenda-se que sempre que for referida a linearização do modelo dinâmico estar-se-á a fazer referência a uma"linearização por *feedback* não linear".

[5]Desprezando, neste caso, o termo $d(\dot{q}, q)$.

De modo a obter uma linearização e desacoplamento do modelo real do robô, ficando a planta do sistema reduzida a

$$\ddot{X} = F^{\star} \ , \tag{3.11}$$

é necessário que a força de comando F_c seja dada por:

$$F_c = -\hat{F}_e + \hat{C}(\dot{q}, q) + \hat{g}_t(q) + \hat{\Lambda}(q)F^{\star} \ , \tag{3.12}$$

com $\hat{F}_e$, $\hat{C}(\dot{q}, q)$, $\hat{g}_t(q)$ e $\hat{\Lambda}(q)$ as estimativas, calculadas computacionalmente, de F_e, $C(\dot{q}, q)$, $g_t(q)$ e $\Lambda(q)$, respectivamente. O modelo dinâmico obtido, dado pela equação (3.11), é equivalente ao modelo de uma massa unitária em espaço livre, sendo $\ddot{X}$ a aceleração dessa massa e F^* a força e binário totais nela aplicados.

Por vezes pretende-se recorrer apenas à linearização do modelo para controlar a posição do *end-effector* pois a orientação deste é controlada por uma arquitectura de controlo diferente. Nestes casos é usada uma versão truncada J_v que contém apenas as três primeiras linhas (correspondentes à componente linear da velocidade) do Jacobiano. Do mesmo modo que se obteve (3.2) é possível obter uma relação entre as velocidades das juntas e a velocidade linear do *end-effector*, resultando:

$$\dot{p} = J_v(q)\dot{q} \ . \tag{3.13}$$

A equação (3.8) pode ser rescrita numa equação que descreve a dinâmica, no espaço cartesiano, apenas para a posição [17], obtendo-se:

$$\Lambda_p(q)\ddot{p} + C_p(\dot{q}, q) + g_p(q) = f \ , \tag{3.14}$$

sendo $\Lambda_p(q)$, $C_p(\dot{q}, q)$ e $g_p(q)$ calculados da mesma forma que em (3.5), (3.6), (3.7) e (3.9), mas substituindo J por J_v e J^+ por J_v^+. De (3.10) e (3.12) facilmente se verifica que a linearização do modelo dinâmico, para a posição cartesiana, é conseguida fazendo a força de comando f_c igual a:

$$f_c = -\hat{f}_e + \hat{C}_p(\dot{q}, q) + \hat{g}_p(q) + \hat{\Lambda}_p(q)f^{\star} \ , \tag{3.15}$$

ficando o sistema caracterizado pelo modelo

$$\ddot{p} = f^{\star} \ . \tag{3.16}$$

Quando implementado na prática, o método de linearização por feedback não é feito no espaço cartesiano mas sim no espaço das juntas do manipulador. Pela equação (3.3) é possível mapear F_c no espaço das juntas:

$$\tau_c = \hat{C}(\dot{q}, q) + \hat{g}(q) + J^T[\hat{\Lambda}(q)(F^* - \dot{J}\dot{q}) - \hat{F}_e] \ , \tag{3.17}$$

com τ_c o binário de comando a aplicar nas juntas. Estendendo o resultado obtido para o caso da linearização ser feita unicamente ao nível da posição do *end-effector*:

$$\tau_c = \hat{C}_p(\dot{q}, q) + \hat{g}_p(q) + J_v^T[\hat{\Lambda}_p(q)(F^* - \dot{J}_v\dot{q}) - \hat{f}_e] \ . \tag{3.18}$$

Como se pode observar, desta forma não é necessário calcular J^+ nem J_v^+. A figura 3.1 ilustra esquematicamente a implementação prática do método de linearização descrito. Da análise da figura verifica-se que para o controlo, o sistema dinâmico que está a ser controlado fica reduzido a um simples duplo integrador independente para cada dimensão cartesiana.

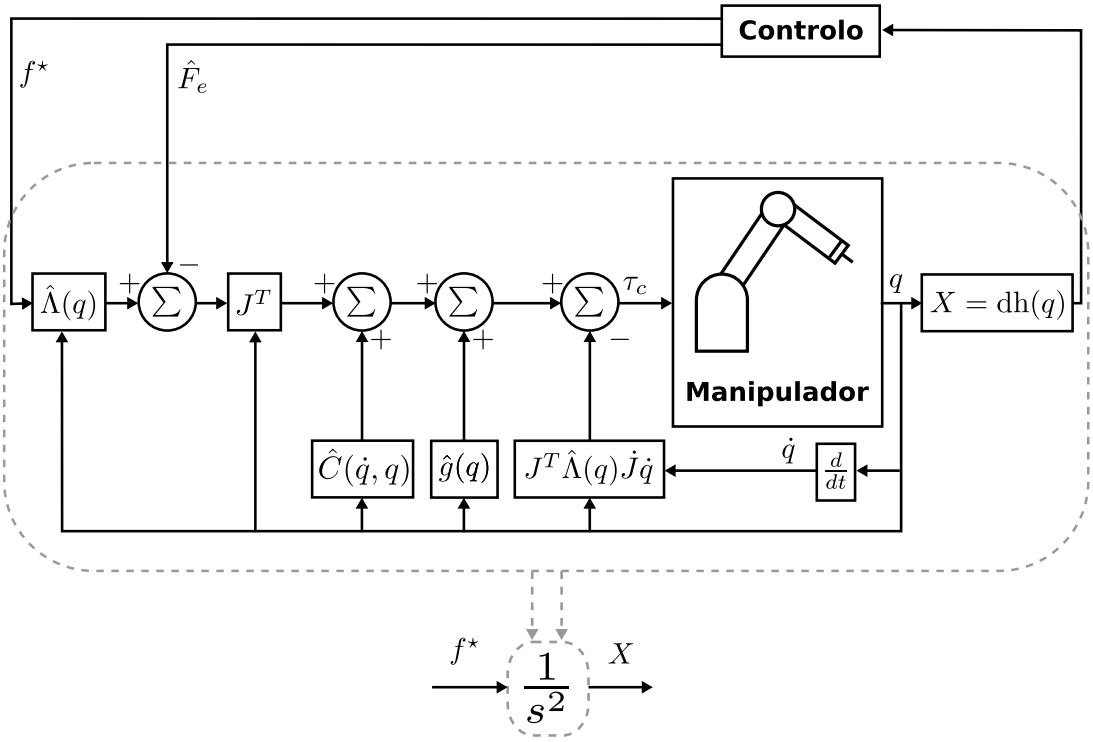


Figura 3.1: Linearização por *feedback* não linear de um manipulador.

3.2 Simulação do manipulador

A simulação de um manipulador robótico consiste em tentar reproduzir, por meio de cálculos computacionais, o comportamento real que o mesmo tem quando é aplicado um binário às suas juntas. Na secção 3.1.1 foi referido que para se conseguir simular correctamente este comportamento é necessário ter o conhecimento da dinâmica do robô, ou seja, conhecer todos os elementos presentes na equação (3.4). A estimação dos parâmetros, inerentes a cada manipulador, que permitem determinar os elementos referidos nem sempre é uma tarefa simples, principalmente para manipuladores tão complexos como o usado no âmbito deste projecto. A estimação incorrecta destes parâmetros pode levar a que os algoritmos de controlo que recorrem à linearização do modelo dinâmico (discutida na secção 3.1.2) não funcionem, mesmo estando correctamente implementados. Esta situação não só torna bastante difícil a detecção de *bugs* nos algoritmos de controlo, caso existam, como dificulta a sua afinação. Implementando estes algoritmos num manipulador simulado é possível testar o controlo em condições "ideais", ou seja, o cancelamento dos termos não lineares do modelo dinâmico é conseguido[6] e a planta do sistema passa a ser exactamente descrita por (3.11).

Na verdade, a equação (3.4) descreve a dinâmica inversa do modelo do robô. No entanto, para implementar a simulação é necessária a dinâmica directa com a qual é possível determinar qual a aceleração adquirida por cada uma das juntas em função

[6]Os termos dinâmicos estimados são os mesmos que descrevem o modelo do simulador.

da sua actual posição e velocidade e do binário que lhe está a ser aplicado. Assim sendo, é necessário trabalhar a equação (3.4) de modo a obter a expressão

$$\ddot{q} = M(q)^{-1}[\tau - c(\dot{q}, q) - g(q) - d(\dot{q}, q)] \tag{3.19}$$

com

$$\tau = \tau_c + \tau_e \ , \tag{3.20}$$

sendo τ_c e τ_e, respectivamente, o binário de comando e o binário resultante do efeito de forças externas (por exemplo forças de contacto com outros objectos). Tendo uma forma de determinar a aceleração das juntas do robô a cada iteração, é possível determinar as suas sucessivas posições e velocidades através da integração no tempo dessa mesma aceleração. O método de integração usado para a implementação deste simulador é um método discreto que é executado em tempo real.

Sendo (3.19) uma equação diferencial de segunda ordem, a sua resolução numérica torna-se mais simples se esta for decomposta em duas equações diferenciais de primeira ordem [18]. Assim, fazendo[7]

$$\dot{q} = v \tag{3.21}$$

$$\dot{v} = F(\tau, q, v) \tag{3.22}$$

e usando a variável auxiliar $y = [q^T \ v^T]^T$, é possível obter a equação

$$\dot{y} = f(\tau, y) \tag{3.23}$$

$$f(\tau, y) = \begin{bmatrix} v \\ F(\tau, q, v) \end{bmatrix} . \tag{3.24}$$

Uma vez que τ é uma variável que depende do tempo, a equação (3.23) pode ser rescrita como

$$\dot{y} = f(t, y) \tag{3.25}$$

$$f(t, y) = \begin{bmatrix} v \\ F(t, q, v) \end{bmatrix} . \tag{3.26}$$

Para efectuar a integração numérica foi usado o método *Runge-Kutta-Fehlberg (RKF45)* [19] dado por:

$$y_{k+1} = y_k + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6 \tag{3.27}$$

[7] $F()$ significa em função de.

com

$$k_1 = hf(t_k, y_k) \tag{3.28}$$

$$k_2 = hf(t_k + \frac{1}{4}h, y_k + \frac{1}{4}k_1) \tag{3.29}$$

$$k_3 = hf(t_k + \frac{3}{8}h, y_k + \frac{3}{32}k_1 + \frac{9}{32}k_2) \tag{3.30}$$

$$k_4 = hf(t_k + \frac{12}{13}h, y_k + \frac{1932}{2197}k_1 + \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3) \tag{3.31}$$

$$k_5 = hf(t_k + h, y_k + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4) \tag{3.32}$$

$$k_6 = hf(t_k + \frac{1}{2}h, y_k - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5) \tag{3.33}$$

, sendo h o passo de integração, que, neste caso corresponde ao tempo decorrido entre duas iterações consecutivas.

3.3 Implementação prática da simulação do manipulador

3.3.1 Manipulador 7-DOF WAM

O simulador desenvolvido neste trabalho é relativo ao manipulador robótico *7-degrees of freedom* (DOF) WAM da empresa Barrett Technology® Inc. que é apresentado na figura 3.2. Este manipulador tem sete juntas de revolução e uma configuração semelhante à de um braço humano[8]. As juntas são controladas em binário através de um computador remoto que comunica com o robô através de um barramento de CAN. É um robô leve cuja transmissão de forças entre os motores e as juntas é feita por meio de cabos, o que torna os seus coeficientes de atrito muito baixos.

Os dados necessários ao cálculo dos modelos geométrico, cinemático e dinâmico foram retirados dos manuais deste manipulador [20, 21].

3.3.2 Arquitectura do *software* desenvolvido

Para a implementação prática do simulador do robô WAM foi desenvolvido um conjunto de módulos/livrarias de *software*, na linguagem de programação C++, responsáveis por efectuar os cálculos dos modelos referidos em 3.1.1, realizar a comunicação do simulador com os módulos de controlo e garantir uma correcta gestão de todas as tarefas envolvidas na simulação. Foi ainda desenvolvido um módulo de interface tridimensional de forma a permitir que o utilizador tenha um *feedback* visual do que está a acontecer durante a simulação.

[8] O "braço" inclui o ombro com três graus de mobilidade, o cotovelo com um e o punho com mais três.

Figura 3.2: Manipulador 7-DOF *WAM*. Acoplado ao *end-effector* está a ferramenta médica abordada no capítulo 2.

Sendo o C++ uma linguagem de programação orientada a objectos, cuja estrutura principal é a classe, este conceito foi bastante usado no desenvolvimento deste simulador. O esquema apresentado na figura 3.3 representa a arquitectura de *software* usada para a simulação. Para a compreensão deste esquema há a salientar que:

- Os elementos representados por rectângulos correspondem às livrarias de C++ desenvolvidas;
- Cada elemento representado por uma elipse corresponde a um objecto de uma classe cujo nome se encontra escrito em itálico dentro dessa mesma elipse;
- O símbolo (▶——) significa que o objecto em causa pertence a uma classe implementada na livraria ligada a esse mesmo símbolo;
- O símbolo (——⋙) significa que o objecto em causa foi criado pelo elemento para onde "aponta" esse mesmo símbolo;
- Os elementos representados por rectângulos com os vértices arredondados correspondem a programas em execução;
- Cada secção dentro de um programa, com o nome em itálico e dividida pelo tracejado, corresponde a um *thread* em execução dentro desse mesmo programa.
- O símbolo (◂·····▸) corresponde a uma ligação efectuada por *sockets*.

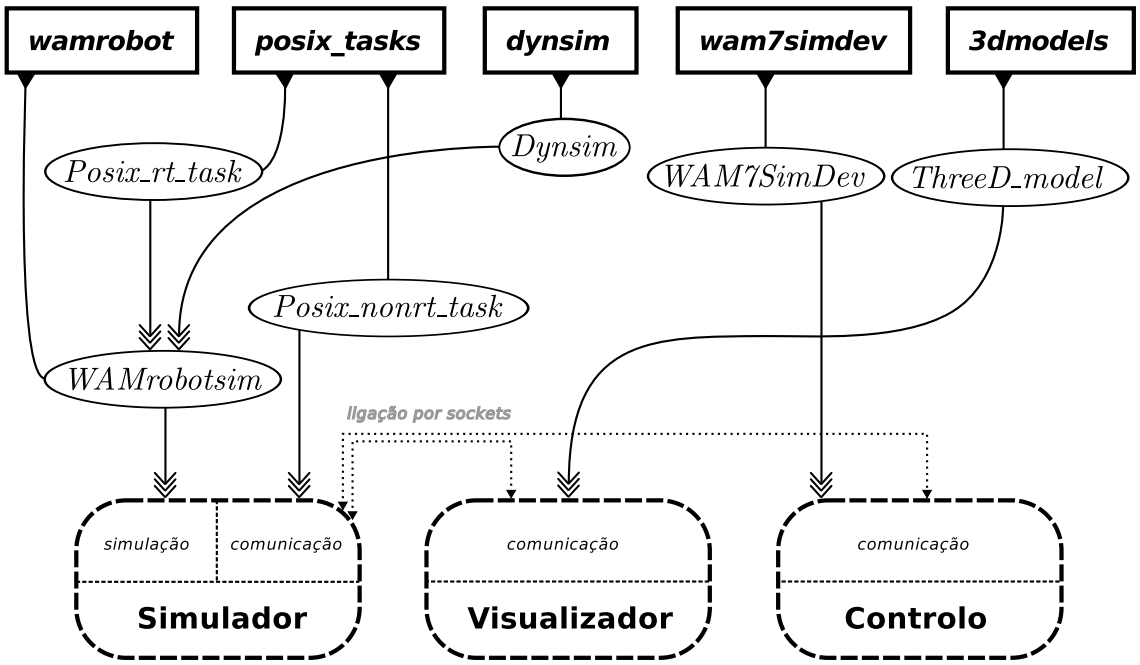


Figura 3.3: Arquitectura de *software* implementada.

