Felipe Bartelt de Assis Pessoa

Modelagem e Controle de um Robô para Cirurgias

Belo Horizonte

2022

Felipe Bartelt de Assis Pessoa

Modelagem e Controle de um Robô para Cirurgias

Monografia apresentada durante o Seminário dos Trabalhos de Conclusão do Curso de Graduação em Engenharia Elétrica da UFMG, como parte dos requisitos necessários à obtenção do título de Engenheiro Eletricista.

Universidade Federal de Minas Gerais – UFMG

Escola de Engenharia

Curso de Graduação em Engenharia Elétrica

Orientador: Prof. Vinicius Mariano Gonçalves

Belo Horizonte

2022

Agradecimentos

Primeiramente, agradeço aos meus pais, Renata e João Bosco, que sempre me apoiaram, incentivaram e participaram ativamente da minha vida. Por tudo que fizeram e fazem, tê-los-ei como exemplos para a vida. À minha namorada, Giovanna, além do amor gigante, encorajamento e companheirismo durante todos esses anos (desde os perrengues em ASDL até a entrega deste texto), você também escutou diversos raciocínios cuja base não compreendia e buscou entendê-los todas as vezes para me ajudar a formar uma conclusão ou uma nova ideia. Aos amigos de longa data, do famigerado Vip Nation, agradeço pelos inúmeros momentos de lazer e pelas muitas referências internas, quase impossíveis de explicar a outrem. Um agradecimento especial ao Holchan, pelo auxílio no tratamento dos modelos 3D, que me salvou muitas horas de trabalho.

Aos amigos do D.A.Icex que participaram de grande parte dessa jornada acadêmica. O entretenimento e ociosidade do D.A., fora as milhares de partidas de sinuca jogadas, diversos jogos de baralho aprendidos naquele ambiente, as conversas aleatórias com Leozin e Decão e as piadas do Phelps com certeza mitigaram o trajeto até esse ponto. Pra vocês, ainda digo: chegou a minha hora, ~~forrei~~ formei, vazei. Durante esses dois últimos períodos o GEC se tornou meu novo espaço de lazer e, claro, não poderia deixar de agradecer aos amigos desse novo ambiente, principalmente ao Chikin pela infinidade de Jack Power e ao Arthuzin por ser o melhor degustador de alimentos.

Ao professor Vinicius, tanto pela orientação deste trabalho quanto pela influência na paixão pela matemática e robótica. Obrigado pelas chamadas durante a pandemia com horas de duração para explicar conceitos matemáticos que fugiam do escopo das matérias, com certeza foram muito proveitosas e tem/terão grande valor. E, finalmente, àqueles que participaram indiretamente dessa trajetória, faxineiros, porteiros, funcionários dos serviços gerais, seguranças, etc. Se aproveitei todo o espaço do campus, a participação de vocês foi essencial.

"Quanto mais abstrata for a verdade que pretendes ensinar, maior deverá ser a arte em seduzir os sentimentos a favor de tal verdade."

(Friedrich Nietzsche)

Resumo

A robótica cirúrgica é uma área, atualmente, com ativo desenvolvimento e pesquisas. Entretanto, não há exemplos de sistemas cirúrgicos autônomos devido a alta complexidade e grande número de restrições que regem procedimentos cirúrgicos. Dessa forma, o intuito desse trabalho é implementar o sistema da Vinci Si, bastante conhecido no mercado, no simulador UAIBot, um simulador desenvolvido para fins didáticos pelo prof. Vinicius e por isso contém diversas funções úteis para implementações rápidas de uma alta gama de algoritmos de controle. Dessa forma, tornar-se-á possível a implementação e desenvolvimento de estratégias de controle no simulador para o da Vinci.

Para a implementação do robô, tornou-se necessário a estimativa dos seus parâmetros de Denavit-Hartenberg, uma vez que não há material oficial com essas informações, assim como a criação de primitivas de colisão para cálculo de distância em ambientes com obstáculos. A fim de testar implementação feita, propõe-se, por simulação, controlar um dos braços do robô para realizar a tarefa de atingir uma pose alvo em um ambiente com obstáculos complexos. Dada as restrições do problema, é de interesse o uso de algoritmos de controle baseados em programação quadrática que permitem minimizar uma função objetivo, atendendo-se a restrições de desigualdade quaisquer. Ainda, pretende-se considerar juntas passivas do da Vinci Si como atuadas, visando estudar a possibilidade de maiores graus de liberdade e, consequentemente, maior autonomia para o robô.

A estratégia de controle, com restrições de colisão, será comparada com uma outra formulação que não envolve restrições de colisão. Analisando-se o desempenho de ambas as estratégias, notou-se a incapacidade de evitar colisões quando essas restrições não são explícitas. Demonstrou-se a capacidade da otimização por programação quadrática convergir a uma pose alvo realizando, concomitante, evitamento de obstáculos. Observou-se que o gargalo para a utilização desses métodos no contexto de robótica cirúrgica é a construção do problema e não sua solução, devido ao tempo empregado para calcular diversas distâncias e concatenar todas as restrições em uma única representação.

Palavras-chaves: robótica cirúrgica; da Vinci Si; programação quadrática convexa; controle com restrição; manipuladores robóticos.

Abstract

Surgical robotics is an area, currently, with active development and research. However there are no examples of autonomous surgical systems due to the high complexity and large number of restrictions that govern surgical procedures. Thus, the intention of this work is to implement the da Vinci Si system, well known in the market, in the UAIBot simulator, a simulator developed for didactic purposes by Prof. Vinicius and therefore contains several useful functions for quick implementations of a high range of control algorithms. Thus, it will become possible to implement and develop control strategies in the simulator for the da Vinci.

For the implementation of the robot, it became necessary to estimate its Denavit-Hartenberg parameters, since there is no official material with this information, as well as to create collision primitives for distance calculation in environments with obstacles. In order to test the implementation made, it is proposed, by simulation, to control one of the arms of the robot to perform the task of reaching a target pose in an environment with complex obstacles. Given the constraints of the problem, it is of interest to use control algorithms based on quadratic programming that allow minimizing an objective function, given any inequality constraints. Furthermore, it is intended to consider passive joints of the da Vinci Si as actuated, aiming to study the possibility of greater degrees of freedom and, consequently, greater autonomy for the robot.

The control strategy, with collision constraints, will be compared with another formulation that does not involve collision constraints. Analyzing the performance of both strategies, it was noted the inability to avoid collisions when these constraints are not explicit. The ability of quadratic programming optimization to converge to a target pose while performing obstacle avoidance was demonstrated. It was observed that the bottleneck for the use of these methods in the context of surgical robotics is the construction of the problem and not its solution, due to the time used to calculate several distances and concatenate all constraints in a single representation.

Key-words: surgical robotics; da Vinci Si; convex quadratic programming; constrained control; robotic manipulators.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IA	Inteligência Artificial
DoA	*Degree of Autonomy* (grau de autonomia)
RAMIS	*Robot-Assisted Minimally Invasive Surgery*
MLP	Manipulador do Lado do Paciente
GDL	Graus de Liberdade
DH	Denavit-Hartenberg
MTH(s)	Matriz(es) de Transformação Homogênea
MAP	Método de Projeções Alternadas
QP	Programação Quadrática
AABB(s)	Axis-Aligned Bounding Box(es)

Lista de símbolos

P	Junta Prismática
R	Junta Rotativa
C	Subespaço convexo de $\mathbb{R}^n$. Conjunto de pontos na superfície de primitivas de colisão de um robô
D	Subespaço convexo de $\mathbb{R}^n$. Conjunto de pontos na superfície de um obstáculo
$x_{k,n}$	$n-$ésimo ponto pertencente ao conjunto K obtido por meio do método de projeções alternadas de Von Neumann
$P_K(x)$	Projeção ortogonal em K do ponto x
x_c^*	Ponto testemunha em C obtido pelo método de projeções alternadas de Von Neumann. É o ponto no robô que fornece a mínima distância entre robô e obstáculo
x_d^*	Ponto testemunha em D obtido pelo método de projeções alternadas de Von Neumann. É o ponto no obstáculo que fornece a mínima distância entre obstáculo e robô
t	Tempo
$q(t)$	Vetor de configurações
$\dot{q}(t)$	Vetor de velocidades das configurações
$J_{geo}(q)$	Jacobiana geométrica
$J_v(q)$	Jacobiana de velocidade linear, correspondente às três primeiras linhas da jacobiana geométrica
J_ω(q)	Jacobiana de velocidade angular, correspondente às três últimas linhas da jacobiana geométrica
$r(q)$	Função de tarefa
$J_r(q)$	Jacobiana de tarefa. Definida como a derivada parcial da função de tarefa $r(q)$ em função da configuração q
W, w	Matriz e vetor que definem restrições da forma $W\dot{q} \leq w$

$S(\cdot)$	Operador que mapeia um vetor a em sua respectiva matriz antissimétrica
$\Psi_{CD}(q)$	Distância mínima entre os objetos, subespaços, C e D
$\nabla\Psi_{CD}(q)$, J_Ψ	Gradiente ou jacobiana da distância $\Psi_{CD}(q)$
$\bar{b}$	Vetor das k distâncias mais críticas $\Psi_{C_iD_j}(q)$
$\bar{A}$	Matriz das k jacobianas mais críticas $J_{\Psi_{C_iD_j}}(q)$, as respectivas jacobianas dos elementos de $\bar{b}$
$\bar{b}_{\text{auto}}$	Vetor das k distâncias mais críticas $\Psi_{C_iD_j}(q)$, obtidas para a estimativa de auto-colisão
$\bar{A}_{\text{auto}}$	Matriz das k jacobianas mais críticas $J_{\Psi_{C_iD_j}}(q)$, as respectivas jacobianas dos elementos de $\bar{b}_{\text{auto}}$
$q_{\max}$	Vetor com os limites superiores para os valores das juntas
$q_{\max}$	Vetor com os limites inferiores para os valores das juntas
δ_{ij}	Diferença constante definida por $\delta_{ij} \equiv q_i(0) - q_j(0)$
$\tilde{q}_{ij}$	Matriz que implementa $q_i(t) - q_j(t)$
K_r	Ganho do controlador utilizado para realizar uma tarefa $r(q)$
μ, μ_{auto}	Parâmetro que regula a precisão do cálculo de distâncias, para a estimativa de colisões externa e auto-colisão, respectivamente
η, η_{auto}	Fator de amortecimento para a evitamento de colisão, para colisões externa e auto-colisão, respectivamente
ξ	Fator de amortecimento para restrição de limites de juntas
σ	Fator de amortecimento para restrição de juntas passivas

Sumário

1 Introdução

1.1 Cirurgia Robótica

Seres humanos são suscetíveis às variáveis não controláveis que afetam a consistência cirúrgica de médicos, como fadiga, questões físicas e mentais. Em Panesar et al. (2019), considera-se que esses fatores podem contribuir para a variabilidade na taxa de complicações em cirurgias, tornando-se portanto uma grande desvantagem humana quando comparado aos robôs cirúrgicos, que são insuscetíveis à fadiga, resistentes à tremores e com movimentos mais precisos e maior grau de liberdade. Para os autores, a combinação desses robôs com inteligência artificial (IA) reduziriam potenciais erros humanos, melhorariam o acesso a partes do corpo humano difíceis de acessar, além de reduzir o tempo de cirurgia.