A execução em paralelo das várias tarefas necessárias à simulação é conseguida com a livraria *posix_tasks*. Esta livraria permite criar tarefas periódicas e não periódicas (dependendo do construtor do objecto utilizado) que podem, ou não, ser executadas em tempo real, consoante é usada a classe *Posix_rt_task* ou *Posix_nonrt_task*, respectivamente. Por cada objecto criado num programa é iniciado um novo *thread* cujo código a executar é especificado numa função que é definida no momento da criação do objecto em causa. Esta classe foi implementada segundo as normas *POSIX*[9] [22, 23].

Na classe *WAMrobotsim* é feito o cálculo dos modelos geométrico, cinemático e dinâmico, sendo este último calculado com o método de integração numérica (referido em 3.2) implementado na classe *Dynsim*. O cálculo iterativo destes modelos é efectuado numa tarefa periódica de tempo real que é iniciada pela criação de um objecto da classe *Posix_rt_task*. Da análise de vários valores para o período desta tarefa conclui-se que 1,2 ms é um valor suficientemente pequeno para passo de integração da dinâmica que garante que a simulação seja escalonável.

A comunicação do simulador com o visualizador é feita numa tarefa periódica que é iniciada pela criação de um objecto da classe *Posix_nonrt_task*. O período desta tarefa foi escolhido de modo a garantir a frequência de refrescamento pretendida para visualizador (60 Hz). Para tal foi escolhido um período de 10 ms. A comunicação do simulador com o controlo é feita, do lado do simulador, por uma tarefa não periódica da classe *Posix_nonrt_task*. A escolha de uma tarefa não periódica deve-se ao facto de a comunicação do lado do simulador ser sincronizada com uma tarefa

[9]Esta classe foi usada num sistema operativo Linux no qual teve de ser instalada a extensão de *kernel* "linux-rt" para se tirar pleno partido da componente de tempo real.

periódica, criada pela classe *WAM7simDev*, do lado do controlo. Esta sincronização é conseguida pois a comunicação, do lado do simulador, é bloqueante relativamente ao controlo, ou seja, cada vez que são recebidos dados enviados pelo controlo, a tarefa de comunicação no simulador fica suspensa até surgirem novos dados. Por se tratar de tarefas de comunicação, com todos os atrasos a elas inerentes, optou-se pela sua execução em *thread*s que não correm em tempo real.

Do lado do programa do visualizador a comunicação é feita por um objecto da classe *ThreeD_model*, classe esta que é também responsável por criar o modelo tridimensional do manipulador. Esta comunicação é bloqueante relativamente ao simulador.

No programa de controlo a comunicação é efectuada por um objecto da classe *WAM7simDev*. Esta classe foi implementada como uma derivação de uma outra já existente (*RobotDevice*) de onde deriva também a classe *WAM7Device* (também já existente) que representa um manipulador WAM real. Esta implementação permite ao controlo "ver" apenas um único dispositivo *RobotDevice* (que pode corresponder a um manipulador real ou simulado), ficando assim o código com um certo nível de abstracção relativamente aos dispositivos controlados.

Visualizador

O visualizador gráfico criado neste trabalho é um programa que faz a renderização, em ambiente openGL, de um modelo em *3D* do manipulador WAM de acordo com os parâmetros de Denavit-Hartenberg descritos em [21]. A câmara deste visualizador move-se de forma completamente livre no espaço tridimensional e o seu controlo é feito através do teclado e do rato.

De modo a possibilitar um desenvolvimentos simples e rápido de visualizadores gráficos para outros robôs que não o WAM, foram criadas duas livrarias que servem de suporte ao desenvolvimento de um visualizador genérico:

- ***3dmodels*** - usada para importar objectos *3D* especificados em ficheiros no formato *OBJ* e criar um modelo que depois possa ser renderizado pelo openGL. Quando se cria um modelo cuja configuração é condicionada por variáveis de translação/rotação (no caso do WAM as posições das juntas), a livraria *3dmodels* cria uma tarefa de comunicação por *sockets* que permite obter, de programas que se liguem a esta, os valores dessas mesmas variáveis. Cada modelo criado com esta livraria consiste num objecto da classe *ThreeD_model*.

- ***3ddisplay*** - usada para criar um ambiente openGL que incorpora a navegação da câmara e o interface com o teclado e rato referidos. Na criação deste ambiente são iniciados dois *thread*s, como mostra a figura 3.4. Como se pode observar, um dos *thread*s é utilizado para renderizar os modelos criados pela *3dmodels*, enquanto que o outro serve como "monitor" que detecta qualquer evento que requeira a actualização do ambiente gráfico. Desta forma consegue-se que os recursos necessários à renderização deste ambiente só sejam usados quando houver alguma alteração do mesmo.

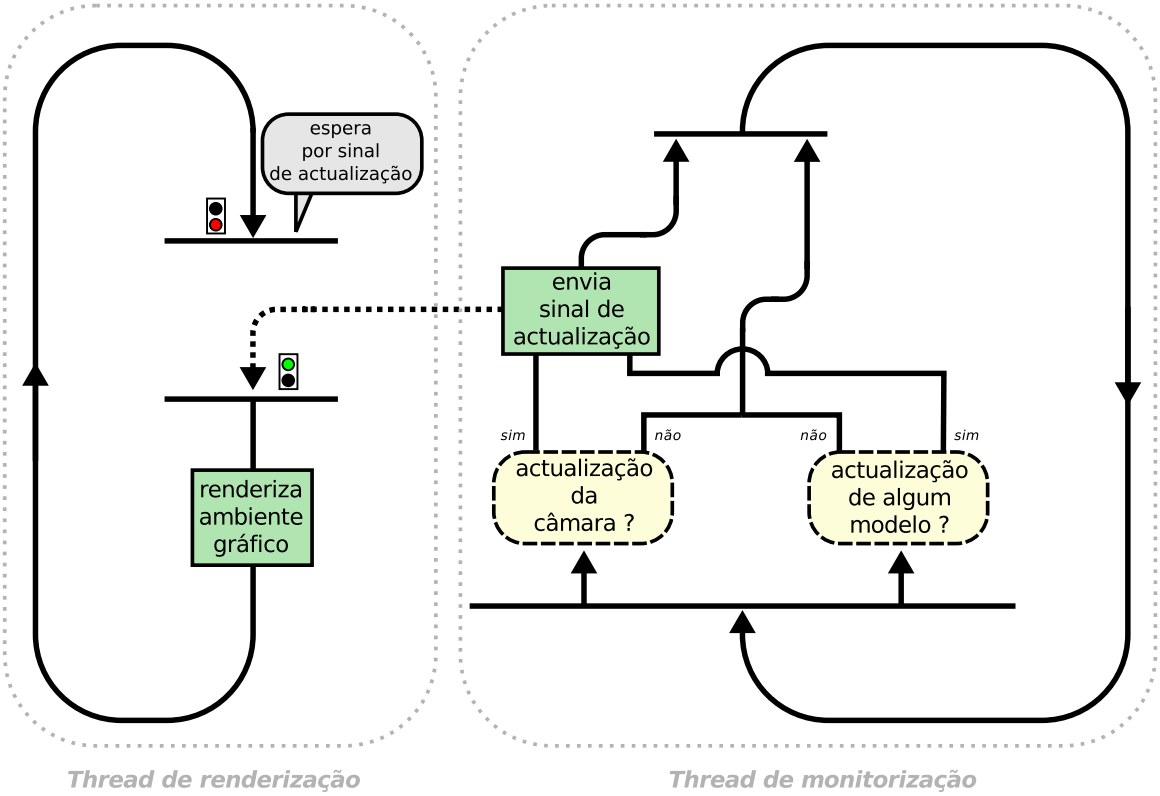


Figura 3.4: Funcionamento interno da livraria *3ddisplay*.

Na figura 3.5 é apresentado o interface gráfico do visualizador desenvolvido. Neste caso, para além do modelo do WAM foi acrescentado o sensor de força que é acoplado ao *end-effector* e um modelo da ferramenta médica apresentada no capítulo 2. Ao ser premida a tecla 'h' é mostrado um menu com todas as opções que a aplicação permite efectuar.

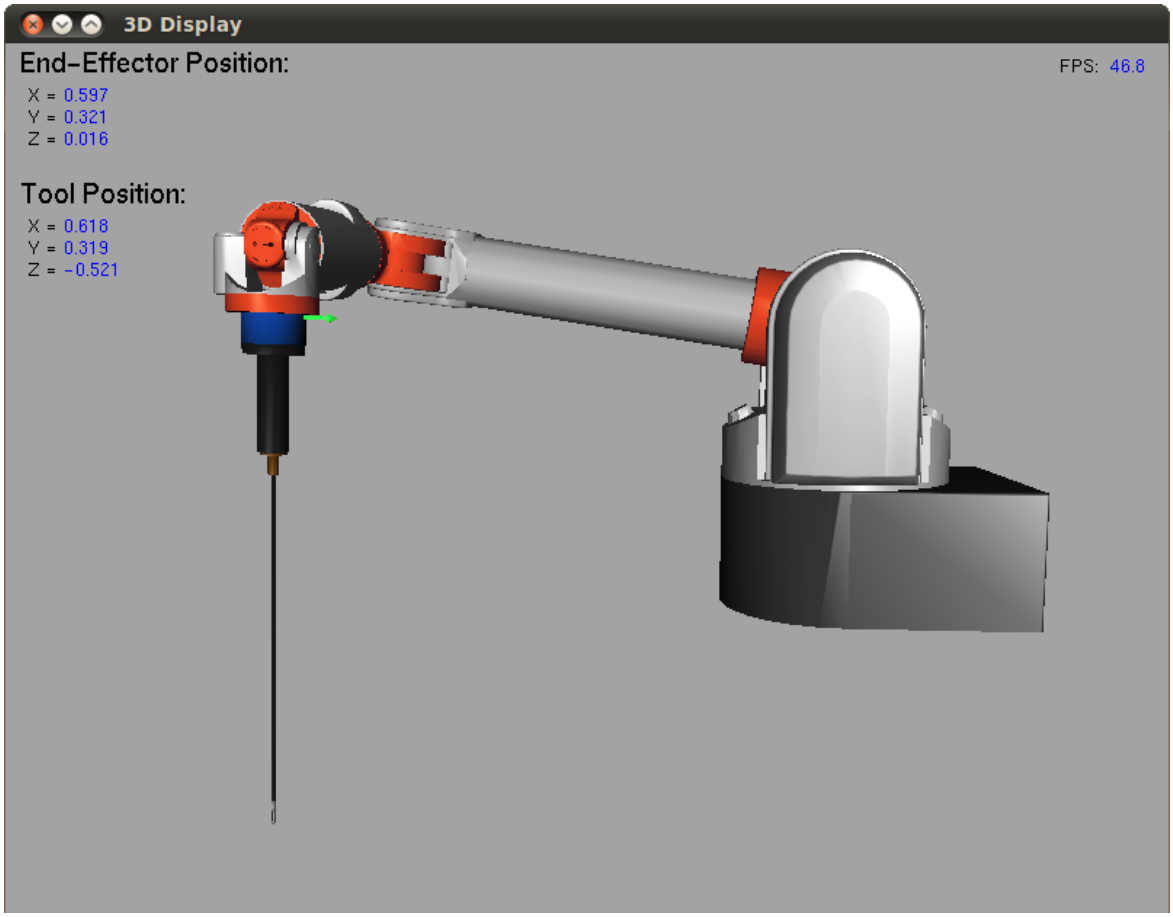


Figura 3.5: Interface gráfico do visualiador.

Capítulo 4

Simulação de forças de contacto

De modo a ser possível testar, num manipulador robótico, algoritmos de controlo para compensação de movimentos fisiológicos em ambiente de simulação, é necessário conseguir reproduzir da forma mais fidedigna possível a interacção desse mesmo manipulador com o meio em que se insere. O efeito da interacção de um manipulador robótico com outros objectos é contemplado no vector de binários τ_e da equação da sua dinâmica (de (3.19) e (3.20)). Com a equação (3.3) é possível determinar este binário desde que seja conhecida a força cartesiana que é aplicada no *end-effector*, resultante do contacto deste com objectos presentes no ambiente simulado. Para determinar esta força é então necessário recorrer a um método que permita não só a detecção de colisões entre objectos mas também que seja capaz de, a partir das colisões detectadas, modelar a força cartesiana total aplicada ao *end-effector* do robô.

O processo para verificar se dois ou mais objectos estão em contacto entre si é chamado de detecção de colisões [24]. Para efectuar a detecção de colisões entre objectos é necessário ter informação sobre a sua geometria. Esta informação é normalmente obtida a partir de modelos gráficos *3D* criados computacionalmente. Estes modelos são compostos por vários polígonos elementares (quadrados ou triângulos) e contêm, geralmente, informação geométrica apenas da superfície do objecto que representam.

A forma mais simples de verificar se um objecto A colide com um objecto B é fazer um teste de intersecção de cada polígono pertencente a A relativamente a cada polígono pertencente a B [24]. Este método rapidamente se torna incomportável, em termos computacionais, com aumento da complexidade dos objectos envolvidos. Uma forma de reduzir significativamente o processamento envolvido na detecção de colisões, principalmente quando estão envolvidos muitos objectos, foi sugerida por Hubbard [25] e consiste em dividir o processo de detecção em duas fases: "alargada" (*broad-phase*) e "estreita" (*narrow-phase*). A fase "alargada" é relativamente simples em termos computacionais e consiste em detectar intersecções não das geometrias complexas dos objectos mas de modelos mais simples envolventes dos mesmos. Estes modelos, designados de volumes limitadores (*bounding volumes*) são volumes justos aos objectos, ou a um grupo de primitivas que os compõem, que geralmente con-

têm cada um desses mesmos objectos (ou grupos de primitivas) na sua totalidade. Nesta fase é possível reduzir drasticamente o número de elementos sobre os quais são feitos os testes de colisão, tornando assim a fase "estreita" mais restrita e consequentemente mais rápida. A fase "estreita" é responsável por determinar de uma forma mais detalhada como é que os objectos seleccionados na fase anterior colidem uns com os outros. Ao contrário da fase "alargada" esta requere um esforço computacional muito maior pelo que é de extrema importância encontrar métodos que permitam reduzir ao máximo o número de elementos sobre os quais a mesma incide.