Cirurgias com a participação, ativa ou passiva, de robôs permitem maior precisão, controle e evitação de dano aos tecidos (MOUSTRIS et al., 2011). Dessa forma, há hoje diversos robôs em desenvolvimento para a realização de cirurgias, com variados graus de autonomia, visando tornar cirurgias o menos invasivas possível. A norma IEC/TR 60601-4-1:2017(E) (2017) traz uma métrica para classificação de autonomia de robôs segundo um grau de autonomia (DoA) por meio de quatro funções correlacionadas à cognição: **gerar opções**, ou seja, baseado na tarefa de monitoramento de uma aplicação, o robô é capaz de formular opções para que se atinja um objetivo; **executar opções**, que é a habilidade executar uma opção selecionada, de forma ativa ou passiva; **monitorar opções**, a habilidade de coletar informações necessárias quanto ao equipamento, paciente, operador e ambiente, que são externas aos sinais de controle do robô; e **selecionar opções**, decidir a melhor opção dentre as várias geradas. A qualidade de gerar, executar e monitorar opções já é presente em diversos sistemas, porém a habilidade de decisão é o fator chave para que se alcance a completa autonomia, uma vez que as decisões hoje são feitas por cirurgiões experientes (HAIDEGGER, 2019).

A habilidade de decidir entre diversas opções retrata uma lacuna tecnológica, enquanto algumas cirurgias são rotineiras e previsíveis, outras têm alta variabilidade com possíveis complicações únicas ao paciente (HAIDEGGER, 2019). Dessa forma, os robôs atuais com maior grau de autonomia são desenvolvidos para tarefas específicas, enquanto os robôs mais genéricos tendem a ter menos autonomia. Assim, os tipos mais comuns de robôs cirúrgicos são os de telecirurgia mestre-escravo, comumente designados RAMIS (*Robot-Assisted Minimally Invasive Surgery*). Robôs desse tipo são completamente controlados pelo médico e, assim, herdam a segurança passada pelo mesmo. Robôs com mais autonomia, que ativamente participam no procedimento médico, são capazes de executar um procedimento planejado de forma 100% automática, desde que haja a aprovação de

um humano, sendo chamados de *dispositivos controlados de forma supervisória*. Os principais motivos para nenhum desses tipos de robôs terem alterado a forma como cirurgias são feitas hoje em dia, são devido à fatores econômicos (dado o alto custo desses equipamentos), sociais (a aceitação de pacientes e cirurgiões para essas tecnologias ainda não é suficiente) e legais (regulação desses produtos).

Os robôs cirúrgicos só serão completamente aceitos quando demonstrarem melhorias práticas para o paciente e em custo-benefício, o que só pode ser analisado após vários anos depois do procedimento médico (MOUSTRIS et al., 2011). Além disso, é muito importante que haja avanços no âmbito legal para que se possa presenciar cirurgias autônomas cotidianamente, uma vez que não se tem consenso quanto a responsabilidade caso um procedimento fracasse. Em O'Sullivan et al. (2019), apresenta-se que é possível implicar responsabilização social para os robôs autônomos, mas não responsabilidade legal e culpabilidade sob as leis atuais, porém, ao se manter um *log* de todas as ações do robô, seria possível justificar, em corte, eventuais problemas, promovendo assim uma forma de responsabilizar legalmente um robô. Ainda, consideram as leis atuais para carros autônomos análogas às necessárias leis para robôs cirúrgicos e, assim, há a necessidade de criar novas leis para sistemas completamente autônomos.

Um dos robôs cirúrgicos telemanipulados que tem autorização para realizar cirurgias no Estados Unidos e Europa é o da Vinci, uma vez que há sempre um cirurgião especializado controlando os movimentos do mesmo (O'SULLIVAN et al., 2019). Esse robô é o RAMIS mais famoso e que domina o mercado atualmente, desenvolvido pela Intuitive Surgical (HAIDEGGER, 2019). Há mais de 4400 sistemas da Vinci instalados em mais de 60 países, com mais de 5 milhões de procedimentos realizados por cirurgiões utilizando esse sistema, das quais mais de 875 000 foram realizados só em 2017 (DESAI et al., 2017). Esse sistema é indicado, segundo as regulamentações dos Estados Unidos, para cirurgias urológicas, laparoscopia geral, laparoscopia ginecológica, cirurgias de otorrinolaringologia transorais restritas a tumores benignos e malignos classificados como T1 e T2, cirurgias gerais toracoscópicas e procedimentos de cardiotomia assistidas por toracoscopia (DESAI et al., 2017).

A busca por cirurgias minimamente invasivas, incluindo o uso da laparoscopia, é dado pelo seguinte fato: o tecido enfermo ou danificado não é aparente, tornando-se necessário que o cirurgião percorra tecidos saudáveis em busca de seu alvo, tipicamente escondido dentro das estruturas do corpo, de tal forma que alcançá-lo pode levar a danos colaterais significativos (DESAI et al., 2017). O sistema da Vinci dá um passo além da laparoscopia, tornando a visualização e manipulação de tecidos o mais transparente e natural possível, objetivos contidos nos objetivos gerais de cirurgia robótica: estabilizando movimentos, redimensionando movimentos para incrementar a precisão, antecipar e avisar o usuário de possíveis eventos críticos (DESAI et al., 2017).

1.2 da Vinci

Figura 1 – da Vinci Si

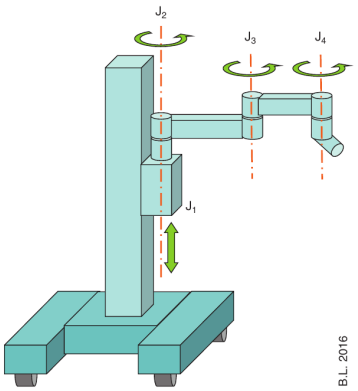
Fonte: (ROBOTS (IEEE Spectrum), 2018)

O sistema de robôs da Vinci, Figura 1, consiste em três partes, o console do cirurgião (manipuladores mestre), o carrinho do paciente (manipuladores escravos) e um sistema de imagem. O cirurgião controla os manipuladores escravos por meio de duas manivelas mestre no console, que é capaz de adquirir visões tridimensionais e ampliadas do campo de operação em alta resolução (BAI et al., 2019). Apesar do robô ser telemanipulado, tipicamente o console do cirurgião é alocado no mesmo ambiente que o paciente, dessa forma o cirurgião pode atuar rapidamente caso ocorra alguma falha no sistema. Os manipuladores do lado do paciente (MLP) correspondem a quatro braços robóticos que atuam em paralelo, sendo controlados alternadamente pelo cirurgião. Normalmente um dos braços é utilizado com um endoscópio, outro é utilizado para segurar tecidos e um terceiro para realizar ou remover suturas.

A estrutura dos MLP é constituída por três partes, a primeira consiste nos braços passivos, a segunda nos braços ativos e a terceira nos instrumentos *EndoWrist*. A parte passiva, onde há juntas não atuadas, serve para o melhor posicionamento dos braços do sistema e é manipulada pela própria equipe médica (LOMBARD; CÉRUSE; FUMAT, 2017). Para os braços 1, 2 e 4, conforme Figura 1, ela é composta de uma junta prismática (P) seguida de três juntas rotativas (R), enquanto o braço 3 é composto de uma junta rotativa a mais. A Figura 2 ilustra a composição PRRR dos braços 1, 2 e 4.

A parte atuada, Figura 3, é composta por quatro juntas rotativas e uma prismática na forma RR[RR]P, onde as juntas entre colchetes são juntas passivas. Ao contrário das juntas passivas anteriores, essas, apesar de não atuadas, replicam o movimento da segunda junta ativa, por meio de um sistema de cabos (LOMBARD; CÉRUSE; FUMAT, 2017),

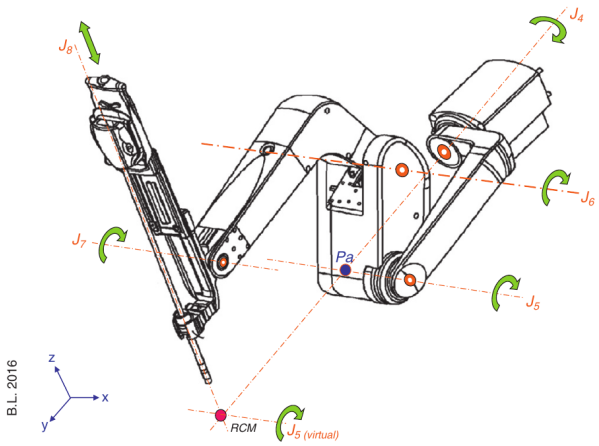
Figura 2 – Parte passiva do MLP referente ao braço 1



Fonte: (LOMBARD; CÉRUSE; FUMAT, 2017)

conforme a Figura 4. A parte atuada é comum a todos os braços. Dessa forma, a parte atuada tem 3 graus de liberdade (GDL).

Figura 3 – Parte atuada do MLP



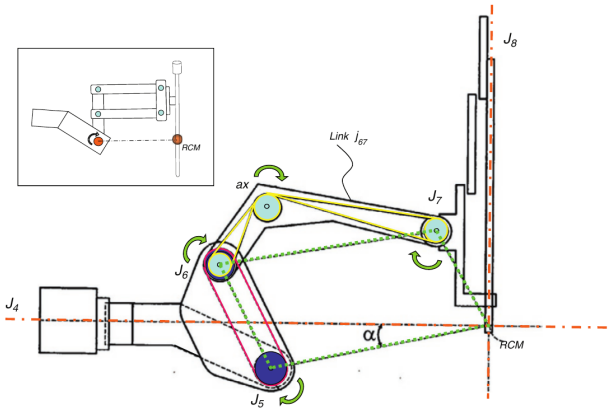
Fonte: (LOMBARD; CÉRUSE; FUMAT, 2017)

A última parte, que é o maior diferencial dos sistemas da Vinci, é composta por instrumentos *EndoWrist*, nome dado à coleção de instrumentos médicos que podem ser acoplados ao sistema. Fórceps, porta-agulha, endoscópios e grampeadores (Figura 5) são exemplos de instrumentos médicos *EndoWrist*, apresentados em diferentes diâmetros e comprimentos. Todos esses instrumentos apresentam 3 GDL na forma de juntas rotativas e, caso se considere o movimento de aperto das garras, em instrumentos como o fórceps, tem-se 4 GDL.

Dessa forma, o sistema da Vinci é da forma RR[RR]PRRR, onde os colchetes indicam juntas passivas, tendo-se então 6 GDL (FONTANELLI et al., 2017; BAI et al., 2019; DESAI et al., 2017), desprezando-se o possível movimento de pinça.

O espaço de trabalho do MLP do sistema de robôs da Vinci Si é altamente depen-

Figura 4 – Restrição de passividade para juntas da parte ativa do MLP



Fonte: (LOMBARD; CÉRUSE; FUMAT, 2017)

Figura 5 – Instrumentos *EndoWrist Staplers*

Fonte: (Intuitive Surgical, 2022)

dente da orientação de sua base, devido à disposição de seus braços, o que requer uma antecipação por parte da equipe da melhor posição para esse equipamento, de forma a evitar a necessidade de interrupções durante uma cirurgia para reposicioná-lo (DESAI et al., 2017). Essa deficiência foi corrigida na versão mais nova do sistema, da Vinci Xi (LOMBARD; CÉRUSE; FUMAT, 2017), porém ainda é interessante estudar as dificuldades implicadas pela disposição dos braços do sistema Si.

1.3 UAIBot

UAIBot (GONÇALVES, 2022b), Figura 6, é um simulador *web-based* de robótica, que utiliza a linguagem Python. O simulador foi desenvolvido pelo professor Vinicius Mariano Gonçalves e seus alunos, com o intuito de criar um simulador mais atraente para

Figura 6 – Logotipo do UAIBot

Fonte: (GONÇALVES, 2022b)

alunos de graduação. Devido ao simulador ser baseado em Python, linguagem com bastante demanda no mercado de trabalho atualmente, os alunos têm menos desinteresse em realizar simulações, pois estão de certa forma aprendendo uma linguagem de programação atrativa. Também é possível utilizar o simulador em navegadores, não sendo necessário baixar nenhum *software* ou arquivo, o que também age positivamente para não desmotivar os alunos a usar o simulador.