Vários algoritmos têm sido sugeridos numa tentativa de melhorar o processo de detecção de contactos, tornando-o o mais rápido e fidedigno possível. Os principais métodos actualmente usados podem ser divididos em duas categorias:

- **baseados em hierarquias de volumes limitadores (HVL)** - as primitivas simples que compõem um determinado objecto vão sendo agrupadas, sucessivamente, em volumes limitadores cada vez maiores, formando estes uma árvore hierárquica como a mostrada na figura 4.1 [26]. Estes métodos permitem acelerar o processo de detecção de colisões através da redução significativa das primitivas a considerar nos testes de intersecção..

- **métodos de decomposição espacial** - ao contrário dos métodos de HVL que efectuam uma decomposição ao nível dos objectos, estes métodos baseiam-se na decomposição do espaço em várias sub-regiões, também designadas de células, que podem ser, ou não, iguais entre si. Neste caso a detecção de colisões baseia-se no facto de dois objectos não poderem ocupar a mesma célula espacial a não ser que estejam a colidir um com o outro. Um método baseado na decomposição espacial uniforme (células todas iguais/uniformes), que é muito utilizado na modelação de objectos em aplicações de imagiologia médica, é o método baseado em *voxel*s [27, 28].

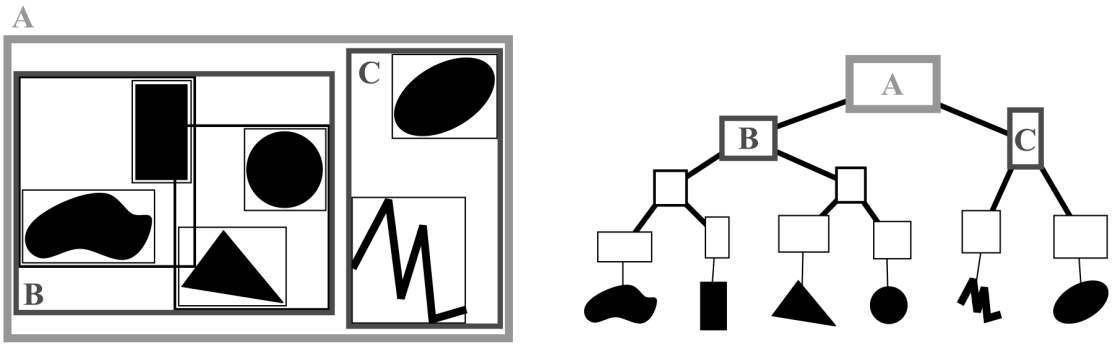


Figura 4.1: Exemplo de uma hierarquia usando rectângulos como volumes limitadores.

Apesar de alguns dos métodos mais usados serem extremamente eficazes e rápidos, todos eles se baseiam no pre-processamento dos dados que utilizam, ou seja, quer a decomposição dos objectos por métodos de HVL quer a decomposição espacial é feita

uma única vez antes de ser iniciada a simulação. Esta característica torna impossível estender estes métodos a objectos deformáveis pois a sua modelação necessitaria que fosse efectuada uma decomposição cada vez que ocorresse uma mudança na forma dos mesmos. Tal situação tornaria qualquer dos métodos referidos incompatíveis com aplicações que requerem taxas de detecção de colisões extremamente elevadas[1]. Uma solução para este problema é apresentada por Eisemann e Décoret [13] e consiste num método de voxelização (modelação dos objectos usando *voxel*s) totalmente efectuado na unidade de processamento gráfico (GPU) de um computador. Este método permite uma voxelização dos modelos extremamente rápida o que possibilita que a mesma seja feita a cada iteração do algoritmo de detecção de colisões.

Neste capítulo é apresentada uma solução, baseada no método referido anteriormente, que foi desenvolvida para simular as forças aplicadas ao *end-effector* (ou a outra ferramenta a ele acoplada) resultantes da interacção deste com objectos virtuais em movimento.

4.1 Voxelização de modelos *3D*

A voxelização de um modelo consiste na criação de uma estrutura tridimensional, formada por *voxel*s, que represente virtualmente a estrutura do mesmo. Esta estrutura é criada a partir do modelo gráfico superficial do objecto sendo que cada *voxel* preenchido corresponde a uma célula do espaço contida total ou parcialmente por esse mesmo modelo. Na figura 4.2 são apresentados os modelos gráfico e voxelizado de um objecto. A representação de um objecto por *voxel*s tem ainda a vantagem de

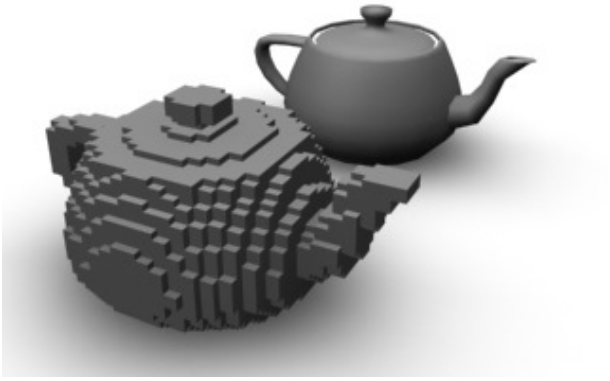


Figura 4.2: Modelos gráfico (a trás) e voxelizado (à frente) de um objecto.

se poder ajustar o nível de detalhe do modelo voxelizado em função do custo computacional pretendido para as operações efectuadas com o mesmo. Assim quanto maior for o tamanho de cada *voxel* menor é o custo computacional associado, mas menos preciso é o modelo quando comparado com o objecto real.

[1]Em aplicações que envolvem feedback háptico a frequência de actualização de colisões é na ordem dos 1000 Hz

4.1.1 Voxelização implementada na GPU

A renderização de uma cena na placa gráfica de um computador define implicitamente uma grelha a duas dimensões cujo número de células[2] é definido pela resolução da imagem obtida. Durante o processo de rasterização a placa gráfica analisa cada primitiva e gera o conjunto de fragmentos a ela associada. Cada fragmento é caracterizado pelas coordenadas (x, y) do pixel a ele associado bem como por uma coordenada z de profundidade usada para determinar, para cada pixel, qual o fragmento mais próximo do plano de imagem, desprezando assim todos os outros fragmentos por ele ocultados. Com a coordenada z deixa-se de ter uma grelha a duas dimensões para se passar para uma grelha tridimensional, tornando assim o processo de renderização de uma imagem ideal para a voxelização de objectos. A ideia base desta implementação consiste em guardar, durante o processo de rasterização , a informação relativa a todos os fragmentos gerados (incluindo os ocultados) e armazenar essa informação nos canais RGBA da placa gráfica, ou seja, codificar a informação numa imagem/textura.

Para efectuar a voxilização é necessário definir o *frustum* da câmera virtual de acordo com os objectos que se pretende voxilizar, bem como a resolução da imagem que vai ser gerada. Esta resolução é especificada no *viewport* da câmara e as suas dimensões especificam a resolução da grelha tridimensional segundo x e y. A resolução segundo z é especificada pelo número total de *bits* usados pela placa gráfica para a codificação da cor, como é mostrado na figura 4.3. No esquema apresen-

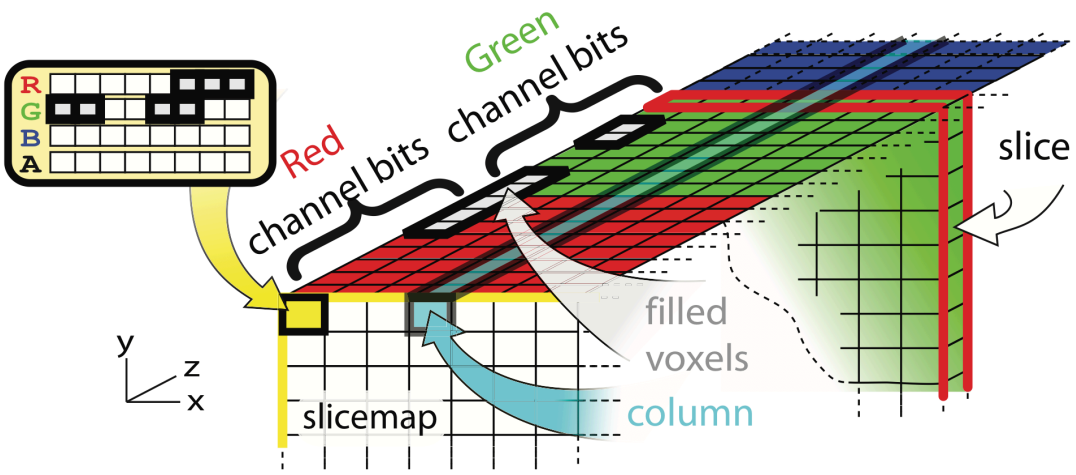


Figura 4.3: Esquema de um *slicemap*.

tado, cada pixel (x, y) corresponde a uma coluna da grelha, segundo z. O valor das células de cada coluna é codificado no valor RGBA do pixel correspondente. Esta codificação é feita considerando que cada valor RGBA não corresponde a quatro *bytes* (um para cada valor de cor) mas sim a um vector único de 32 *bits*, cada um representando o valor de uma célula da coluna. O valor de cada célula é colocado a

[2]No caso de uma imagem, as células correspondem a pixeis.

um se alguma primitiva a intersecta e é colocado a zero caso contrário. O conjunto de todas as células correspondentes a um determinado *bit* de cor é designado por *slice* e consequentemente cada imagem/textura gerada é designada de *slicemap*.

Sendo 32 o número de *bits* usados para a codificação de cor, o número de *voxel*s possíveis, segundo z, num *slicemap* é bastante reduzido. Para melhorar esta situação pode usar-se uma funcionalidade que as placas gráficas actuais têm designada de Multiple Render Targets (MRT). Esta funcionalidade permite à placa gerar múltiplas imagens/texturas em simultâneo. Desta forma, e tendo em conta que actualmente uma placa gráfica comum possui 8 MRT, é possível expandir a resolução do *slicemap*, segundo z, até 256.

Uma vez codificada a voxelização sob a forma de uma textura, presente na memória gráfica, é necessário apenas fazer a leitura da mesma e, descodificando-a, obter as coordenadas cartesianas dos centros de todos os *voxel*s. Esta descodificação é feita em função do *frustum* definido para a câmara e do tamanho escolhido para os *voxel*s.

O método apresentado implementa uma voxelização superficial de objectos. Devido à forma como é implementado este método apresenta algumas limitações na voxelização de primitivas com um alinhamento muito próximo do do eixo z da grelha (*slicemap*), como se pode ver pela figura 4.4. No contexto do trabalho desenvolvido

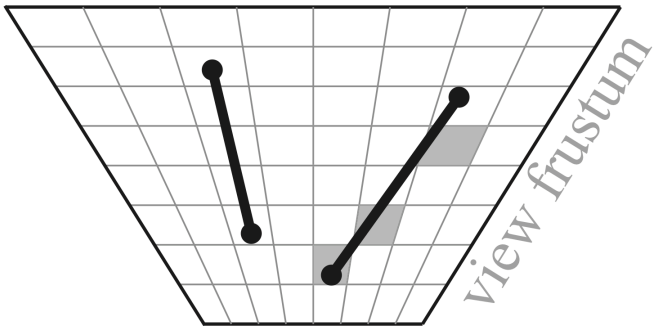


Figura 4.4: Limitações da voxelização superficial. A primitiva da esquerda encontra-se alinhada com uma coluna e por isso passa completamente despercebida à voxelização. Na primitiva da direita, como o seu alinhamento é muito próximo do do eixo z, a voxelização não é contígua.

estas limitações não são significativas pois os objectos que se pretende modelar não apresentam superfícies completamente planas pois representam elementos pertencentes ao corpo humano.

Uma forma de contornar as limitações referidas é recorrer ao trabalho também apresentado por Eisemann e Décoret [29] em que a voxelização dos modelos é completa e não apenas superficial.

4.2 Algoritmo para determinação de forças de contacto

O método apresentado nesta secção é responsável por calcular as forças que actuam na extremidade da ferramenta médica (apresentada no capítulo 2) que se encontra acoplada ao *end-effector* do manipulador WAM. Por causa da elevada taxa de actualização das forças que é necessária para os algoritmos de controlo (cerca de 1000 actualizações por segundo), este método foi desenvolvido de modo a ser compatível com a voxelização superficial dos modelos *3D*, uma vez que é mais rápida que a voxelização total.

O algoritmo desenvolvido consiste em determinar qual o plano de incidência da ferramenta numa determinada estrutura e, sabendo qual a distância da extremidade dessa ferramenta ao plano, determinar a força exercida. A força calculada é normal ao plano de incidência e é dada por:

$$F_e = K_s \times d \times \vec{N} \ , \tag{4.1}$$

em que K_s é rigidez da estrutura , d é a distância da extremidade da ferramenta ao plano e $\vec{N}$ é o vector normal ao plano cujo sentido aponta para "fora" da estrutura em causa. Devida à configuração e dimensões da extremidade desta ferramenta optou-se por considerar que a mesma é representada por um único *voxel*. As coordenadas do centro deste *voxel* correspondem à posição cartesiana da extremidade da ferramenta (P_{tool}). Desta forma não é necessário voxelizar a ferramenta e por isso consegue-se uma maior rapidez para o método de modelação de forças.

O plano de incidência corresponde ao plano de ajuste, obtido pelo método dos mínimos quadrados totais [30], dos N pontos mais próximos da extremidade da ferramenta. O valor de N tem de ser no mínimo igual ao menor número de pontos que caracterizam um plano, ou seja, igual a três. Na figura 4.5 encontra-se repesentado o método descrito.

Mínimos quadrados totais

Com este método pretende-se encontrar um plano de ajuste, definido por

$$ax + by + cz + d = 0 \ , \tag{4.2}$$

de tal maneira que a soma das distâncias dos N pontos escolhidos ao mesmo seja mínima. Para tal considere-se a expressão da distância de um ponto a um plano [31]:

$$D = \frac{|ax_0 + by_0 + cz_0 + d|}{\sqrt{a^2 + b^2 + c^2}} \ , \tag{4.3}$$

com (a, b, c, d) os coeficientes da equação do plano e (x_0, y_0, z_0) as coordenadas do ponto. Para minimizar D pode calcular-se o valor mínimo para D^2, pois D é sempre não negativo. Assim, a equação (4.3) pode ser rescrita como:

$$D = \frac{|ax_0 + by_0 + cz_0 + d|^2}{a^2 + b^2 + c^2} \ . \tag{4.4}$$

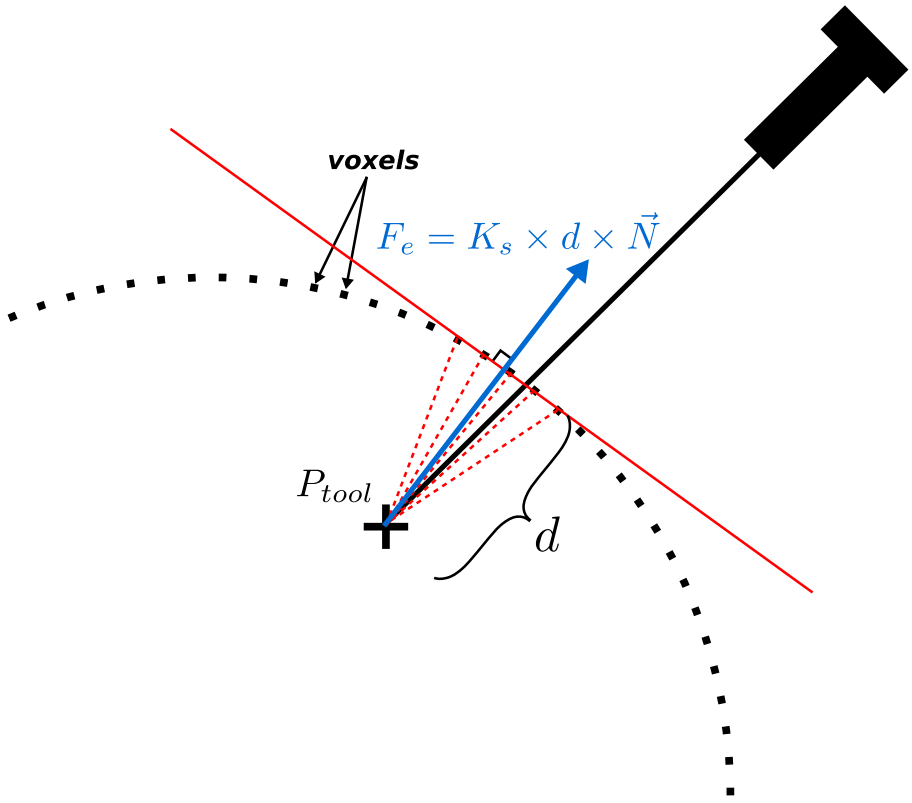


Figura 4.5: Esquema do método usado para o cálculo das forças de contacto.

Pretende-se determinar a, b, c e d que minimizem a função $f(a, b, c, d)$ dada por

$$f(a,b,c,d) = \sum_{i=1}^{N} D_i \tag{4.5}$$

$$= \sum_{i=1}^{N} \frac{|ax_i + by_i + cz_i + d|^2}{a^2 + b^2 + c^2} \ , \tag{4.6}$$

com (x_i, y_i e z_i) as coordenadas de cada um dos N pontos mais próximos da extremidade da ferramenta. Fazendo a derivada parcial da função em ordem a d igual a zero e resolvendo a equação de modo a determinar d, obtém-se:

$$d = -(ax_c + by_c + cx_c) \ , \tag{4.7}$$

em que (x_c, y_c, z_c) são as coordenadas do centróide dos N pontos. De (4.7) conclui-se que o centróide pertence ao plano de ajuste pretendido. Substituindo d na expressão (4.6) plano obtém-se:

$$f(a,b,c) = \sum_{i=1}^{N} \frac{|a(x_i - x_c) + b(y_i - y_c) + c(z_i - z_c)|^2}{a^2 + b^2 + c^2} \ . \tag{4.8}$$

Utilizando uma representação matricial e considerando as variáveis auxiliares

$$v^T = \begin{bmatrix} a & b & c \end{bmatrix} \tag{4.9}$$

$$M = \begin{bmatrix} x_1 - x_c & y_1 - y_c & z_1 - z_c \\ x_2 - x_c & y_1 - y_c & z_1 - z_c \\ \vdots & \vdots & \vdots \\ x_N - x_c & y_N - y_c & z_N - z_c \end{bmatrix} , \tag{4.10}$$

pode rescrever-se (4.8) como

$$f(v) = \frac{(v^T M^T)(Mv)}{v^T v} \tag{4.11}$$

$$= \frac{v^T (M^T M) v}{v^T v} . \tag{4.12}$$

Considerando uma variável $W = M^T M$, a função $f(v)$ corresponde ao quociente de Rayleigh[3] de M e v, que é minimizado pelo vector próprio correspondente ao menor valor próprio de W. Podiam ser calculados os vectores próprios de W, mas não é necessário uma vez que se pode obter o vector v a partir da decomposição SVD da matriz M. Sendo esta decomposição dada por

$$M = U \Sigma V^T , \tag{4.13}$$

pode-se igualar a matriz W a

$$W = (U \Sigma V^T)^T (U \Sigma V^T) \tag{4.14}$$

$$= (V \Sigma^T U^T)(U \Sigma V^T) \tag{4.15}$$

$$= V \Sigma^2 V^T . \tag{4.16}$$

Sendo W uma matrix simétrica, a equação (4.16) representa a sua decomposição espectral, estando os seus valores próprios em Σ^2. Assim, os valores prórpios de W correspondem aos quadrados dos valores singulares de M e os vectores próprios de W são os vectores singulares de M.

O plano de ajuste pretendido é então um plano que contém o centróide dos N pontos cuja normal é o vector singular correspondente ao menor valor singular de M.

4.3 Implementação prática da simulação de contactos

Para a implementação prática dos métodos apresentados nas secções anteriores foi desenvolvido o módulo *contacts*, em C++, que inclui também algumas rotinas de openGL para o processo de voxelização na GPU. Este módulo foi implementado de

[3] http://en.wikipedia.org/wiki/Rayleigh_quotient

modo a que a voxelização e o cálculo das forças de contacto fossem executados em tarefas distintas que são executadas em paralelo. Desta forma aproveita-se o facto de os dois métodos serem relativamente independentes em termos de execução, para se conseguir um aumento na rapidez de cada iteração da simulação de contactos. A implementação das duas tarefas paralelas requere a existência de um buffer de dados partilhado que pode ser simples ou duplo como é mostrado na figura 4.6. A implementação de um buffer simples tem como vantagem o requisito de menos

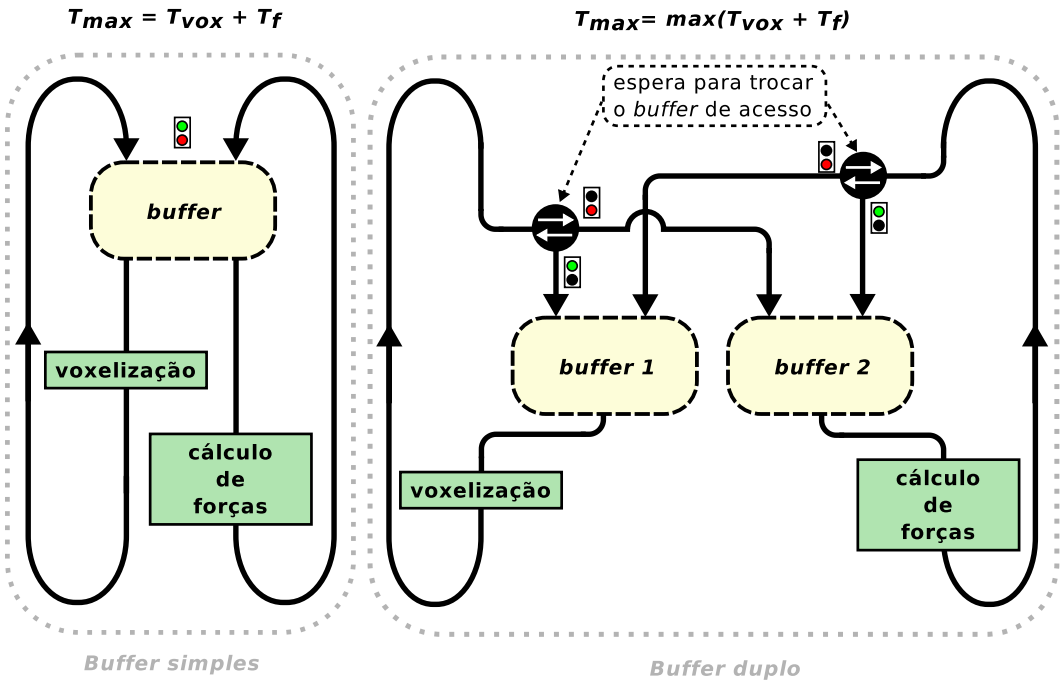


Figura 4.6: Simulação de forças para um *buffer* de dados simples e duplo.

memória, mas o tempo de execução total, no pior caso, corresponde à soma dos tempos de execução dos dois métodos. A utilização de um buffer duplo embora exija o dobro da memória utilizada, permite que o tempo de execução total seja sempre igual ao tempo de execução do método mais demorado. Uma vez que para o trabalho em causa se pretende que a simulação seja o mais rápida possível, optou-se pela implementação de um buffer de dados duplo.

O método dos *slicemaps*, descrito em 4.1.1, necessário à voxelização só pode ser implementado se se tiver acesso a determinadas "etapas" do *pipeline* gráfico. Este *pipeline*, apresentado na figura 4.7, é o responsável pela realização de todo o processo de renderização que ocorre nas placas gráficas. Ele é constituído por vários módulos incluindo algumas unidades programáveis (rectângulos verdes na figura 4.7), unidades essas que tornam possível o método de voxelização apresentado. No contexto deste trabalho a programação destas unidades consistiu na criação de dois programas em GLSL, um para ser executado pelos *vertex shaders* e o outro pelos *fragment shaders*.

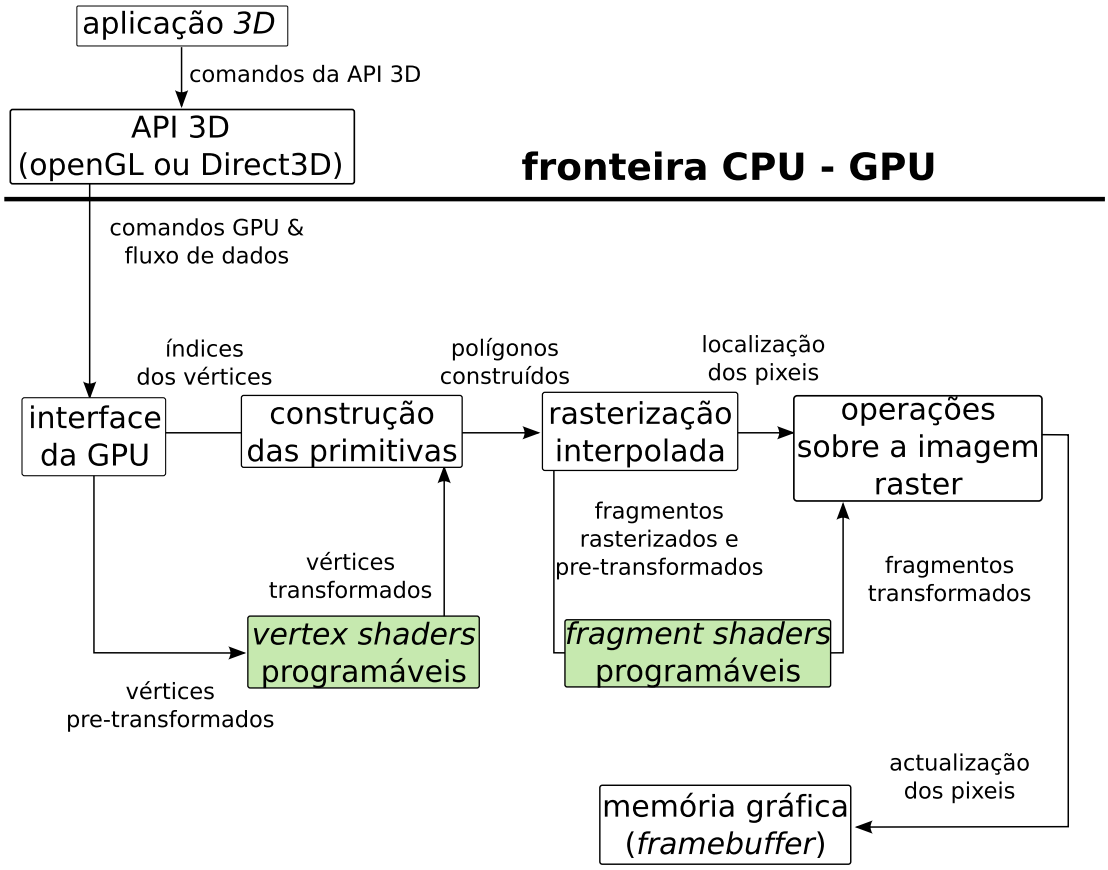


Figura 4.7: Esquema simplificado de um *pipeline* gráfico.

Criação de um *slicemap*

Antes de se construir um *slicemap* é necessário "limpar" o *framebuffer* onde ele vai ser criado, colocando todos os *bits* a zero. Para gerar o *slicemap* é necessário determinar, para cada primitiva, quais os *voxel*s que esta intersecta, sendo os *bits* correspondentes colocados a um. Durante o processo de rasterização de uma primitiva é criado um fragmento para cada coluna intersectada por esta. Por causa da interpolação existente no pipeline gráfico, é possível obter a coordenada z de cada fragmento criado a partir das coordenadas z dos vértices, sendo estas últimas calculadas nos *vertex shaders* com as matrizes de transformação. A coordenada z de cada fragmento permite determinar a que *slice* o mesmo pertence. Para codificar esta coordenada numa textura/imagem é necessário mapeá-la de forma linear em valores compreendidos entre zero e um. Esse mapeamento é feito pela expressão:

$$z_{norm} = \frac{z + z_n}{z_n - z_f} \ , \tag{4.17}$$

em que z_n corresponde à coordenada z do plano mais próximo da câmara e z_f à do plano mais afastado. O valor normalizado z_{norm} é usado para fazer a consulta de uma textura 1D que representa uma máscara cujo número de *bits* é igual ao número de *slices* do *slicemap*. A codificação das coordenadas z dos fragmentos é

feita desta forma porque é muito mais rápido fazer consultas de texturas do que cálculos nos *fragment shaders*. Finalmente, para garantir que todas as coordenas z dos fragmentos de uma mesma coluna sejam codificadas na imagem final, é necessário utilizar a operação lógica OR, disponível no openGL, durante a construção dessa mesma imagem.

Descodificação de um *slicemap*

O método de descodificação dos *slicemaps* foi implementado recorrendo ao multi-processamento com *thread*s. Esta implementação, representada na figura 4.8, não só permite que a codificação e a descodificação de um *slicemap* sejam de certo modo executadas em simultâneo como acelera o processo de descodificação no caso de existirem vários *framebuffers* MRT para descodificar.