O UAIBot foi criado para ser um artifício didático, dessa forma o simulador é baixo-nível. Consequentemente, cabe ao usuário realizar todas as simulações necessárias com auxílio das funções implementadas. O simulador, até então, é focado em manipuladores robóticos seriais de cadeia aberta, apesar de existir suporte limitado para outros tipos de robôs.

Por meio de uma integração de Python com JavaScript, o simulador é capaz de criar animações para movimentos realizados por um robô. A criação de um cenário e sua união com o robô são realizadas por meio de funções presentes na biblioteca, assim como os movimentos realizados pelo robô. Cada movimento é tratado como um quadro separado da animação final do movimento. Essa animação é por fim criada com auxílio da biblioteca `Three.js` de JavaScript, que gera uma animação interativa como um arquivo HTML.

O simulador contém diversos robôs já configurados, porém é possível adicionar novos robôs. Para isso, é necessário obter um modelo 3D do robô em formato `.obj`. Os parâmetros de Denavit-Hartenberg (DH) devem ser conhecidos ou estimados, assim como o tipo da respectiva junta (prismática ou rotativa). Deve-se então, para cada elo, criar um objeto `Link` com seu respectivo modelo 3D, tipo de junta e os parâmetros de DH a ele associados. Feito isso, é possível controlar o robô.

Para fins de controle de manipuladores robóticos, o UAIBot conta com funções embutidas para cálculo de cinemática direta, cinemática inversa, cômputo de jacobianas analítica e geométrica, cálculo de funções de tarefa e outros métodos mais avançados. Dentre esses métodos, destaca-se a função `compute_dist`, que calcula a distância entre

cada *link* do robô e cada obstáculo presente no cenário, que pode ser consequentemente utilizada para evitamento de colisões.

Para que seja possível estimar distâncias entre elos e obstáculos, é necessário adicionar primitivas de colisão para cada *link* do robô. As primitivas existentes no simulador são esferas, caixas e cilindros. Elas devem ser acopladas aos respectivos links e, consequentemente, a matrizde transformação homogênea (MTH) que descreve a pose dessas superfícies é dependente dos parâmetros de DH associados ao *link* correspondente.

1.4 Objetivos

Um ambiente cirúrgico, apesar de não periculoso, é dotado de capacidade letal perante a possibilidade de negligências médicas ou projetista, dentro do contexto da robótica. Diversas operações são realizadas em uma área pequena do corpo, tornando-se de suma importância que um robô só interaja com essas áreas em específico, devendo-se evitar quaisquer colisões com outros tecidos. Dessa forma, as restrições em um problema de robótica cirúrgica são muitas, além de deveras complexas. Tal complexidade implica que diversos estudos devem ser feitos primeiramente em ambientes de simulação.

O simulador UAIBot é ideal para esse fim, uma vez que contém grande parte das funções base para utilizar diversos algoritmos de controle para manipuladores, tornando-se mais prático a implementação de algoritmos estado da arte em mais alto nível. Além disso, por utilizar linguagem Python, o ato de escrever códigos se torna mais direto.

O robô da Vinci não está presente no UAIBot, portanto, tomar-se-á como primeiro objetivo a sua implementação. Para isso, faz-se necessário encontrar um modelo 3D do robô e que se obtenha seus parâmetros de Denavit-Hartenberg. Devido a falta de referências oficiais quanto a esses parâmetros, esses serão estimados de forma visual por meio da interface do próprio simulador, assim como os limites de juntas. Far-se-á fundamental a implementação de primitivas de colisão para cada elo do robô, permitindo o evitamento de colisão por meio de funções de cálculo de distância intrínsecas ao simulador.

Com o modelo do robô obtido, um dos seus quatro braços será controlado, com objetivo de atingir uma pose alvo evitando, concomitantemente, colisões externas e autocolisões. A parte passiva do MLP será considerada atuada, buscando um maior número de graus de liberdade, além de permitir estudar a necessidade do posicionamento dos manipuladores ser feita pela equipe médica. O algoritmo de controle empregado terá como princípio a velocidade de decisão, portanto algoritmos de planejamento de movimento não serão utilizados.

2 Revisão Bibliográfica

2.1 Detecção de colisões

Detecção de colisões, comumente um gargalo para aplicações devido ao alto custo computacional, é um problema comum de engenharias e computação gráfica, tendo aplicações em, por exemplo, simulação cirúrgica, jogos de computador, modelagem molecular, planejamento de movimento (CHANG; WANG; KIM, 2010; DINAS; BAÑÓN, 2015). Existem diversos métodos para estimar colisões, sendo os mais comuns métodos analíticos, métodos geométricos, representações hierárquicas e métodos de otimização (DINAS; BAÑÓN, 2015). A técnica mais comum de ser utilizada para contornar esse problema é o de representações hierárquicas, mais especificamente hierarquia de volumes delimitadores (CHANG; WANG; KIM, 2010).

Robôs geralmente têm geometrias complexas e bastante diversas, dessa forma é custoso realizar estimativas de colisão baseados nas geometrias originais do robô. Dessa forma, é bastante útil utilizar uma hierarquia de volumes delimitadores. Essa técnica consiste em englobar os elos do robô com superfícies mais simples, como caixas, esferas, cilindros e elipsoides. São feitas diversas primitivas com níveis diferentes de retesamento, formando uma árvore hierárquica de primitivas que permite estimar colisões com maior precisão somente quando necessário, ou seja, quando há colisões para as primitivas mais largas (BRADSHAW; GARETH, 2002).

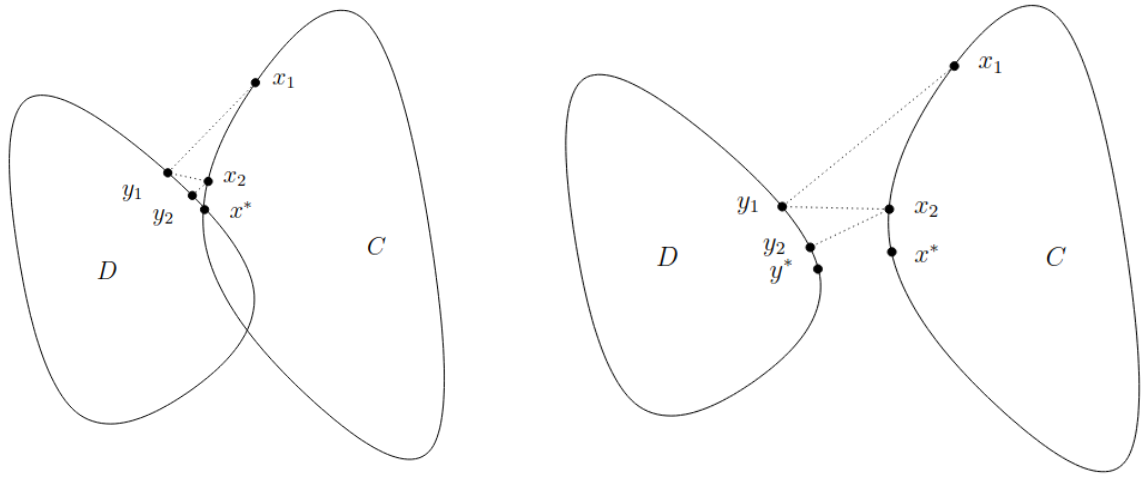
Para montar essa árvore, é necessário escolher qual tipo de primitiva utilizar. A escolha dentre as possíveis primitivas de colisão é dada pela maximização do volume englobado e a minimização do número de primitivas necessário para englobar o objeto como um todo (BRADSHAW; GARETH, 2002). Idealmente a geometria total do robô é reduzida a um conjunto menor e mais simples de superfícies. Consequentemente, a estimativa de colisões se torna muito mais eficiente (CHANG; WANG; KIM, 2010; DINAS; BAÑÓN, 2015).

As primitivas de colisão facilitam a estimativa de colisões. Dessa forma, ainda se torna necessário obter uma método que permita investigar a interseção de duas superfícies. Uma vez que uma colisão é um fenômeno que ocorre quando a distância entre dois objetos quaisquer é nula, basta tomar uma técnica genérica de medição de distâncias. Um método bastante útil para esse tipo de aplicação é o Método de Projeções Alternadas (MAP) de Von Neumann (ESCALANTE; RAYDAN, 2011).

No contexto de estimativa de colisões, o MAP consiste em tomar um ponto $x_{c,0}$ pertencente à superfície de um objeto C e realizar uma projeção ortogonal de $x_{c,0}$ em

Figura 7 – Resultados do Método de Projeções Alternadas de Von Neumann para situações com e sem colisão entre dois objetos

(a) Caso onde há colisão de dois objetos (b) Caso onde não há colisão de dois objetos



Para concordância entre texto e figura, tome as equivalências $x_n \equiv x_{c,n-1}$ e $y_n \equiv x_{d,n-1}\, \forall\, n \in \mathbb{N}^+$. Fonte: (BOYD; DATTORRO, 2003)

um objeto D, obtendo-se um ponto $x_{d,0}$, que corresponde ao ponto mais próximo em D de $x_{c,0}$. O ponto $x_{d,0}$ é reprojetado ortogonalmente em C resultando em um ponto $x_{c,1}$, o ponto mais próximo em C de $x_{d,0}$. Esse processo é repetido, realizando-se as projeções alternadamente, até que os pontos projetados convirjam ao ponto de interseção entre os objetos C e D, ou seja, ao ponto onde ocorre colisão, Figura 7a. Matematicamente, sejam C, D subespaços convexos de $\mathbb{R}^n$, $P_C(x)$ a projeção em C do ponto x, $P_D(x)$ a projeção em D do ponto x, então para cada $x \in \mathbb{R}^n$ vale (2.1).

$$\lim_{n\to\infty} \left(P_D P_C\right)^n x = P_{C\cap D} x \tag{2.1}$$

Também é possível utilizar o MAP como um estimador de distâncias entre objetos quaisquer, uma vez que, caso $C \cap D = \varnothing$, o processo converge para pontos, comumente chamados de pontos testemunha, x_c^*, x_d^*, Figura 7b, que correspondem ao par de pontos que minimiza a distância entre C e D (CHENEY; GOLDSTEIN, 1959). Na maioria dos casos, é mais computacionalmente profícuo estimar as projeções P_C e P_D do que a projeção $P_{C\cap D}$ (ESCALANTE; RAYDAN, 2011).

2.2 Controle com restrições

No contexto cirúrgico, o espaço de trabalho de um manipulador é bastante restrito, sendo necessário que o robô respeite restrições anatômicas do corpo humano e do procedimento médico em questão. Todos os tecidos ou regiões que não são alvos de um procedimento médico podem ser considerados obstáculos a serem evitados. Além disso,

durante um procedimento médico, é inevitável decisões sejam tomadas reativamente ou ao menos com elevada velocidade, o que impede o uso de algoritmos de planejamento de movimento, que podem tomar diversos minutos para que uma solução viável seja encontrada.

É sabido que para realizar o controle de um manipulador robótico com restrições de igualdade, pode-se utilizar a estimativa de matrizes pseudo-inversas como método de cinemática inversa (SICILIANO et al., 2009), para o qual é possível adicionar restrições com diferentes prioridades por meio da projeção no espaço nulo (SICILIANO; SLOTINE, 1991). Entretanto, para evitamento de colisões, não é necessário se manter uma distância fixa entre o manipulador e um obstáculo qualquer, bastando apenas manter uma distância no mínimo maior que uma distância de segurança arbitrária. Esse também é o caso para limites de juntas, uma vez que não se deseja manter as configurações de junta iguais a algum valor, mas sim dentro de limites inferiores e superiores.

Restrições mais flexíveis como as supracitadas são descritas por meio de inequações, que geralmente não podem ser resolvidas de forma eficiente por meio de pseudo-inversas, sendo necessário o uso de algoritmos de programação quadrática (QP) convexa (KANOUN; LAMIRAUX; WIEBER, 2011). Para o controle com esse tipo de restrição, pode-se utilizar tanto métodos que implicam campos potenciais para abranger as restrições (KHATIB, 1985) quanto métodos que utilizam otimização diretamente para atender às restrições (FAVERJON; TOURNASSOUD, 1987). Controle com restrições de desigualdade diferencial é conhecido na literatura de teoria de controle pela nomeação *Control Barrier Function* (AMES et al., 2019). Dessa forma, a formulação por programação quadrática tem, também, respaldo na literatura de *Control Barrier Functions*. Pode-se então definir o problema de controle de robôs com restrições de desigualdade por (2.2).

$$\begin{aligned} &\min_{\dot{q}} \|J_r\dot{q} + K_r r\|_2^2 \\ &\text{sujeito a } W\dot{q} \leq w, \end{aligned} \tag{2.2}$$

onde $q \equiv q(t) \in \mathbb{R}^n$ é o vetor de configurações do robô, $r \equiv r(q) : \mathbb{R}^n \mapsto \mathbb{R}^m$ é uma função de tarefa, $J_r \equiv J_r(q) = \frac{\partial r(q)}{\partial q} \in \mathbb{R}^{m\times n}$ é a matriz jacobiana de tarefa e $K_r \in (0, \infty)$ é um ganho. $W \equiv W(q) \in \mathbb{R}^{\ell\times n}$ e $w \equiv w(q) \in \mathbb{R}^\ell$ definem as restrições de desigualdade.

Tomando-se um contexto onde as restrições são unicamente de colisão, a solução de (2.2) consiste em encontrar uma solução para o problema de programação quadrática convexa sobre um politopo induzido pelas restrições impostas. Esse politopo limita as velocidades do robô em direção aos obstáculos, apesar de não restringir velocidades ortogonais a essa direção. Dessa forma, a tarefa é satisfeita e, concomitantemente, evita-se colisões com obstáculos.

Com a utilização de QP ainda é possível considerar múltiplas restrições com diferentes ordens de prioridade (KANOUN; LAMIRAUX; WIEBER, 2011; ESCANDE; MANSARD; WIEBER, 2014); garantir o atendimento às restrições minimizando o número de juntas atuadas simultaneamente de um robô altamente redundante, reduzindo-se o custo do movimento (GONÇALVES et al., 2016); assim como projetar um controlador estável, em malha fechada, por construção que atenda a restrições (GONÇALVES et al., 2020). O controle de manipuladores por meio de QP também é eficaz para robôs cirúrgicos (MARINHO et al., 2019a; MARINHO et al., 2019b), sendo também possível que as restrições sejam dinâmicas.

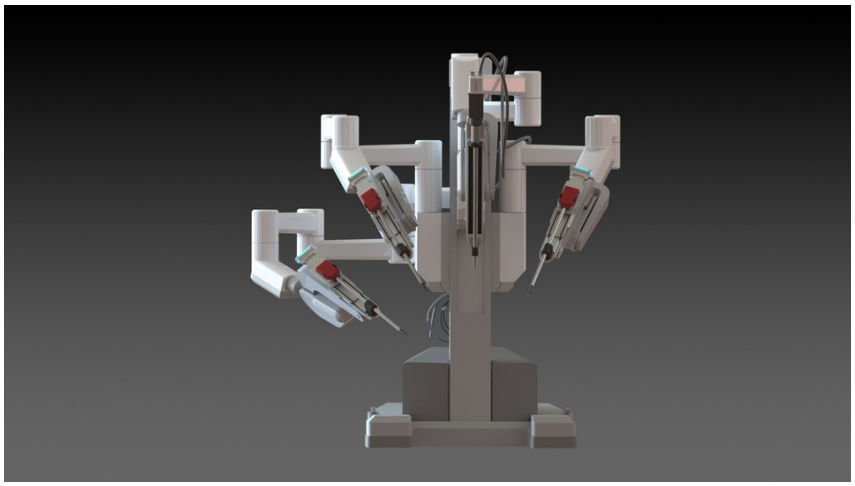
3 Metodologia

Para este trabalho, tomou-se considerações que extrapolam o funcionamento real do robô cirúrgico real. É de interesse avaliar a possibilidade de atuar sobre as juntas da parte passiva do robô da Vinci, uma vez que essa análise pode levantar possibilidades de melhoria em projetos futuros de robôs cirúrgicos e no planejamento médico da disposição desses braços passivos antes de uma cirurgia com o da Vinci Si. Dessa forma, todas essas juntas serão consideradas atuadas, aumentando-se 3 GDL.

Ainda, o modelo 3D obtido não consta com instrumentos *EndoWrist* e, dessa forma, perde-se 3 GDL. Consequentemente, o robô considerado tem 7 GDL da forma PRRRRR[RR]P.

3.1 Implementação do da Vinci no UAIbot

Figura 8 – Modelo 3D do da Vinci utilizado



Fonte: (OKAN, 2021)

Primeiramente, tomou-se o modelo do robô da Vinci em Solid Works (OKAN, 2021), Figura 8, cujas peças foram convertidas para arquivos do tipo `.obj` por meio do *software* 3ds Max. Obteve-se assim um conjunto de 63 peças que compõe o modelo do robô. Então, analisou-se quais peças compunham cada *link* dos braços do robô. Assim, agrupou-se cada conjunto de *links*, referentes aos diferentes braços, no simulador UAIBot (GONÇALVES, 2022b), assim como a base do robô.

Cada um dos braços é tratado como um manipulador robótico independente, da classe `Robot`, e a base é tratada como um objeto rígido, classe `RigidObject`, dessa

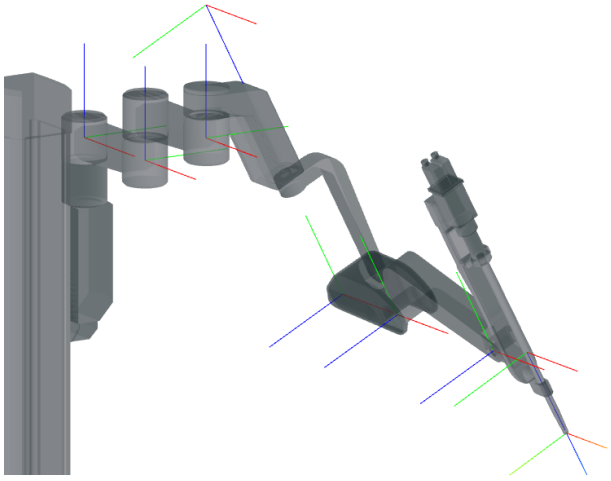
forma o sistema completo é o conjunto das quatro entidades `Robot`, junto da entidade `RigidObject`.

Para que as peças se movam de forma correta, é necessário adicionar suas respectivas matrizes de transformação homogênea, não em relação ao referencial do mundo, mas em relação ao *frame* anterior. Assim, tornou-se necessário estimar os parâmetros de DH dos manipuladores.

3.1.1 Estimativa dos parâmetros de Denavit-Hartenberg

Devido à falta de informações quanto aos parâmetros de DH e dimensões de cada parte do robô nos manuais da *Intuitive Surgical*, a estimativa dos parâmetros foi feita de forma visual com base no modelo 3D obtido, assumindo-se assim que o autor do modelo original o criou com proporções reais. Para isso, variou-se os valores dos parâmetros de DH, por meio do método da bisseção e avaliou-se o comportamento das juntas frente a variações de sua configuração. Uma vez que os *links* se movessem sem haver desacoplamento perceptível das peças, isso é, os *links* rotacionassem centrados no eixo desejado, considerou-se como correto os parâmetros obtidos. Na Figura 9 é possível ver os *frames* obtidos para o braço 4 seguindo esse procedimento.

Figura 9 – *Frames* de DH obtidos para o braço 4 do sistema da Vinci Si.



A equivalência de cores e eixos é: Vermelho – eixo **x**, Verde – eixo **y**, Azul – eixo **z**. Fonte: Elaboração própria

O simulador não permite acrescentar rotações iniciais no eixo **z** dos *frames* de DH de juntas rotativas, ou seja, não é possível modificar o parâmetro θ de juntas rotativas. Assim, tornou-se necessário a rotação das peças, modelos que compõe os *links*, para que os eixos **x** ficassem conforme a convenção de DH. Consequentemente, a estimativa dos parâmetros de DH depende da configuração inicial do da Vinci.

Existem peças diferentes na composição dos braços do da Vinci e, para as peças idênticas, há diferenças em suas disposições. Além disso, a pose dos braços no modelo

original são bastantes distintas entre si. Esses fatos, dada a dependência da configuração inicial, implicaram em parâmetros de DH diferentes para os braços do robô, exceto para o braço 1, que é idêntico ao 2 e, dessa forma, apenas foi necessário pré multiplicar o braço 2 por uma MTH para se implementar o braço 1. Devido a essas dificuldades, a obtenção dos parâmetros de DH requereu a maior parte do tempo empregado para a realização deste trabalho.

3.1.2 Estimativa dos limites de junta

O procedimento para a estimativa dos limites de juntas é análogo ao tomado para a estimativa dos parâmetros de Denavit-Hartenberg. Variando-se iterativamente cada junta de forma individual, observou-se para quais valores de configuração ocorreria auto-colisão entre elos subsequentes. Para as juntas prismáticas, também se observou quais valores aparentavam ferir os limites físicos do robô.

Especificamente para juntas rotativas onde não foi observado situações de auto-colisão, tomou-se limites de junta $(-4\pi, 4\pi)$, uma vez que não é possível tomá-las como ilimitadas e esse intervalo abrange rotações suficientes para movimentos complexos.

3.1.3 Primitivas de colisão

Durante uma cirurgia realizada com o sistema da Vinci, existem diversos elementos que podem ser considerados obstáculos sob a perspectiva de controle. Ao se controlar um braço específico do sistema, tem-se a base, onde todos os braços estão acoplados, e todos os demais braços como obstáculos, ou seja, o conjunto de configurações que gera colisão com esses elementos deve ser proibido. A mesa de cirurgia e regiões do paciente que não sofrerão algum procedimento também são consideradas obstáculos.

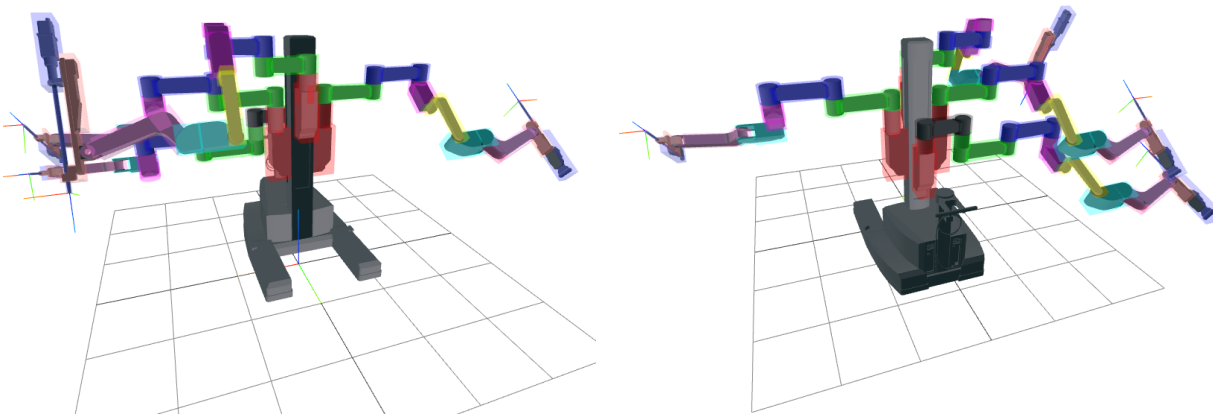
A fim de realizar o controle de um robô com evitamento de obstáculos, é necessário o cálculo da distância entre ambos. Uma vez que a geometria do sistema da Vinci não é simples, pode-se tomar a abordagem de cobrir o robô com primitivas de colisão, permitindo-se a estimativa de distância entre superfícies com geometria mais simples, como paralelepípedos, esferas e cilindros.