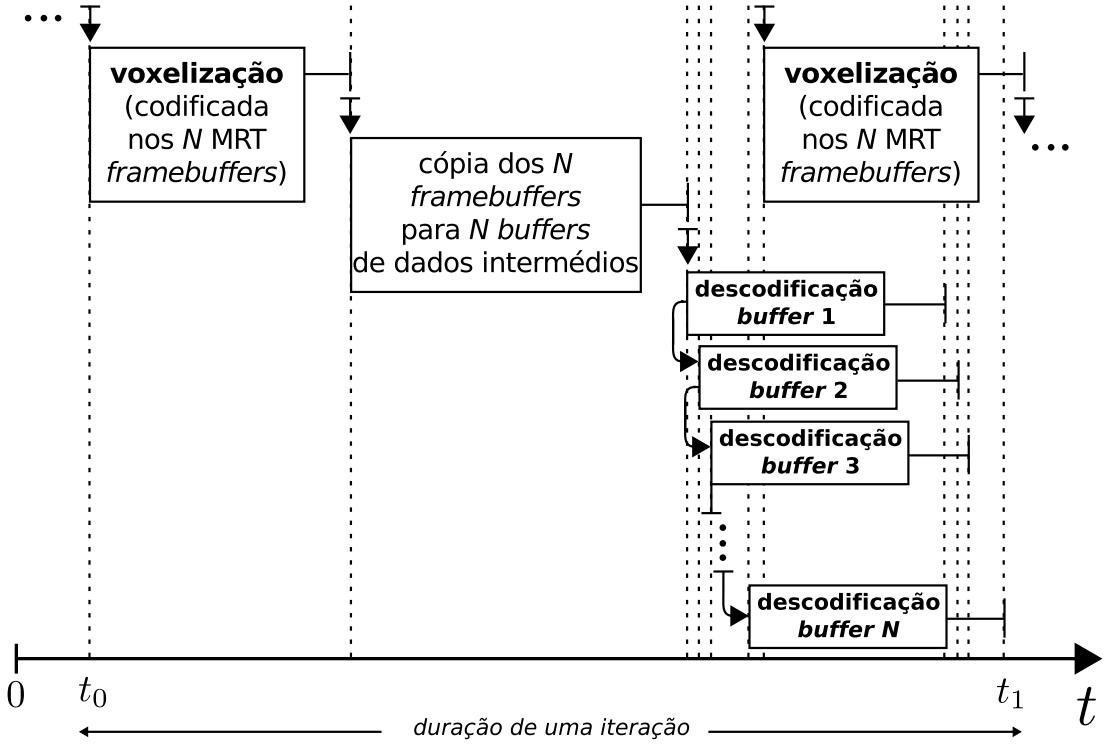


Figura 4.8: Sequência de codificação/descodificação de um *slicemap* (a relação das durações entre tarefas é meramente ilustrativa, não correspondendo às durações reais).

Capítulo 5

Controlo para compensação de movimentos fisiológicos

5.1 Controlo preditivo linear baseado no modelo

O controlo preditivo baseado no modelo, ou MPC, é um método de controlo que utiliza o modelo explícito do sistema que controla para prever o seu comportamento futuro ao longo de um horizonte definido. Este método tem sido alvo de um intenso estudo ao longo das últimas décadas e é actualmente uma das metodologias de controlo avançadas mais utilizadas em aplicações industriais. Esta vasta aceitação ao nível industrial deve-se ao facto do MPC não só permitir o controlo eficaz de sistemas com múltiplas variáveis mas também ser possível especificar no próprio controlador as restrições dos vários processos a controlar. Outra característica do MPC é a sua capacidade de reagir de forma dinâmica às variações que podem ocorrer num processo, como por exemplo falhas de equipamentos. Esta propriedade tem especial importância no âmbito deste projecto pois permite que este método possa ser usado para compensar as perturbações resultantes dos batimentos cardíacos numa cirurgia assistida por robôs.

5.1.1 Formulação

Para um sistema linear discreto

$$\begin{cases} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{cases}, \tag{5.1}$$

em que u_k é o comando de entrada e x_k é o vector de estados do sistema, o método MPC é descrito pelos seguintes passos (ver figura 5.1):

1. É definida uma trajectória de referência w_{k+i} segundo a qual a saída do sistema deve atingir o sinal de referência r_k.

2. Através do modelo do sistema definido por (5.1), são calculadas as futuras saídas do mesmo para cada instante k ao longo de um horizonte de predição

H_p. Estas previsões das saídas[1] $\tilde{y}_{k+i}$ ($i \in [1, H_p]$) são calculadas com base nos valores passados das saídas e entradas, conhecidas no instante k, e com base no sinal de controlo $\tilde{u}_{k+i}$ ($i \in [0, H_p - 1]$) futuro.

3. Os valores $\tilde{u}_{k+i}$ são calculados, tendo em conta o horizonte de controlo H_u, de modo a minimizar uma função de custo. Esta função tem em conta os erros entre as previsões das saídas $\tilde{y}_{k+i}$ e a trajectória de referência w_{k+i} assim como o esforço de controlo. Sendo o modelo em causa linear, é possível calcular de forma analítica a sua solução mínima.

4. O primeiro elemento da sequência de controlo $\tilde{u}_{k+i|i=0}$ é aplicado ao sistema e o horizonte de predição é deslocado em direcção ao futuro. O algoritmo é repetido com os valores actualizados.

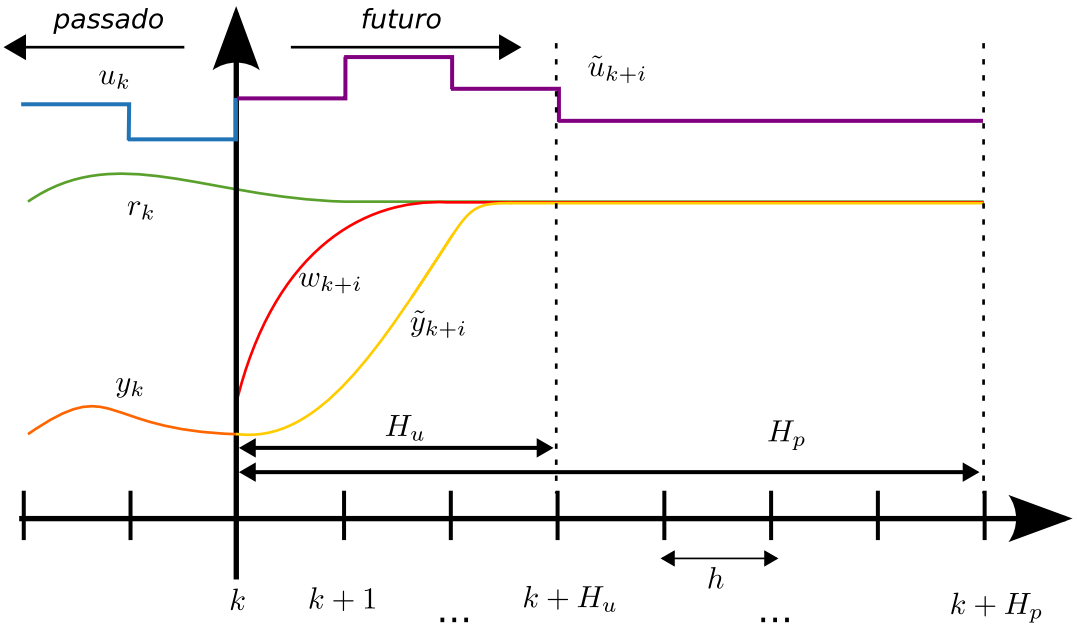


Figura 5.1: Exemplo ilustrativo da evolução temporal de algumas variáveis envolvidas no controlo preditivo.

De modo a garantir que o sistema, a partir do valor actual da saída y_k, atinge a referência dentro do intervalo de predição, w_{k+i} é definido por

$$w_{k+i} = r_{k+i} - \phi^i \epsilon_k \ , \tag{5.2}$$

com ϵ_k o erro na saída $(r_k - y_k)$. r_{k+i} tem um valor constante ao longo de H_p e ϕ^i é dado por

$$\phi^i = e^{-i(h/T_{ref})} \ , \tag{5.3}$$

com h o período de amostragem e T_{ref} a constante de tempo da dinâmica exponencial escolhida para a trajectória.

[1] O til colocado por cima da letra significa que se trata de uma previsão do sinal correspondente

O cálculo das previsões dos valores futuros do vector de estados ao longo de H_p é feito de forma iterativa através de (5.1):

$$\begin{array}{rcl} \tilde{x}_{k+1} & = & Ax_k + B\tilde{u}_k \\ \tilde{x}_{k+2} & = & A\tilde{x}_{k+1} + B\tilde{u}_{k+1} \\ & = & A^2x_k + AB\tilde{u}_k + B\tilde{u}_{k+1} \\ \vdots & & \vdots \\ \tilde{x}_{k+H_p} & = & A\tilde{x}_{k+H_p-1} + B\tilde{u}_{k+H_p-1} \\ & = & A^{H_p}x_k + A^{H_p-1}B\tilde{u}_k + \ldots + B\tilde{u}_{k+H_p-1} \ . \end{array} \tag{5.4}$$

Considerando que $\tilde{u}_k$ apenas pode variar no horizonte de controlo H_u, com $H_u \leq H_p$, o mesmo pode ser descrito por

$$\begin{array}{rcl} \tilde{u}_k & = & \Delta\tilde{u}_k + u_{k-1} \\ \tilde{u}_{k+1} & = & \Delta\tilde{u}_{k+1} + \Delta\tilde{u}_k + u_{k-1} \\ \vdots & & \vdots \\ \tilde{u}_{k+H_u-1} & = & \Delta\tilde{u}_{k+H_u-1} + \ldots + \Delta\tilde{u}_k + u_{k-1} \ , \end{array} \tag{5.5}$$

com

$$\Delta\tilde{u}_{k+i} = \left\{ \begin{array}{ll} \tilde{u}_k - u_{k-1} & , i = 0 \\ \tilde{u}_{k+i} - \tilde{u}_{k+i-1} & , i \neq 0 \end{array} \right. . \tag{5.6}$$

Sendo $\Delta\tilde{u}_{k+i} = 0$ e $\tilde{u}_{k+i} = \tilde{u}_{k+H_u-1}$ para $i \geq H_u$, e com (5.4) e (5.5), obtém-se:

$$\begin{array}{rcl} \tilde{x}_{k+1} & = & Ax_k + B[\Delta\tilde{u}_k + u_{k-1}] \\ \tilde{x}_{k+2} & = & A^2x_k + B[\Delta\tilde{u}_{k+1} + \Delta\tilde{u}_k + u_{k-1}] + AB[\Delta\tilde{u}_k + u_{k-1}] \\ & = & A^2x_k + B\Delta\tilde{u}_k(A+I) + Bu_{k-1}(A+I) + B\Delta\tilde{u}_{k+1} \\ \vdots & & \vdots \\ \tilde{x}_{k+H_u} & = & A^{H_u}x_k + B\Delta\tilde{u}_k(A^{H_u-1} + \ldots + A + I) \\ & & + Bu_{k-1}(A^{H_u-1} + \ldots + A + I) \\ & & + B\Delta\tilde{u}_{k+H_u-1} \\ \tilde{x}_{k+H_u+1} & = & A^{H_u+1}x_k + B\Delta\tilde{u}_k(A^{H_u} + \ldots + A + I) \\ & & + Bu_{k-1}(A^{H_u} + \ldots + A + I) \\ & & + B\Delta\tilde{u}_{k+H_u-1}(A + I) \\ \vdots & & \vdots \\ \tilde{x}_{k+H_p} & = & A^{H_p}x_k + B\Delta\tilde{u}_k(A^{H_p-1} + \ldots + A + I) \\ & & + Bu_{k-1}(A^{H_p-1} + \ldots + A + I) \\ & & + B\Delta\tilde{u}_{k+H_u-1}(A^{H_p-H_u} + A + I) \ . \end{array} \tag{5.7}$$

Utilizando uma representação matricial pode rescrever-se (5.7) como

$$\tilde{y}_k = \Psi x_k + \Upsilon u_{k-1} + \Theta \left[\begin{array}{c} \Delta\tilde{u}_k \\ \vdots \\ \Delta\tilde{u}_{k+H_u-1} \end{array} \right] , \tag{5.8}$$

com

$$\begin{aligned}
\Psi &= \begin{bmatrix} CA \\ \vdots \\ CA^{H_u} \\ CA^{H_u+1} \\ \vdots \\ CA^{H_p} \end{bmatrix} \quad \Upsilon = \begin{bmatrix} CB \\ \vdots \\ \sum_{i=0}^{H_u-1} CA^iB \\ \sum_{i=0}^{H_u} CA^iB \\ \vdots \\ \sum_{i=0}^{H_p-1} CA^iB \end{bmatrix} \\
\Theta &= \begin{bmatrix} CB & \dots & 0 \\ C(AB+B) & \dots & 0 \\ \vdots & \ddots & \vdots \\ \sum_{i=0}^{H_u-1} CA^iB & \dots & CB \\ \sum_{i=0}^{H_u} CA^iB & \dots & C(AB+B) \\ \vdots & \ddots & \vdots \\ \sum_{i=0}^{H_p-1} CA^iB & \dots & \sum_{i=0}^{H_p-H_u} CA^iB \end{bmatrix} .
\end{aligned} \tag{5.9}$$

O método de predição, dado pela equação (5.8), é composto por dois termos (Ψ e Υ) que dependem dos valores passados e actuais dos estados, no instante k e por um termo (Θ) que depende do vector que contem os valores preditos [32]. Ψ e Υ constituem a reposta livre do sistema e Θ representa a resposta por parte deste a um degrau unitário.

A sequência de controlo $\Delta\tilde{u}$ é obtida pela minimização da função de custo

$$J = (\tilde{y}_k - w_k)^T \delta (\tilde{y}_k - w_k) + \Delta\tilde{u}^T \lambda \Delta\tilde{u} \ , \tag{5.10}$$

em que δ e λ são respectivamente os pesos do erro e do esforço de controlo. A função J penaliza as diferenças entre as previsões $\tilde{y}_k$ da saída do sistema e a trajectória de referência w_k. Considerando E_k como a diferença entre a trajectória de referência w_k e a resposta livre do sistema

$$E_k = w_k - \Psi x_k - \Upsilon u_{k-1} \ , \tag{5.11}$$

a função de custo pode ser rescrita como

$$J = (\Theta\Delta\tilde{u} - E_k)^T \delta (\Theta\Delta\tilde{u} - E_k) + \Delta\tilde{u}^T \lambda \Delta\tilde{u} \ . \tag{5.12}$$

Desenvolvendo (5.12) obtém-se:

$$J = E_k^T \delta E_k - 2\Delta\tilde{u}^T \Theta^T \delta E_k + \Delta\tilde{u}^T (\Theta^T \delta \Theta + \lambda) \Delta\tilde{u} \ . \tag{5.13}$$

Para minimizar J basta calcular o seu gradiente parcial e igualá-lo a zero:

$$\frac{\partial J}{\partial \Delta\tilde{u}} = -2\Theta^T \delta E_k + 2(\Theta^T \delta \Theta + \lambda)\Delta\tilde{u} = 0 \ . \tag{5.14}$$

A partir de (5.14) é possível obter o vector $\Delta\tilde{u}$ que é igual a

$$\Delta\tilde{u} = (\Theta^T \delta \Theta + \lambda I)^{-1} \Theta^T \delta E_k \ . \tag{5.15}$$

De modo a garantir que o valor encontrado minimiza a função J basta determinar o seu gradiente de segunda ordem e garantir que o mesmo é positivo. O gradiente de segunda ordem de J é dado por

$$\frac{\partial^2 J}{\partial \Delta \tilde{u}^2} = 2(\Theta^T \delta \Theta + \lambda) \ , \tag{5.16}$$

e considerando que $\delta \geq 0$, é garantido que $\Theta^T \delta \Theta \geq 0$. Para assegurar que $\Delta\tilde{u}$ minimiza de facto J, basta impor um valor positio para λ.