Encobriu-se cada *link* do robô com um conjunto de primitivas de colisão de tal forma que todo o elo ficasse totalmente coberto pelo conjunto. Ainda, quando possível, tomou-se primitivas levemente maiores que os *links*, de forma a adicionar uma distância de segurança intrínseca para quaisquer cálculos de distância. A escolha de geometrias deve ser tomada como a mais simples possível tal que revista grande parte do elo. Deve-se também minimizar a quantidade de primitivas que formam cada conjunto, de forma a reduzir a complexidade dos cálculos de distância.

Ao final, obteve-se 31 primitivas de colisão para os braços 3 e 4 e 28 primitivas

Figura 10 – Primitivas de colisão utilizadas para o da Vinci

(a) Frente (b) Trás



As cores indicam um conjunto de primitivas relativo a cada *link* de um braço. Elos análogos entre braços são representados com as mesmas cores. Fonte: Elaboração própria

para os braços 1 e 2, utilizando-se somente cilindros e paralelepípedos como superfícies. Pode-se observar esse resultado pela Figura 10, onde o conjunto de primitivas de colisão de cada *link* tem uma cor única. Ainda, *links* análogos têm as mesmas cores.

3.2 Controle

3.2.1 Função de tarefa

Para se utilizar algoritmos de otimização de QP convexa para solucionar o problema da forma (2.2), é necessário definir uma função de tarefa $r(q)$, onde $q \in \mathbb{R}^n$ é a configuração do robô. Tomando-se como objetivo o posicionamento ideal do efetuador de um dos braços do da Vinci para um procedimento cirúrgico, a partir de uma configuração inicial qualquer, pode-se definir a tarefa como alcançar uma pose alvo. Pode-se então definir a função de tarefa por (3.1) (GONÇALVES, 2022a).

$$r(q) = \begin{pmatrix} p_e(q) - p_{tg} \\ 1 - x_{tg}^T x_e(q) \\ 1 - y_{tg}^T y_e(q) \\ 1 - z_{tg}^T z_e(q) \end{pmatrix} \in \mathbb{R}^6, \tag{3.1}$$

onde $p_e(q) \in \mathbb{R}^3$ é a posição do centro do referencial do efetuador, $p_{tg} \in \mathbb{R}^3$ é a posição do centro do alvo, $x_e(q), y_e(q), z_e(q) \in \mathbb{R}^3$ são os eixos x, y, z do referencial do efetuador e $x_{tg}, y_{tg}, z_{tg} \in \mathbb{R}^3$ são os eixos x, y, z do alvo. Todos os elementos são medidos no referencial do mundo.

A jacobiana $J_r(q)$ da tarefa $r(q)$ pode ser calculada como (3.2) (GONÇALVES, 2022a).

$$J_r(q) = \frac{\partial r(q)}{\partial q} = \begin{bmatrix} J_p(q) \\ x_{tg}^T S\left(x_e\left(q\right)\right) J_\omega(q) \\ y_{tg}^T S(y_e(q)) J_\omega(q) \\ z_{tg}^T S(z_e(q)) J_\omega(q) \end{bmatrix} \in \mathbb{R}^{6\times n}, \tag{3.2}$$

onde $J_p(q) \in \mathbb{R}^{3\times n}$ é a jacobiana de posição do efetuador, que são as três primeiras linhas de sua jacobiana geométrica $J_{geo} \in \mathbb{R}^{6\times n}$, $J_\omega(q) \in \mathbb{R}^{3\times n}$ é a jacobiana de orientação do efetuador, que são as três últimas linhas de J_{geo}. S é um operador definido por: seja $a = \begin{pmatrix} a_x & a_y & a_z \end{pmatrix}^T$ um vetor tridimensional, então o operador $S : \mathbb{R}^3 \mapsto \mathbb{R}^{3\times 3}$ é dado por (3.3)

$$S(a) = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \tag{3.3}$$

Tanto a função de tarefa $r(q)$ quanto sua jacobiana $J_r(q)$ podem ser calculadas pela função intrínseca do UAIBot `task_function`, que é um método da classe `Robot`.

3.2.2 Função de distância e sua jacobiana

Para o evitamento de colisão, é necessário a definição de uma função de distância e sua respectiva jacobiana (MARINHO et al., 2019b). Tomando-se (2.1), sabe-se que, para quaisquer objetos C e D, o método converge a pontos testemunha, x_c^*, x_d^* tais que a distância entre eles corresponde a mínima distância entre C e D. Consequentemente, seja C o conjunto de pontos na superfície de um elo do robô e D o conjunto de pontos na superfície de um obstáculo. Tomando-se $\Psi_{CD}(q)$ como a distância mínima entre os objetos C e D e $J_\Psi \equiv J_{\Psi_{CD}}(q)$ como a jacobiana da distância entre os objetos, tem-se:

$$\Psi_{CD}(q) = \|x_c^*(q) - x_d^*(q)\|$$

$$\implies \nabla\Psi_{CD}(q) = J_\Psi = \begin{pmatrix} \frac{\partial\Psi_{CD}(q)}{\partial q_1} \\ \frac{\partial\Psi_{CD}(q)}{\partial q_2} \\ \vdots \\ \frac{\partial\Psi_{CD}(q)}{\partial q_n} \end{pmatrix},$$

É possível demonstrar que J_Ψ é dada por

$$J_\Psi = \left(J_{x_c^*} - J_{x_d^*}\right)^T \cdot \frac{x_c^*(q) - x_d^*(q)}{\Psi_{CD}(q)}, \tag{3.4}$$

onde $J_{x_c^*}$ é definido da seguinte forma: dado que o ponto testemunha x_c^* pertence a um elo do robô, seja p_o a posição do centro do referencial fixado a esse elo, v_{p_o} a velocidade linear

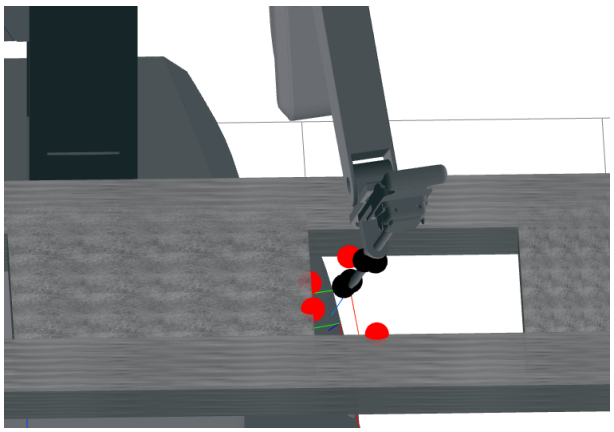
desse referencial e ω_{p_o} a velocidade angular desse referencial, J_{v,p_o} e J_{ω,p_o} as jacobianas de velocidade linear e angular, respectivamente, do referencial fixado em p_o. Ainda, J_{v,p_o} corresponde às três primeiras linhas da jacobiana geométrica J_{geo} e J_{ω,p_o} corresponde às suas três últimas linhas. Então $J_{x_c^*}$ é dado por (3.5).

$$\begin{aligned} \frac{dx_c^*(q)}{dt} &= v_{p_o} + \omega_{p_o} \times (x_c^* - p_o) \\ &= \frac{\partial x_c^*(q)}{\partial q}\dot{q} = J_{v,p_o}(q)\dot{q} - (x_c^* - p_o) \times \omega_{p_o} \\ &= J_{v,p_o}(q)\dot{q} - S(x_c^* - p_o)J_{\omega,p_o}(q)\dot{q} \\ \therefore \frac{\partial x_c^*(q)}{\partial q} &= J_{x_c^*} = J_{v,p_o}(q) + S(p_o - x_c^*)J_{\omega,p_o}(q) \end{aligned} \tag{3.5}$$

Para $J_{x_d^*}$ segue definição análoga, dado x_d^* um ponto pertencente à superfície de um obstáculo e p_o o centro do referencial à ele fixado. Para obstáculos estáticos, $J_{x_d^*}$ é nulo.

A função de cálculo de distâncias do UAIBot já calcula distâncias por meio do algoritmo de projeções alternadas de Von Neumann. Pode-se ver os pontos testemunha obtidos pelo MAP no UAIBot pela Figura 11, na qual as esferas pretas representam os pontos testemunha x_c^*, ponto no manipulador mais próximo do obstáculo e as esferas vermelhas indicam os pontos testemunha x_d^*, ponto no obstáculo mais próximo do manipulador. Na Figura 11 apenas são mostrados os quatro pares de pontos testemunha mais de menor distância, de forma a facilitar a visualização.

Figura 11 – Quatro pares de pontos testemunha obtidos na estimativa de colisão entre da Vinci e obstáculo em uma das iterações



Esferas pretas indicam os pontos testemunha x_c^*, ponto no manipulador mais próximo do obstáculo. Esferas vermelhas indicam os pontos testemunha x_d^*, ponto no obstáculo mais próximo do manipulador. Fonte: Elaboração Própria

3.2.3 Restrições de colisão externa

Dado que cada elo do robô é englobado por primitivas de colisão, então C representa uma primitiva de colisão no manipulador e D um obstáculo qualquer. Pode-se então definir restrições de colisão ao se tomar a distância e sua jacobiana de todos os pares (C_p, D_l), supondo-se p primitivas de colisão e l obstáculos.

Deseja-se que a distância entre cada primitiva de colisão do robô C_i e cada obstáculo D_j seja maior ou igual a uma distância de segurança d_{safe}, dessa forma:

$$\begin{aligned} \frac{d}{dt}\Psi_{C_iD_j} &\geq -\eta\left(\Psi_{C_iD_j} - d_{\text{safe}}\right) \\ \implies J^T_{\Psi_{C_iD_j}}\dot{q} &\geq -\eta\left(\Psi_{C_iD_j} - d_{\text{safe}}\right), \end{aligned}$$

onde η é um coeficiente de amortecimento (KANOUN; LAMIRAUX; WIEBER, 2011).

Concatenando-se todas as distâncias e jacobianas, conforme (3.6), define-se as restrições de colisão para todas as primitivas e obstáculos presentes por (3.7).

$$b = \begin{bmatrix} \Psi_{C_1D_1} \\ \Psi_{C_2D_1} \\ \vdots \\ \Psi_{C_pD_1} \\ \Psi_{C_1D_2} \\ \vdots \\ \Psi_{C_pD_l} \end{bmatrix} \in \mathbb{R}^{p\cdot l}, \quad A = \begin{bmatrix} J^T_{\Psi_{C_1D_1}} \\ J^T_{\Psi_{C_2D_1}} \\ \vdots \\ J^T_{\Psi_{C_pD_1}} \\ J^T_{\Psi_{C_1D_2}} \\ \vdots \\ J^T_{\Psi_{C_pD_l}} \end{bmatrix} \in \mathbb{R}^{(p\cdot l)\times n} \tag{3.6}$$

$$-A\dot{q} \leq \eta\left(b - d_{\text{safe}}\right), \tag{3.7}$$

O custo computacional para se calcular todas as distâncias e jacobianas, conforme (3.6), é alto. Dessa forma, basta tomar k distâncias mais críticas e suas respectivas jacobianas. Isso é, toma-se um vetor $\bar{b} \in \mathbb{R}^k$ que contém apenas as linhas de b que apresentam as k menores distâncias $\Psi_{C_iD_j}$. Suas respectivas jacobianas são armazenadas em uma matriz $\bar{A} \in \mathbb{R}^{k\times n}$. As restrições se reduzem a (3.8).

$$-\bar{A}\dot{q} \leq \eta\left(\bar{b} - d_{\text{safe}}\right), \tag{3.8}$$

O processo de se tomar k menores distâncias é facilmente obtido pela hierarquia de primitivas de colisão. A função do UAIBot para cálculo de distâncias automaticamente cria uma hierarquia, para todo cálculo de distância entre uma primitiva definida e um obstáculo qualquer, a função primeiramente engloba tanto a primitiva quanto o obstáculo com Axis-Aligned Bounding Boxes (AABBs) (BRADSHAW; GARETH, 2002; DINAS;

BAÑÓN, 2015). Calcular distâncias entre AABBs é muito mais rápido, portanto essa distância é calculada primeiro. Caso a distância entre ambas AABBs seja maior ou igual a μ, não são calculadas distâncias mais precisas e não se retorna $\Psi_{C_iD_j}$ ou $J_{\Psi_{C_iD_j}}$. Caso contrário, o cálculo de distâncias é tomado com base nas primitivas definidas e no formato real do obstáculo. Efetivamente, a função retorna apenas distâncias críticas, que foram calculadas a partir das primitivas definidas anteriormente. Tomou-se $\mu = 0{,}5\,\text{m}$ para todas as aplicações da função de distância em colisões externas.