Ao sinal de comando é então adicionado apenas o primeiro valor $\Delta\tilde{u}_{k+i|i=0}$ da sequência de controlo $\Delta\tilde{u}$ determinada.

5.1.2 Implementação no manipulador WAM

Sendo o sistema que descreve o WAM completamente não-linear, é necessário utilizar a linearização por *feedback* do modelo dinâmico, discutida em 3.1.2, de modo a ser possível implementar o método MPC linear apresentado.

Para simular a interacção do manipulador com objectos presentes num ambiente cirúrgico, é usado um modelo simples de uma mola com uma rigidez K_s, correspondente à rigidez de cada objecto considerado. Para o controlo em força é ainda acrescentado um *feedback* que garante uma resposta amortecida para a posição cartesiana do manipulador. A constante de amortecimento é representada por K_2 como mostra a figura 5.2. A planta do sistema com controlo em força é então dada por

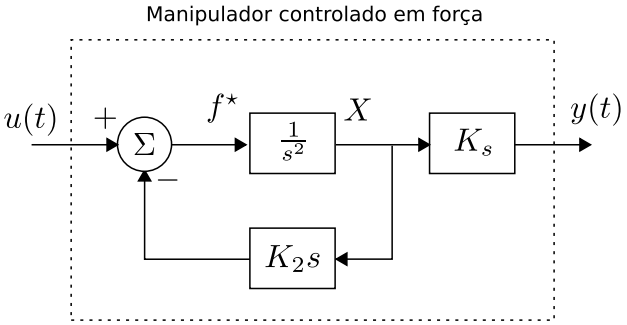


Figura 5.2: Planta do sistema com comando de força $u(t)$ e saída em força $y(t)$.

$$G_s = \frac{K_s}{s(s + K_2)} \ , \tag{5.17}$$

cuja representação temporal é dada por [17]

$$\ddot{y}(t) + K_2 \dot{y}(t) = K_s u(t), \tag{5.18}$$

sendo $y(t)$ a força aplicada pelo manipulador e $u(t)$ o sinal de comando. Considerando duas variáveis de estado $x_1(t) = y(t)$ e $x_2(t) = \dot{y}(t)$ A representação em espaço de estados deste sistema é então dada por

$$\begin{cases} \dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ 0 & -K_2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ K_s \end{bmatrix} u(t) \\ y(t) &= x(t) \end{cases} \ , \tag{5.19}$$

com $x(t) = \begin{bmatrix} x_1(t) & x_2(t) \end{bmatrix}^T$. Descretizando com um tempo de amostragem h[2], obtêm-se as matrizes necessárias à implementação do MPC.

Na figura 5.3 encontra-se representada a arquitectura de controlo usada para a compensação de movimentos fisiológicos com o método MPC. O observador presente

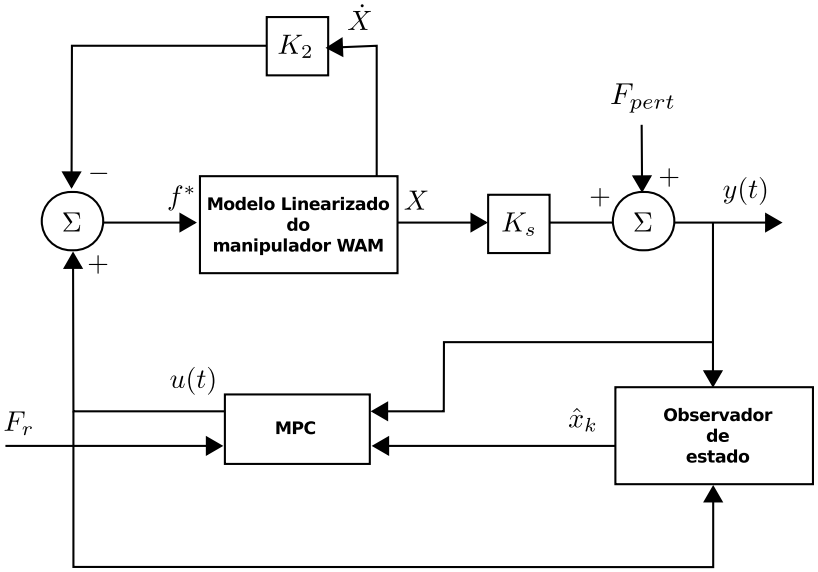


Figura 5.3: Arquitectura de controlo para compensação de movimentos. F_r e F_{pert} correspondem,respectivamente, à força de referência e à perturbação de força. $\hat{x}_k$ é o vector de estados, estimados pelo observador a partir da leitura da força medida $y(t)$.

nesta arquitectura permite a estimação do vector de estados x_k com base na leitura de y_k. Uma vez que a dinâmica, no simulador do WAM, é actualizada somente a cada 1,2 ms (referido em 3.3.2), este foi o valor escolhido para o tempo de amostragem utilizado no controlador.

Ajuste dos parâmetros que caracterizam a actuação do MPC

O tamanho do horizonte H_p influência bastante a capacidade do sistema seguir a referência que lhe é fornecida. Embora a especificação de um horizonte de predição muito alargado permita ao sistema atingir o valor de referência com elevada exactidão, por outro lado o tempo de cálculo computacional também se torna maior. É então necessário arranjar um compromisso para o valor de H_p que satisfaça as especificações do controlo pretendido. O valor de T_{ref} influência directamente a dinâmica da resposta do sistema, sendo que quanto menor for o seu valor mais dinâmica é essa resposta e vice-versa. Relativamente a δ e λ, apenas a sua diferença relativa é significativa para o ajuste do controlo MPC.

[2]O sistema em espaço de estados discreto pode ser facilmente obtido numericamente a partir modelo continuo com a função do Matlab™ *c2dt*

Capítulo 6

Resultados experimentais

Neste capítulo são apresentados os resultados obtidos com o método de voxelização implementado no GPU, assim como os relativos à implementação da arquitectura de controlo preditivo, para compensação de perturbações externas em força. Para as experiências realizadas com esta arquitectura foi utilizado o *software* apresentado nos capítulos 3 e 4, relativo ao simulador do manipulador WAM e à simulação de forças de contacto, respectivamente.

6.1 Voxelização

Para garantir uma correcta voxelização dos objectos antes destes serem incluídos na simulação de contactos, foi criada uma aplicação que permite visualizar a representação em *voxel*s de qualquer objecto *3D*, descrito por um ficheiro no formato *OBJ*. Na figura 6.1 é apresentada a voxelização de uma esfera centrada na origem do referencil cartesiano. Nesta figura é possível verificar as limitações (referidas em 4.1.1)

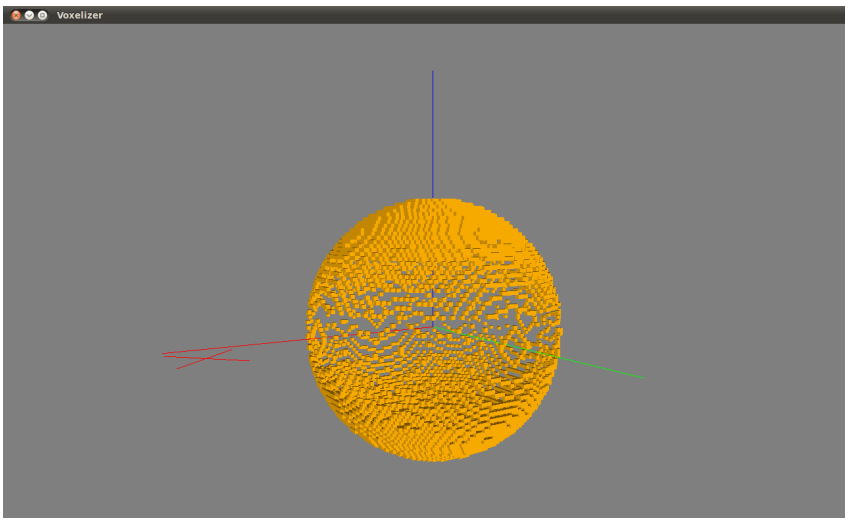


Figura 6.1: Voxelização superficial de uma esfera centrada na origiem do referencial (as linhas vermelha, verde e azul representam os eixos x, y e z, respectivamente).

do método de voxelização usado, ou seja, quanto mais uma região da superfície da esfera se encontra "alinhada" com o eixo z menor é o número de *voxel*s a representarem essa região, devido a um número de primitivas que são ignoradas. Estas limitações, no entanto, não constituem qualquer impedimento para a aplicação do método que calcula as forças de contacto, pois, apesar de reduzido, o número de *voxel*s existentes nas regiões referidas é suficiente para a determinação de um plano de ajuste (plano de incidência).

Na figura 6.2 encontra-se representada a voxelização de um modelo *3D* de um coração. Apesar deste ser um modelo relativamente complexo, é possível obter um

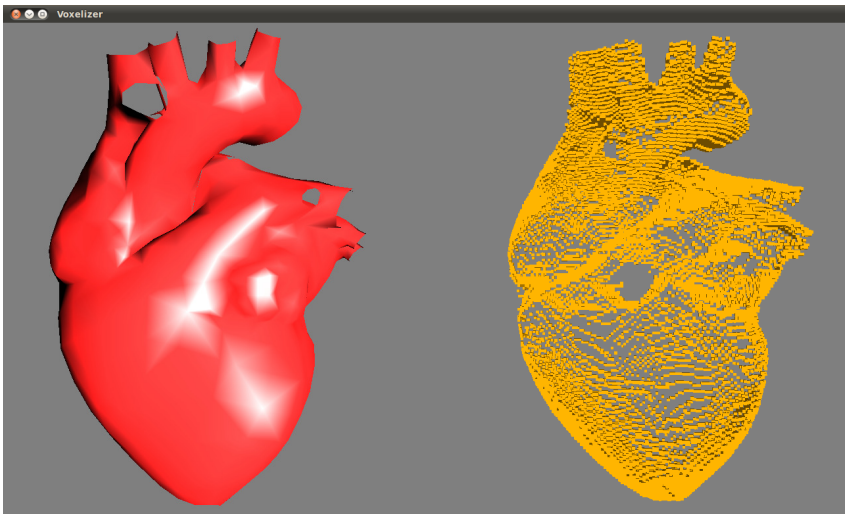


Figura 6.2: Voxelização superficial de um modelo 3D de um coração (modelo 3D à esquerda e respectiva voxelização à direita).

modelo voxelizado que descreva relativamente bem a estrutura superficial o objecto *3D* em causa, desde que a dimensão de cada *voxel* seja suficientemente pequena. Na tabela 6.1 são apresentados os tempos conseguidos para a voxelização dos dois modelos referidos (esfera e coração).

Tabela 6.1: Tempos de execução do algoritmo de voxelização. (DV) dimensão de um *voxel*. (RGV) resolução da grelha de *voxel*s. (CS) codificação do *slicemap*. (LMG) leitura da memória gráfica. (DS) Descodificação do *slicemap*.

	Esfera		**Coração**	
DV	1,4 mm	1 mm	2 mm	1,4 mm
RGV	85^3	120^3	$88 \times 77 \times 131$	$126 \times 110 \times 187$
CS + LMG	0.75 ms	1.4 ms	0.92 ms	1.4 ms
DS	1,55 ms	4,1 ms	2,58 ms	6,1 ms

Analisando os valores temporais obtidos verifica-se que durante a execução do algoritmo de voxelização, a etapa mais demorada corresponde à descodificação do

slicemap criado pela placa gráfica. Esta situação já era de esperar pois a etapa referida é executada inteiramente no processador, que não é tão indicado para o processamento paralelo como a placa gráfica.

6.2 Controlo preditivo linear

Para testar a implementação da arquitectura de controlo preditivo apresentada no capítulo 5, foram realizadas duas experiências. Na primeira foi criado um plano virtual, oblíquo a todas as direcções cartesianas, cuja translação era efectuada ao longo de uma recta perpendicular ao mesmo. O controlo preditivo foi implementado de modo a que a ferramenta médica acoplada ao *end-effector* do manipulador exercesse no plano uma força o mais constante possível, tendo em conta o movimento referido. Na segunda experiência foi feita a compensação de um movimento vertical (segundo o eixo z), sendo o *feedback* de força obtido com o método de simulação de contactos abordado em 4.

O projecto do controlador usado nas duas experiências foi feito como descrito no capítulo 5. O controlador foi projectado para se ter um sistema com uma resposta criticamente amortecida, sendo a constante de tempo $\tau_c = 0.03$ s e o factor de amortecimento $K_2 = 10$. O projecto estocástico do observador foi feito de modo a que o mesmo se baseie apenas na medida de força para a estimação do primeiro estado e que a estimação dos outros estados seja feita tendo em conta apenas o modelo do sistema. As matrizes correspondentes à incerteza do modelo e da medida foram

$$Q_k = \begin{bmatrix} 10^{10} & 0 & 0 & 0 \\ 0 & 10^{-24} & 0 & 0 \\ 0 & 0 & 10^{-24} & 0 \\ 0 & 0 & 0 & 10^{-24} \end{bmatrix} \text{ e } R_k = 1 \ , \tag{6.1}$$

respectivamente.

Em ambas as experiências foi considerada, para os elementos em contacto com o manipulador, uma rigidez de 900 [N/m], que corresponde, aproximadamente, à rigidez associada aos tecidos cardíacos. Assim, foi escolhida uma rigidez nominal $K_s = 900$ [N/m].

6.2.1 Compensação de perturbações em força num plano oblíquo virtual

Para ajustar o algoritmo de controlo preditivo foi necessário definir valores apropriados para o horizonte de controlo H_u e de predição H_p, bem como para os valores

de δ, λ e T_{ref}. Os valores escolhidos para esta experiência foram

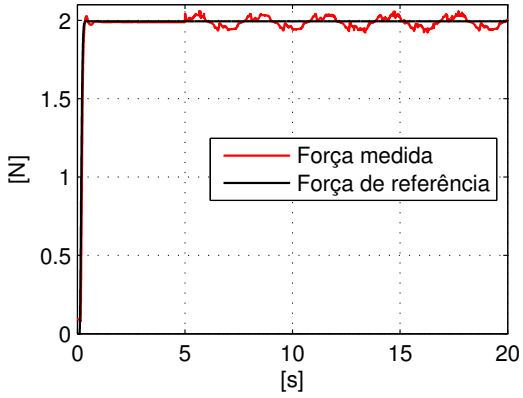
$$\begin{array}{lcl} H_u & = & 30 \\ H_p & = & 64 \\ T_{ref_x} & = & T_{ref_y} = T_{ref_z} = 7 \times h \\ \delta_x & = & 0.45 \\ \delta_y & = & 0.4 \\ \delta_z & = & 0.5 \\ \lambda_x & = & 0.55 \\ \lambda_y & = & 0.6 \\ \lambda_z & = & 0.5 \ , \end{array} \tag{6.2}$$

com h igual ao tempo de amostragem. Embora os valores de H_u e H_p tenham sido iguais para controlo segundo as três coordenadas cartesianas, os valores de δ e λ foram escolhidos de modo a que a dinâmica do controlador fosse mais elevada segunda as direcções onde a perturbação em força produzisse maior efeito. Na figura 6.3 são apresentados os resultados obtidos, tendo sido usada a perturbação em força, com 9 N de amplitude pico-a-pico, presente em 6.4(a). Uma vez que a rigidez considerada para o plano foi de 900 [N/m], esta perturbação corresponde a um deslocamento total do plano de 10 mm. Este valor corresponde aproximadamente ao deslocamento da superfície do coração quando este se encontra em batimento livre [32]. Nesta figura são apresentadas as forças medidas e dadas como referência para as três coordenadas cartesianas do espaço de tarefa do manipulador. A acção do controlo preditivo permitiu reduzir a amplitude pico-a-pico da perturbação aplicada em 98,5% segundo x, 99% segundo y e 96,5% segundo z. Os erros em posição correspondentes podem ser calculados a partir destas medidas, bastando para isso dividir os valores de força obtidos pela rigidez $K_s = 900$ [N/m] considerada para o plano. Os erros totais em posição foram de 0,15 mm segundo x, menos de 0,1 mm segundo y e 0.34 mm segundo z.

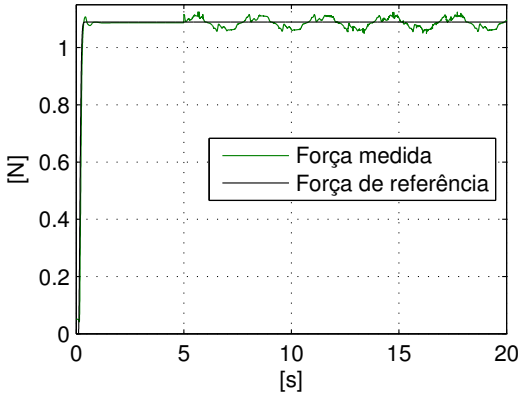
Nesta experiência foi feito ainda um segundo teste que consistiu em substituir a perturbação, em força, sinusoidal por uma com a mesma amplitude, mas constituída por três sinusóides, apresentada em 6.4(b). A figura 6.5 apresenta os resultados obtidos com este segundo teste. Nesta situação a amplitude pico-a-pico da perturbação aplicada foi reduzida em 97,3% segundo x, 98,5% segundo y e 94% segundo z, sendo os correspondentes erros totais em posição de 0,27 mm segundo x, 0,15 mm segundo y e 0,6 mm segundo z. Devido à dinâmica superior da perturbação aplicada nesta situação, os erros em posição obtidos também foram superiores.

6.2.2 Compensação de perturbações em força geradas com a simulação de contactos

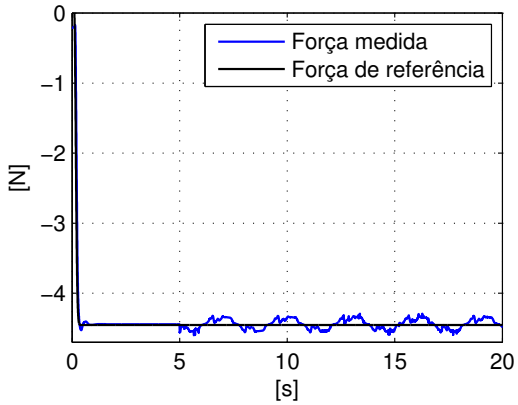
Nesta secção são apresentados os resultados da aplicação do controlo preditivo para compensação de perturbações em força originadas pela expansão e contracção de uma esfera virtual em contacto com a ferramenta médica acoplada ao manipulador. As forças geradas são calculadas pelo método de simulação de contactos descrito em



(a) Forças segundo x.



(b) Forças segundo y.

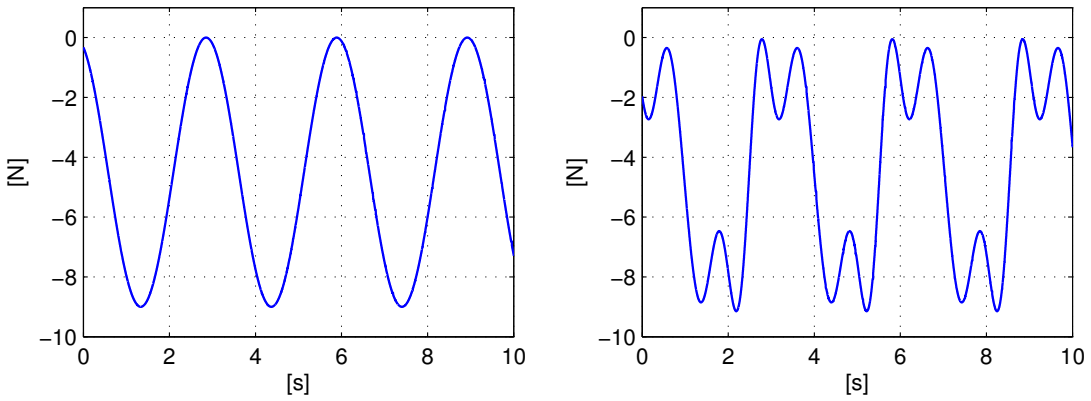


(c) Forças segundo z.

Figura 6.3: Resultados da acção do controlo preditivo sobre um plano virtual oblíquo para uma perturbação de força sinusoidal. A perturbação é iniciada em $t = 5$ s.

4.2, a partir de um modelo voxelizado da esfera. O movimento a ela aplicado foi escolhido de modo a serem geradas as perturbações utilizadas na secção anterior. Para que estas perturbações se manifestem apenas segundo o eixo z, posicionou-se o manipulador de modo a que a ferramenta a ele acoplada ficasse verticalmente alinhada com o centro da esfera.

Para analisar a capacidade do método de geração de forças de calcular correctamente a força resultante do contacto da ferramenta médica com a esfera em movimento, foi criado um plano virtual que produzisse, na extremidade da ferramenta, uma variação de força semelhante à originada por esse mesmo contacto. Na figura 6.6 é apresentada a comparação das forças medidas para os dois casos. Da análise da figura é possível verificar que quanto maior é a dimensão dos *voxel*s, maior é o erro da medição da força quando comparada com a medição de referência. Este erro é de 0,9 N e 0,27 N para uma dimensão dos *voxel*s de 2 mm e 0,6 mm respectivamente. Considerando uma rigidez de 900 [N/m], os erros em força referidos correspondem, respectivamente, a erros de posição de 1 mm e 0,3 mm. Os erros



(a) Perturbação em força sinusoidal.

(b) Perturbação em força constituída por três sinusóides.

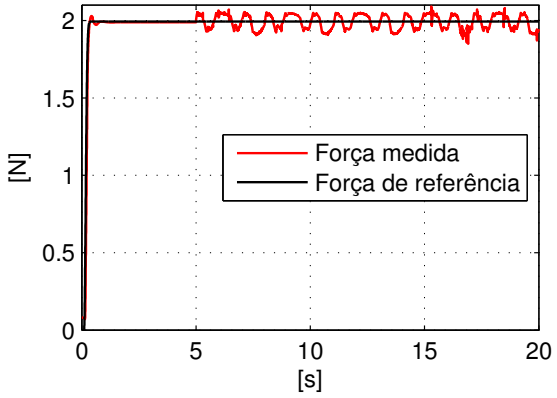
Figura 6.4: Perturbações em força usadas para testar o controlador preditivo implementado.

obtidos podem dever-se ao facto de, durante o processo de voxelização, qualquer primitiva que se encontre na extremidade de um *voxel*, ser mapeada para o centro desse mesmo *voxel*, originando-se um erro de mapeamento com metade do tamanho do mesmo. A escolha da dimensão de cada *voxel* tem de ser feita de modo a obter-se um compromisso entre um erro baixo de posição e um tempo de computação do algoritmo aceitável (quanto menor o tamanho de cada *voxel*, maior o número de *voxel*s a serem processados).

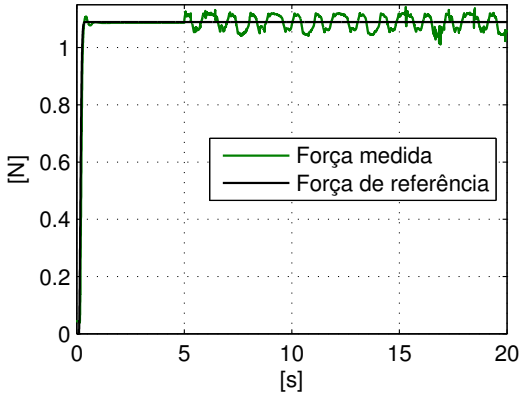
Para a implementação do algoritmo de controlo foi escolhido uma dimensão para os *voxel*s de 0,6 mm. Devido aos tempos de processamento do método de voxelização, apresentados em 6.1, foi necessário proceder à voxelização de apenas uma região de esfera. Desta forma foi possível garantir que cada iteração do método de voxelização não demorasse mais do que 1,2 ms, que corresponde ao tempo de amostragem usado para o controlo. Os parâmtros de ajuste do controlo preditvo escolhidos nesta experiência foram semelhantes aos utilizados na experiência anterior, com excepção dos valores de λ e δ para as três dimensões cartesianas:

$$\begin{aligned} \delta_x &= \delta_y = \lambda_x = \lambda_y = 0.5 \\ \delta_z &= 0.6 \\ \lambda_z &= 0.4\ . \end{aligned} \tag{6.3}$$

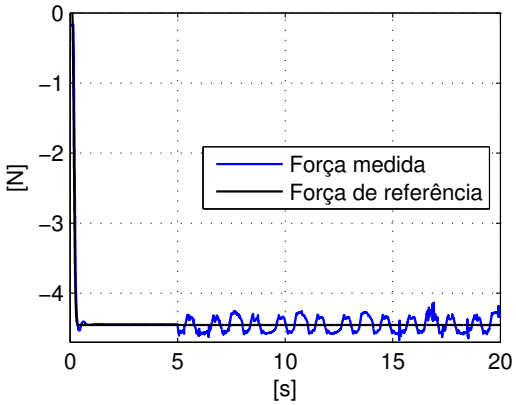
Uma vez que a perturbação de força é aplicada somente segunda a coordenada z, foi escolhido um valor mais baixo para λ_z, e consequentemente um mais elevado para $delta_z$, de modo a que o esforço de controlo seja maior segundo esta dimensão. As figuras 6.7 e 6.8 apresentam os resultados obtidos com a aplicação do controlo preditivo. As medições de força não nulas, segundo x e y, são justificadas pela curvatura da superfície da esfera, que faz com que qualquer pequena variação horizontal na posição da ferramenta médica se traduza no aparecimento de componentes de força ao longo dessas direcções. Esta situação dificulta também a acção do controlador,



(a) Forças segundo x.



(b) Forças segundo y.

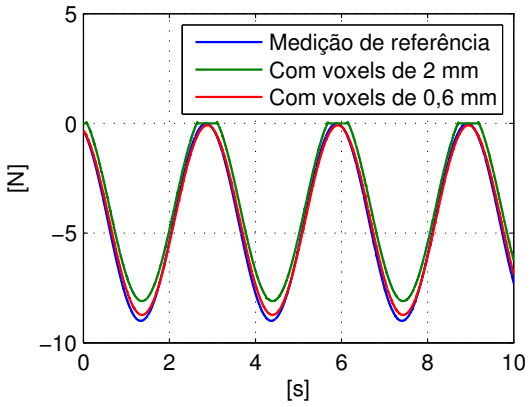


(c) Forças segundo z.

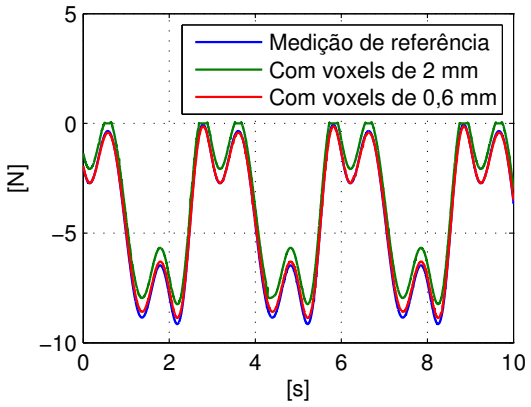
Figura 6.5: Resultados da acção do controlo preditivo sobre um plano virtual oblíquo para uma perturbação de força composta por três sinusóides. A perturbação é iniciada em $t = 5$ s.

obtendo-se assim uma pior compensação de movimentos.

Nesta experiência só faz sentido analisar a compensação de movimentos segundo o eixo z. Assim, a amplitude pico-a-pico da perturbação aplicada foi reduzida em 89,3%, para uma variação de força sinusoidal, e em 87,3% para uma variação de força composta por três sinusóides. Os erros de posição correspondentes foram de 1,1 mm para o primeiro caso e de 1,26 mm para o segundo.

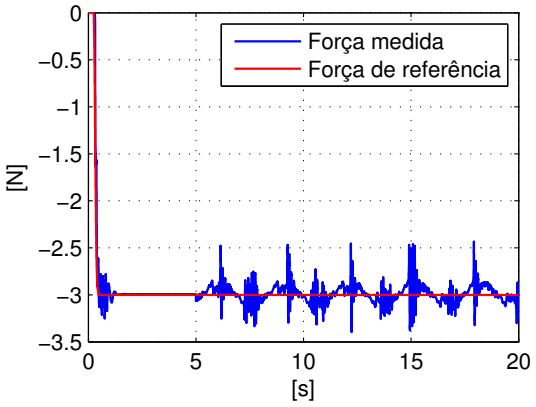


(a) Perturbações em força sinusoidaias.

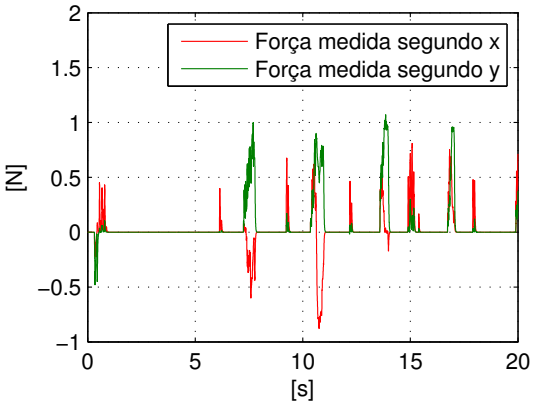


(b) Perturbações em força compostas por três sinusóides.