3.2.4 Restrições de auto-colisão

O processo para a estimativa de auto-colisão é análogo ao processo para computar colisões externas. Considera-se como D as primitivas de colisão dos elos do robô considerados obstáculos. Como os elos são móveis, tem-se $J_{x_d^*}$ não nulo. Uma vez que as restrições de limite de junta sejam respeitadas, não é necessário computar a possibilidade de auto-colisão entre elos subsequentes. O cálculo de distâncias, com todas as considerações feitas, já é implementado pela função `compute_dist_auto` do UAIBot. Dessa forma, a restrição análoga para auto-colisão pode ser expressa por (3.9). Tomou-se para o cálculo de auto-colisões o parâmetro $\mu_{\text{auto}} = 0{,}25$ como decisor para efetuar cálculos de distância mais precisos.

$$-\bar{A}_{\text{auto}}\dot{q} \leq \eta_{\text{auto}} \left(\bar{b}_{\text{auto}} - d_{\text{safe}}\right), \tag{3.9}$$

onde, devido a hierarquia de primitivas, $\bar{A}_{\text{auto}} \in \mathbb{R}^{g\times n}, \bar{b}_{\text{auto}} \in \mathbb{R}^g$.

3.2.5 Restrições de limites de juntas

Para impor limites às juntas do manipulador, basta impor que a velocidade de junta $\dot{q}$ satisfaça (3.10) e, portanto, ambas as inequações de (3.11).

$$\xi\left(q_{\min} - q\right) \leq I\dot{q} \leq \xi\left(q_{\max} - q\right) \tag{3.10}$$

$$\implies \begin{cases} I\dot{q} \leq \xi\left(q_{\max} - q\right) \\ -I\dot{q} \leq -\xi\left(q_{\min} - q\right) \end{cases}, \tag{3.11}$$

onde $q_{\min}, q_{\max} \in \mathbb{R}^n$ são, respectivamente, vetores que contém os valores mínimos e máximos de cada junta, ξ um fator de amortecimento e I a matriz identidade de tamanho $n \times n$.

3.2.6 Restrições de juntas passivas

Para garantir a restrição de juntas passivas para a parte ativa do MLP, basta garantir que a configuração das juntas q_7 e q_8 mantenham uma diferença constante com a configuração q_6, sendo o manipulador representado pela configuração $(q_1, q_2, \ldots, q_9)$. Sejam $q_{6,0}, q_{7,0}$ e $q_{8,0}$ as configurações iniciais das juntas q_6, q_7 e q_8, respectivamente e

tomando $\delta_{76} \equiv q_{7,0} - q_{6,0}$ e $\delta_{86} \equiv q_{8,0} - q_{6,0}$, basta que a diferença entre as velocidades de junta atendam a (3.12) e (3.13).

$$\dot{q}_7 - \dot{q}_6 = -\sigma\left(q_7 - q_6 - \delta_{76}\right) \tag{3.12}$$

$$\dot{q}_8 - \dot{q}_6 = -\sigma\left(q_8 - q_6 - \delta_{86}\right), \tag{3.13}$$

onde σ é um fator de amortecimento.

Tomando as definições matriciais (3.14) e (3.15), pode-se definir a restrição de igualdade matricial (3.16).

$$\widetilde{q}_{76} \equiv \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix} \tag{3.14}$$

$$\widetilde{q}_{86} \equiv \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \tag{3.15}$$

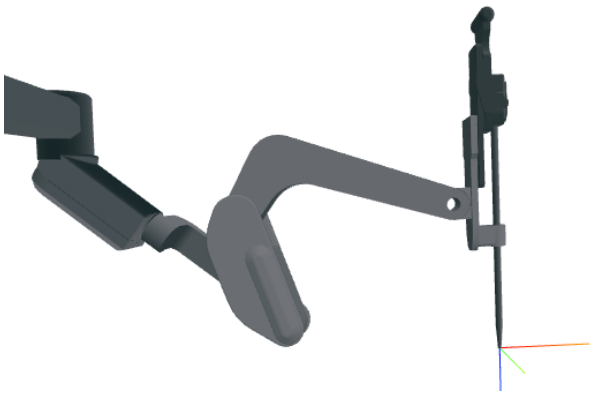
$$\begin{bmatrix} \widetilde{q}_{76} \\ \widetilde{q}_{86} \end{bmatrix} \dot{q} = \begin{bmatrix} -\sigma\left(q_7 - q_6 - \delta_{76}\right) \\ -\sigma\left(q_8 - q_6 - \delta_{86}\right) \end{bmatrix} \tag{3.16}$$

Uma vez que (2.2) toma restrições de desigualdade, define-se as restrições de igualdade anteriores por meio de duas restrições, assim as restrições de junta passiva utilizadas formam (3.17).

$$\begin{bmatrix} \widetilde{q}_{76} \\ \widetilde{q}_{86} \\ -\widetilde{q}_{76} \\ -\widetilde{q}_{86} \end{bmatrix} \dot{q} \leq \begin{bmatrix} -\sigma\left(q_7 - q_6 - \delta_{76}\right) \\ -\sigma\left(q_8 - q_6 - \delta_{86}\right) \\ \sigma\left(q_7 - q_6 - \delta_{76}\right) \\ \sigma\left(q_8 - q_6 - \delta_{86}\right) \end{bmatrix} \tag{3.17}$$

Para obter valores de δ_{76} e δ_{86} próximos aos reais, alterou-se a configuração inicial do manipulador para atingir a situação apresentada na Figura 12. Essa configuração foi obtida ao se tentar aproximar a configuração do manipulador da configuração apresentada em Figura 4. Assim, permite-se definir δ_{76} e δ_{86} avaliando-se essas configurações iniciais.

Figura 12 – Configuração inicial das últimas juntas para garantir a restrição de juntas passivas



Fonte: Elaboração Própria

3.2.7 Conjunto total de restrições e constantes adotadas

Todas as restrições enunciadas podem ser tratadas simultaneamente por meio de sua concatenação. Concatenando-se todas as restrições em uma matriz W e vetor w, tem-se as representações (3.18).

$$W = \begin{bmatrix} -\bar{A} \\ -\bar{A}_{\text{auto}} \\ I_{n\times n} \\ -I_{n\times n} \\ \widetilde{q}_{76} \\ \widetilde{q}_{86} \\ -\widetilde{q}_{76} \\ -\widetilde{q}_{86} \end{bmatrix} \in \mathbb{R}^{(k+g+2n+4)\times n}, \quad w = \begin{bmatrix} \eta\left(\bar{b} - d_{\text{safe}}\right) \\ \eta_{\text{auto}}\left(\bar{b}_{\text{auto}} - d_{\text{safe}}\right) \\ \xi\left(q_{\max} - q\right) \\ -\xi\left(q_{\min} - q\right) \\ -\sigma\left(q_7 - q_6 - \delta_{76}\right) \\ -\sigma\left(q_8 - q_6 - \delta_{86}\right) \\ \sigma\left(q_7 - q_6 - \delta_{76}\right) \\ \sigma\left(q_8 - q_6 - \delta_{86}\right) \end{bmatrix} \in \mathbb{R}^{(k+g+2n+4)} \tag{3.18}$$

Encontrar uma solução $\dot{q}$ para (3.19) implica em encontrar um vetor de velocidades de juntas tal que as restrições de colisão, auto-colisão, limites de junta e juntas passivas sejam satisfeitas. Portanto, o problema de otimização pode ser resolvido ao se tomar valores para as contantes. Para o trabalho, os valores adotados são apresentados na Tabela 1.

$$W\dot{q} \leq w \tag{3.19}$$

Tabela 1 – Valor das constantes adotadas.

Constante	**Valor**
K_r	2
μ	0,5 m
μ_{auto}	0,25 m
d_{safe}	0,01 m
η	0,5
η_{auto}	0,5
ξ	1
σ	0,2

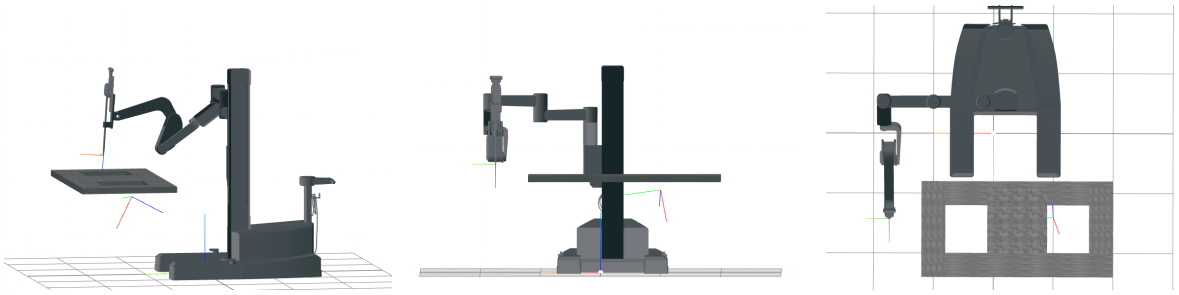
Fonte: Elaboração Própria

3.3 Problema Proposto

Propõe-se, por fim, controlar o segundo braço, ou manipulador, do robô da Vinci Si, visando atingir uma pose alvo realizando, concomitantemente evitamento de obstáculos. Como obstáculos, considera-se todos os objetos do ambiente que podem colidir com algum elo do robô, assim como quaisquer outros elos que podem colidir entre si. Para encontrar uma solução para a variável de controle, pretende-se utilizar a formulação (2.2), utilizando-se a função de tarefa e sua respectiva jacobiana definidas em (3.1) e (3.2) respectivamente.

Figura 13 – Configuração inicial do manipulador e pose alvo para a tarefa

(a) Vista lateral (b) Vista frontal (c) Vista superior



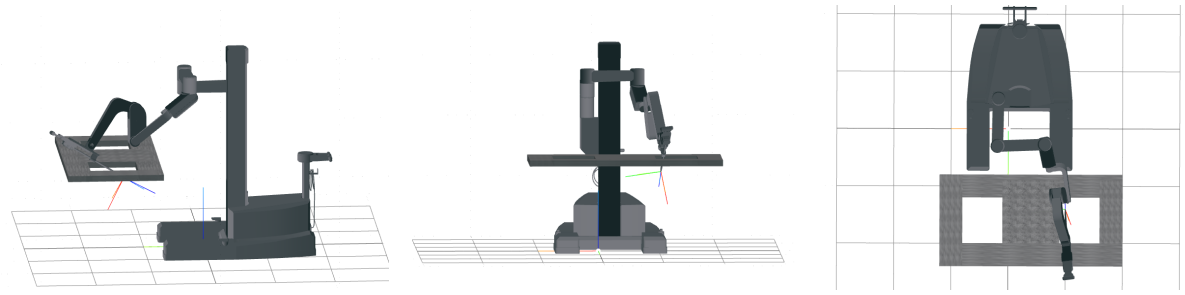
Em cada uma das figuras existem três referenciais: o referencial do efetuador, o referencial do mundo e o referencial da pose alvo, que se encontra abaixo do buraco à direita do obstáculo prateado. Fonte: Elaboração própria

Os demais braços do da Vinci foram removidos por simplicidade e também devido à falta de funções para computar distância entre dois manipuladores distintos. Dessa forma, o ambiente, a pose alvo e a configuração inicial do manipulador são apresentadas na Figura 13.