Figura 6.6: Medição de força no ponto correspondente à extremidade da ferramenta acoplada ao manipulador. A medição de referência foi obtida com o plano virtual e as restantes com o método de geração de forças, tendo os *voxel*s um tamanho de 2 mm e 0,6 mm, respectivamente.

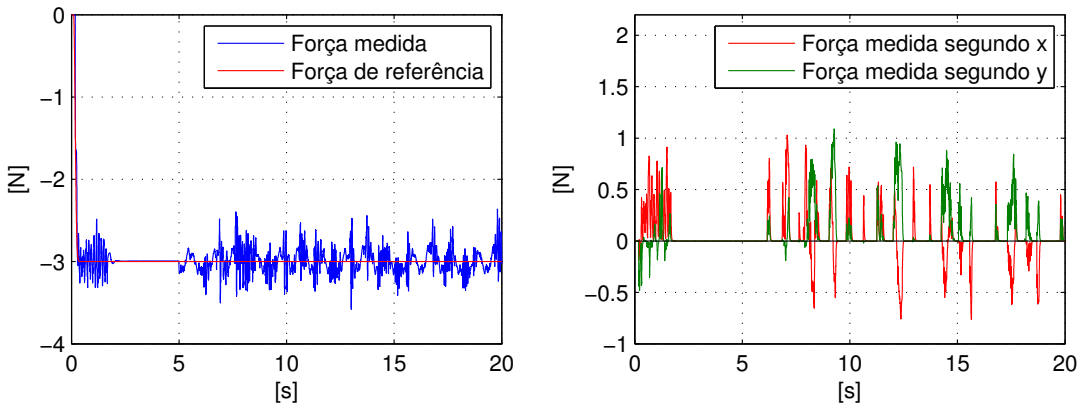


(a) Forças segundo z.



(b) Forças segundo x e y.

Figura 6.7: Compensação de uma perturbação sinusoidal, aplicada segundo o eixo z. A perturbação é iniciada em $t = 5$ s.



(a) Forças segundo z.

(b) Forças segundo x e y.

Figura 6.8: Compensação de uma perturbação composta por três sinusóides, aplicada segundo o eixo z. A perturbação é iniciada em $t = 5$ s.

Capítulo 7

Conclusão

Neste trabalho foi desenvolvido um simulador do manipulador 7-DOF WAM que permite reproduzir o seu comportamento dinâmico real quando o mesmo se encontra em contacto com outros objectos. O desenvolvimento deste simulador tem como principal objectivo permitir o teste e ajuste de arquitecturas de controlo, baseadas em *feedback* de força, antes das mesmas serem aplicadas a um robô real. Para além de um simulador físico, foi também criado um interface gráfico que garante um *feedback* visual durante a simulação.

Para a simulação das forças de contacto entre o manipulador robótico e os objectos presentes no ambiente virtual onde este se insere, foi proposto um novo método que recorre à voxelização dos modelos gráficos tridimensionais desses objectos. Devido ao processo de voxelização ser efectuado na GPU, é possível com este método obter tempos de execução mais reduzidos do que os verificados para outros métodos já existentes. Esta rapidez permite que o calculo as forças de contacto possa ser efectuado em cada iteração do algoritmo de controlo utilizado, permitindo assim que os objectos em causa possam alterar a sua estrutura durante a simulação (p. ex. expansão e/ou contracção).

Para a compensação de perturbações em força, resultantes dos movimentos fisiológicos existentes em cirurgias cardíacas, foi proposta uma arquitectura de controlo preditivo linear que recorre à linearização, por *feedback* não linear, do modelo dinâmico do manipulador. Esta arquitectura foi testada através do *software* de simulação desenvolvido para o efeito.

Os resultados experimentais mostraram que os tempos de execução do algoritmo de voxelização, embora reduzidos, não são suficientemente pequenos para que este possa ser implementado juntamente com o algoritmo de controlo utilizado. De forma a contornar este problema optou-se por uma voxelização parcial dos objectos, reduzindo assim o números de *voxel*s a considerar.

Pode ainda concluir-se que o controlador proposto permite, de forma significativa, a compensação de perturbações de forças externas, não sendo para isso necessário ter qualquer conhecimento prévio das mesmas. Muito possivelmente os parâmetros de controlo usados neste trabalho terão de ser reajustados para uma situação em que o controlador seja aplicado a um manipulador WAM real.

No âmbito deste projecto foi também desenvolvida uma arquitectura de *software/hardware* que permite a manipulação remota de uma ferramenta médica acoplada ao *end-effector* do robô WAM.

Trabalho futuro

Para trabalho futuro é sugerido que a implementação do método que calcula as forças de contacto seja feita, através de técnicas de GPGPU, inteiramente na placa gráfica e não apenas o processo de voxelização. Desta forma deixa de ser necessária a transferência de dados entre o processador gráfico e o processador central, o que se traduz numa diminuição do tempo de processamento. Além disso os tempos verificados para a descodificação dos *slicemaps* seriam significativamente reduzidos devido à enorme capacidade de processamento paralelo existente na GPU.

Uma vez que as perturbações de força utilizadas nas experiências tinham uma dinâmica relativamente baixa quando comparadas com o batimento cardíaco, é sugerido que o controlo seja testado com dados reais adquiridos, por exemplo, durante uma cirurgia.

Bibliografia

[1] T. Kanade, B. Davies, and C. N. Riviere, "Special Issue on Medical Robotics," *Proceedings of the IEEE*, vol. 94, no. 9, pp. 1649–1651, Sep. 2006.

[2] Y. Kwoh, J. Hou, and E. Jonckheere, "A robot with improved absolute positioning accuracy for CT guided stereotactic brain surgery," *, IEEE Transactions on*, vol. 35, no. 2, pp. 153–60, Feb. 1988.

[3] V. Falk, "Manual control and tracking–a human factor analysis relevant for beating heart surgery." *The Annals of Thoracic Surgery*, vol. 74, no. 2, pp. 624–628, 2002.

[4] I. Surgical, "da Vinci Surgical System," 2011.

[5] V. Falk, A. Diegler, T. Walther, R. Autschbach, and F. W. Mohr, "Developments in robotic cardiac surgery." *Current Opinion in Cardiology*, vol. 15, no. 6, pp. 378–387, 2000.

[6] R. Dzwonczyk, C. del Rio, B. Sun, R. Michler, and M. Howie, "Devices used to expose the posterior coronary artery in OPCABG surgery may cause ischemia," *Proceedings of the IEEE 31st Annual Northeast Bioengineering Conference, 2005.*, pp. 148–149, 2005.

[7] M. Newman, J. Kirchner, and B. Phillips-Bute, "Longitudinal assessment of neurocognitive function after coronary-artery bypass surgery," *New England Journal*, vol. 2, no. 6, pp. 689–91, Jun. 2001.

[8] Y. Nakamura, K. Kishi, and H. Kawakami, "Heartbeat synchronization for robotic cardiac surgery," in *Proceedings 2001 ICRA IEEE International Conference on Robotics and Automation Cat No01CH37164*, vol. 2. Ieee, 2001, pp. 2014–2019.

[9] W. Bachta, P. Renaud, E. Laroche, J. Gangloff, and A. Forgione, "Cardiolock: An active cardiac stabilizer. First in vivo experiments using a new robotized device," *Computer Aided Surgery*, vol. 13, no. 5, pp. 243–254, Jan. 2008.

[10] R. Richa, "Suivi 3D Robuste pour la Chirurgie Cardiaque Robotisée," *Chirurgie*, 2010.

[11] M. Kitagawa, A. M. Okamura, B. T. Bethea, V. L. Gott, and W. A. Baumgartner, "Analysis of Suture Manipulation Forces for Teleoperation with Force Feedback," *Mechanical Engineering*, vol. 2488, pp. 155–162, 2002.

[12] B. Cagneau, N. Zemiti, D. Bellot, and G. Morel, "Physiological Motion Compensation in Robotized Surgery using Force Feedback Control," *Proceedings 2007 IEEE International Conference on Robotics and Automation*, no. April, pp. 1881–1886, 2007.

[13] E. Eisemann and X. Décoret, "Fast Scene Voxelization and Applications," *ACM SIGGRAPH 2006 Sketches on - SIGGRAPH '06*, p. 8, 2006.

[14] Arduino, "Arduino Duemilanove," 2011. [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardDuemilanove

[15] M. W. Spong and M. Vidyasagar, "Robot Dynamics and Control," *Group*, p. 336, 1989.

[16] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

[17] R. Cortesão, B. Zenowich, R. Araújo, and W. Townsend, "Robotic Comanipulation With Active Impedance Control," *ASMEAFM 2009 World Conference on Innovative Virtual Reality*, pp. 129–135, 2009.

[18] J. De Jalon and E. Bayo, *Kinematic and dynamic simulation of multibody systems*. Springer Berlin, 1994.

[19] J. H. Mathews and K. D. Fink, *Numerical Methods Using MATLAB*. Prentice-Hall Inc., 2004.

[20] B. T. Inc., "Inertial Data for the WAM arm (Rev AA.00)," 2007.

[21] ——, "WAM™ Arm User's Guide (Rev AF.00)," 2007.

[22] M. Harbour, "Real-time POSIX: an Overview."

[23] B. Barney, "POSIX Threads Programming," 2011. [Online]. Available: https://computing.llnl.gov/tutorials/pthreads/

[24] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and R. Rowe, "Collision detection: A survey," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, vol. 72204. IEEE, Oct. 2007, pp. 4046–4051.

[25] P. M. Hubbard, "Interactive Collision Detection," 1993.

[26] C. Ericson, *Real-time collision detection*. Morgan Kaufmann, 2005.

[27] A. Mechelli, C. J. Price, K. J. Friston, and J. Ashburner, "Voxel-Based Morphometry Applications of the Human Brain : Methods and," *Imaging*, pp. 1–9, 2005.

[28] E. Garduño and G. Herman, "Applications of the Geometry of Digital Spaces to Medical Imaging," in *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on.* IEEE, 1998, pp. 244–245.

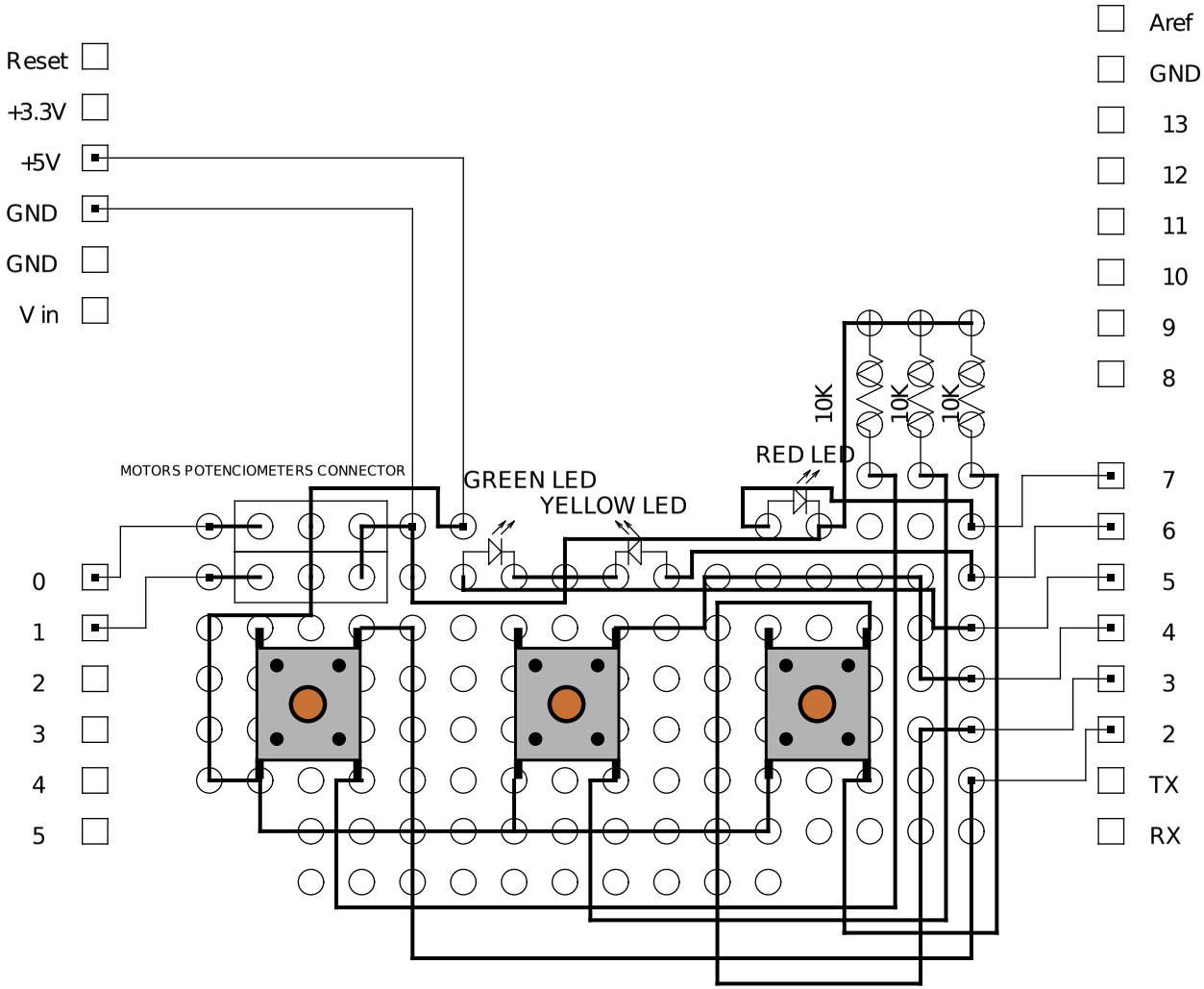
[29] E. Eisemann, "Single-pass GPU solid voxelization for real-time applications," *Proceedings of graphics interface 2008*, 2008.

[30] D. Eberly, "Least squares fitting of data," *Chapel Hill, NC: Magic Software*, pp. 1–10, 2000.

[31] W. Mathworld, "Point-Plane Distance," 2011. [Online]. Available: http://mathworld.wolfram.com/Point-PlaneDistance.html

[32] M. Dominici, P. Poignet, and E. Dombre, "Compensation of physiological motion using linear predictive force control," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1173–1178, Sep. 2008.

Anexo — Esquema do circuito electrónico implementado no desenvolvimento da ferramenta médica motorizada


Reset
+3.3V
+5V
GND
GND
V in
Aref
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX
RX
0
1
2
3
4
5
MOTORS POTENCIOMETERS CONNECTOR
GREEN LED
YELLOW LED
RED LED
10K
10K
10K