4 Resultados e Discussão

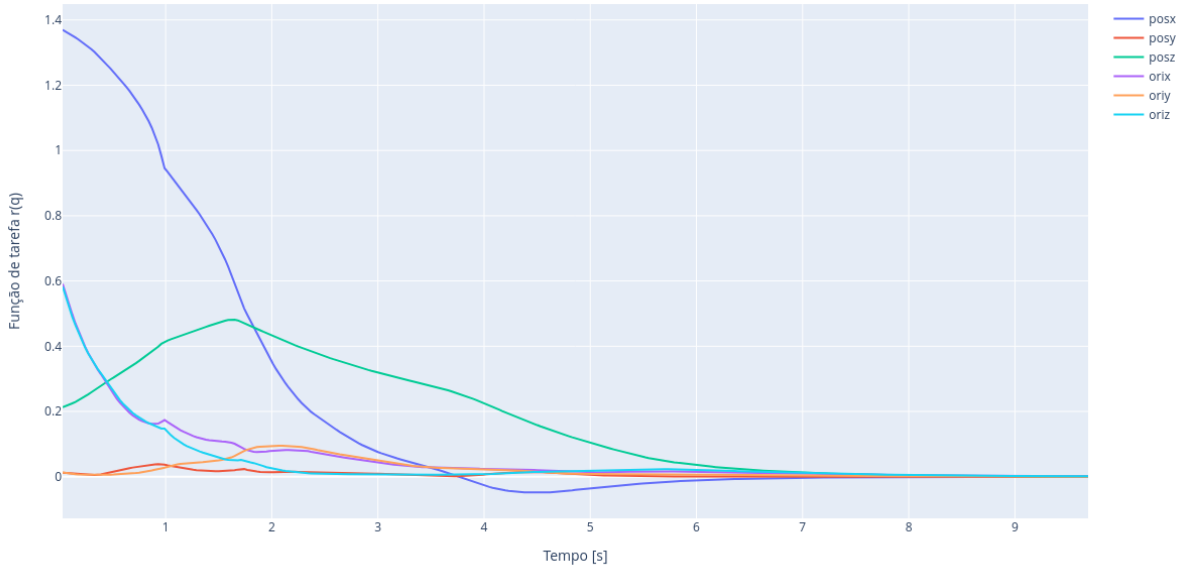
Figura 14 – Manipulador na configuração final, com pose alvo satisfeita e tarefa concluída

(a) Vista lateral (b) Vista frontal (c) Vista superior



Em cada uma das figuras existem três referenciais: o referencial do efetuador, o referencial do mundo e o referencial da pose alvo, que se encontra abaixo do buraco à direita do obstáculo prateado. Vê-se que o referencial do efetuador convergiu para o referencial da pose alvo. Fonte: Elaboração própria

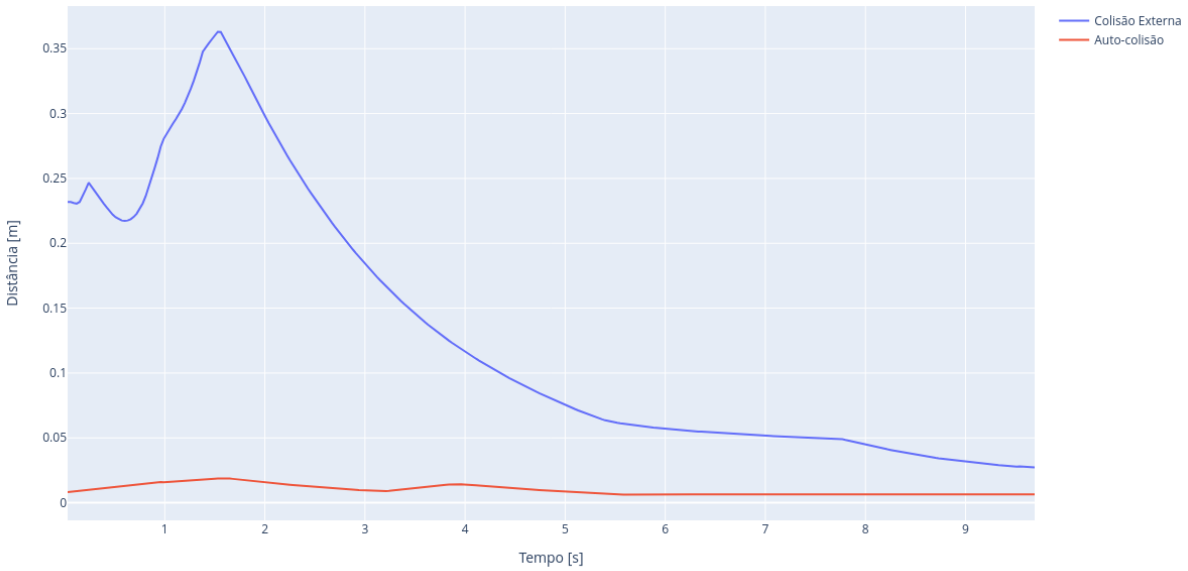
Figura 15 – Função de tarefa ao longo do tempo de execução



A função de tarefa está decomposta nas três componentes de posição (pos) x, y, z e orientação (ori) x, y, z. Fonte: Elaboração própria

Ao se impor todas as restrições por (3.19), alcançou-se satisfatoriamente a pose alvo para a tarefa, Figura 14, por meio da estratégia de controle por programação quadrática convexa, conforme Figura 15. Ainda, nota-se que o manipulador não sofreu colisões externas ou auto-colisões, Figura 16, atendendo-se também à distância de segurança proposta. Nota-se ainda o crescimento da distância entre robô e obstáculo que ocorre em torno

Figura 16 – Distâncias obtidas pela estimativa de colisões externas e auto-colisão ao longo do tempo de execução



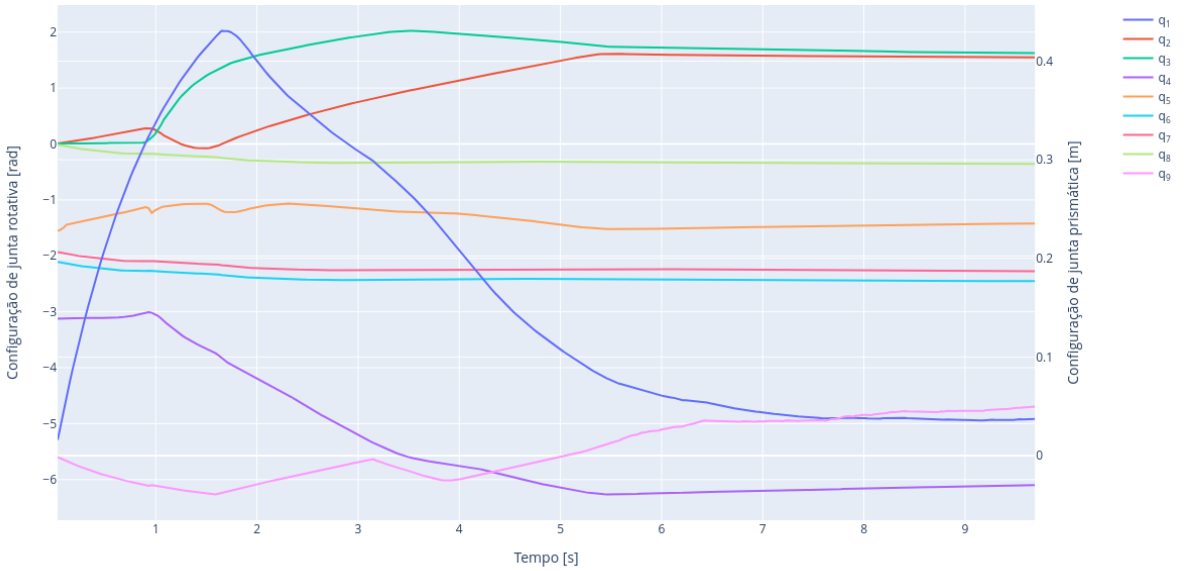
As distâncias são consideradas os valores mínimos de $\Psi_{C_iD_j}$, a distância entre o elo do robô mais próximo e o obstáculo mais próximo (podendo ser também outro elo do robô no caso de auto-colisões). Fonte: Elaboração própria

de 0,6 s na Figura 16, resultado claro da estratégia de controle tomada, que localmente desvia de obstáculos enquanto converge para o objetivo.

É possível observar o comportamento das configurações das juntas na Figura 17, na qual vê-se claramente o acionamento de grande parte das juntas durante todo o movimento, o que não é econômico. Ainda, ao se observar as juntas q_6, q_7 e q_8 fica claro que as restrições de juntas passivas foram atendidas, já que há uma diferença constante entre seus valores, mas suas variações são sempre a mesma. As velocidades das juntas ao longo do tempo, Figura 18, apresentam um comportamento mais ruidoso ao final do movimento, ou seja, conforme a tarefa é alcançada, isso pode ser explicado pela maior proximidade com o obstáculo, que coincide com o fato da tarefa estar próxima de ser alcançada.

Torna-se também de interesse avaliar a redução de complexidade imposta pela hierarquia de primitivas, dada pelo cômputo de distâncias entre AABBs. Dessa forma, o número de linhas de w, e consequentemente o número de restrições consideradas para o problema, pode ser avaliado pela Figura 19. Uma vez que a dimensão de w é dada por $k + g + 2n + 4$, sabe-se que seu valor mínimo é 22, dessa forma qualquer valor acima desse representa uma restrição de colisão externa ou auto-colisão. Nota-se, portanto, que conforme o manipulador se aproxima do obstáculo, o número de distâncias consideradas aumenta, conforme esperado, obtendo-se um valor máximo de 144 linhas e uma média de 121 linhas. Fica claro a vantagem computacional de se considerar hierarquias de primitivas, porém, torna-se necessário estimar um valor μ ideal tal que o número de restrições

Figura 17 – Configuração das juntas ao longo do tempo de execução



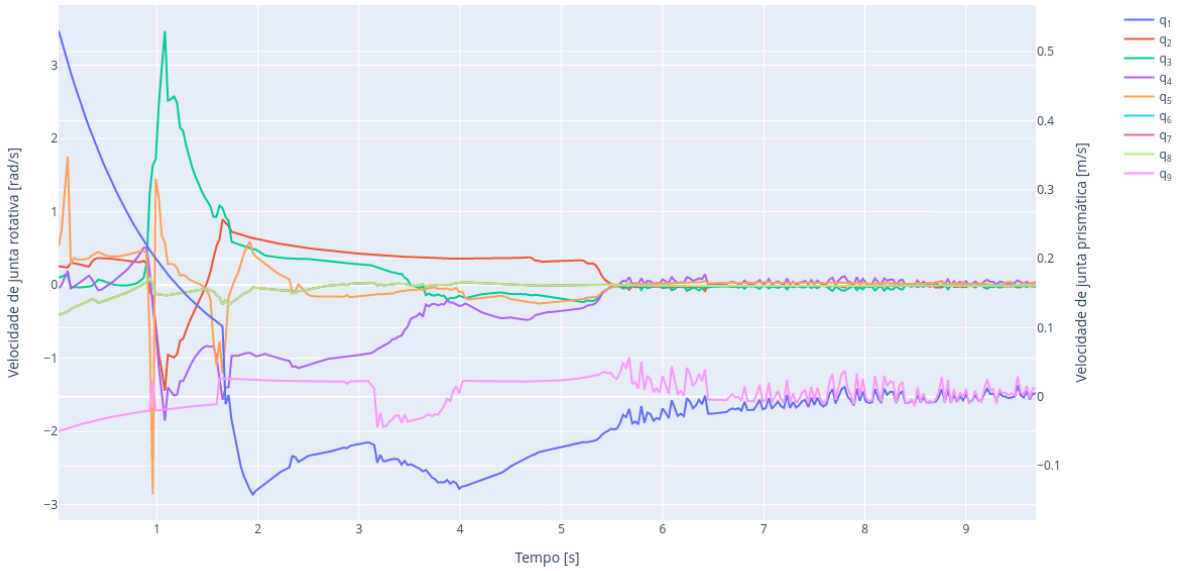
O gráfico apresenta dois eixos verticais para os valores de configuração das juntas, o eixo à esquerda é referência para as juntas rotativas $q_2 - q_8$, enquanto o eixo à direita é referência para as juntas prismáticas q_1 e q_9. Fonte: Elaboração própria

seja o menor possível durante as iterações sem que se afete a convergência para a pose alvo.

Visto a alta dimensão imposta pelas restrições, torna-se de suma importância analisar os tempos gastos para a construção do problema de otimização, isso é, a estimativa de todas as distâncias e a concatenação de cada restrição em uma única matriz, além da solução do problema por QP convexa. Isso pode ser visto na Figura 20, na qual se nota a discrepância entre esses dois tempos e se percebe que o maior gargalo para a implementação de estratégia é a construção do problema de otimização e não a sua solução. Apesar da linguagem utilizada não ser otimizada, demandando mais tempo que C++ ou C para executar funções similares, a diferença de tempo empregado para a construção do problema é muito maior que o tempo de solução, não podendo portanto ser desprezado.

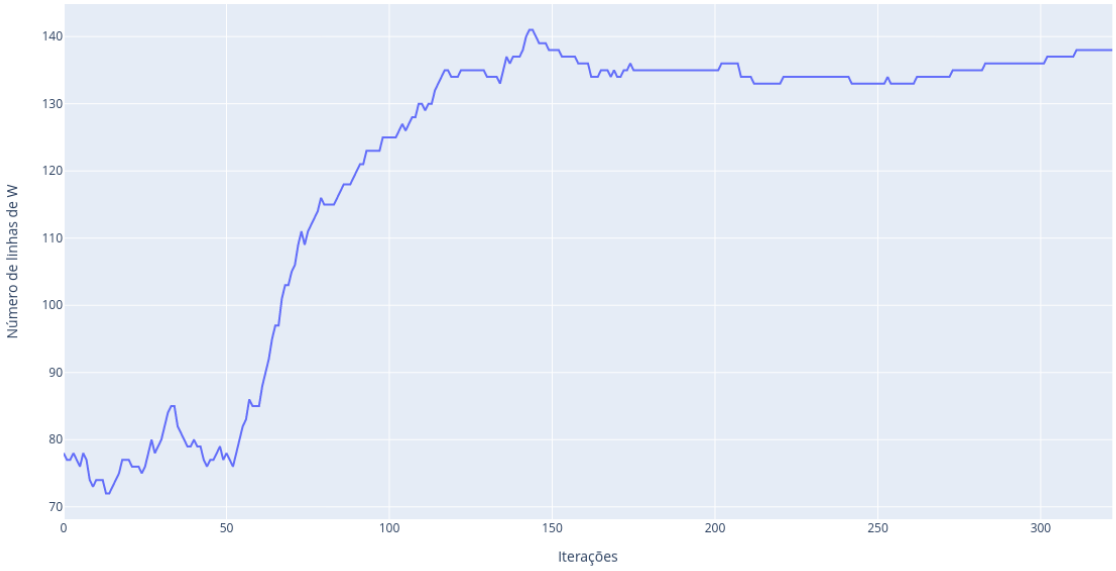
Ao se desprezar as restrições de colisão, tanto colisões externas quanto auto-colisão, pode-se avaliar a qualidade da estratégia de controle previamente utilizada. Vê-se pela Figura 22 e Figura 21 que o manipulador converge à pose alvo, entretanto, nota-se pela Figura 23 que o manipulador colide duas vezes com o obstáculo em questão. Ambas as colisões podem ser vistas na Figura 24, na qual se nota uma colisão direta do manipulador com o obstáculo e outra colisão de uma primitiva com o obstáculo. Ainda, destaca-se a diferença entre as configurações finais do manipulador com restrições de colisão, Figura 14, e manipulador sem restrições de colisão, Figura 21, cuja diferença implica em uma configuração final alvo com colisões no segundo caso.

Figura 18 – Velocidade das juntas ao longo do tempo de execução



O gráfico apresenta dois eixos verticais para os valores de configuração das juntas, o eixo à esquerda é referência para as juntas rotativas $q_2 - q_8$, enquanto o eixo à direita é referência para as juntas prismáticas q_1 e q_9. Fonte: Elaboração própria

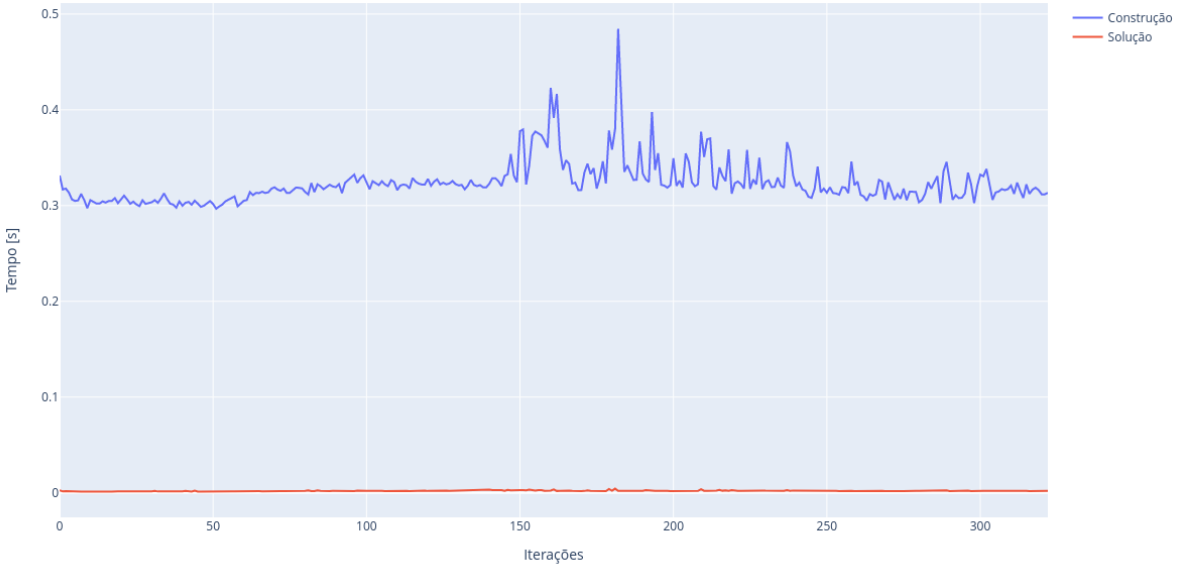
Figura 19 – Número de restrições, ou número de linhas de W, por iteração



Fonte: Elaboração própria

Avaliando-se as configurações das juntas, Figura 25, tem-se novamente a maioria das juntas sendo atuadas durante grande parte do movimento. As velocidades das juntas, Figura 26, indicam que o ruído apresentado ao final do movimento previamente notado, de fato ocorre pela proximidade com o obstáculo e, consequentemente, devido ao cálculo das distâncias envolvidas.

Figura 20 – Tempo gasto para a construção do problema de otimização e para a sua respectiva solução em cada iteração



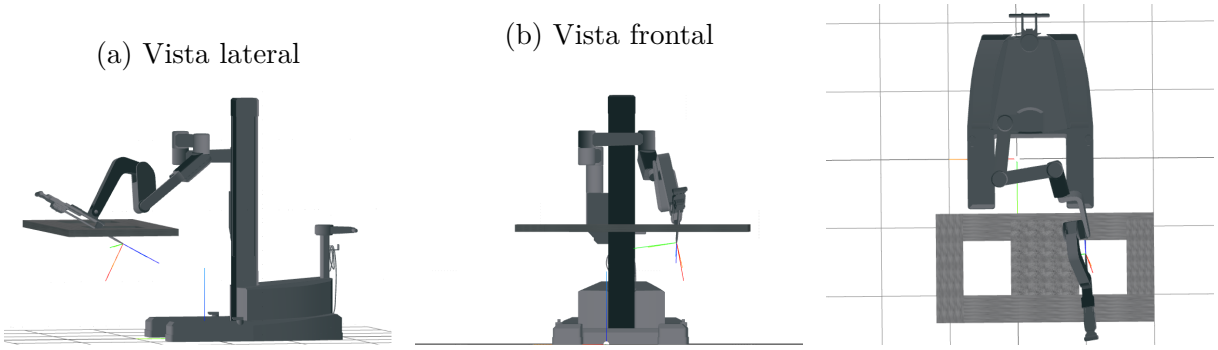
Fonte: Elaboração própria

Figura 21 – Manipulador na configuração final, com pose alvo satisfeita e tarefa concluída, desprezando-se as restrições de colisão (externa e auto-colisão)

(a) Vista lateral

(b) Vista frontal

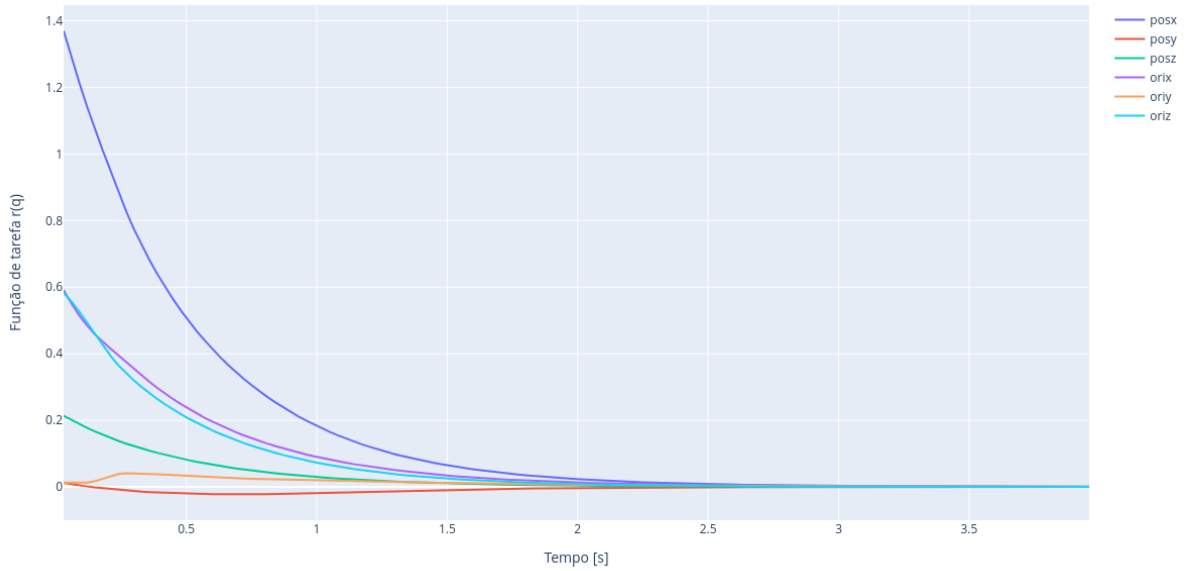
(c) Vista superior



Em cada uma das figuras existem três referenciais: o referencial do efetuador, o referencial do mundo e o referencial da pose alvo, que se encontra abaixo do buraco à direita do obstáculo prateado. Vê-se que o referencial do efetuador convergiu para o referencial da pose alvo. Fonte: Elaboração própria

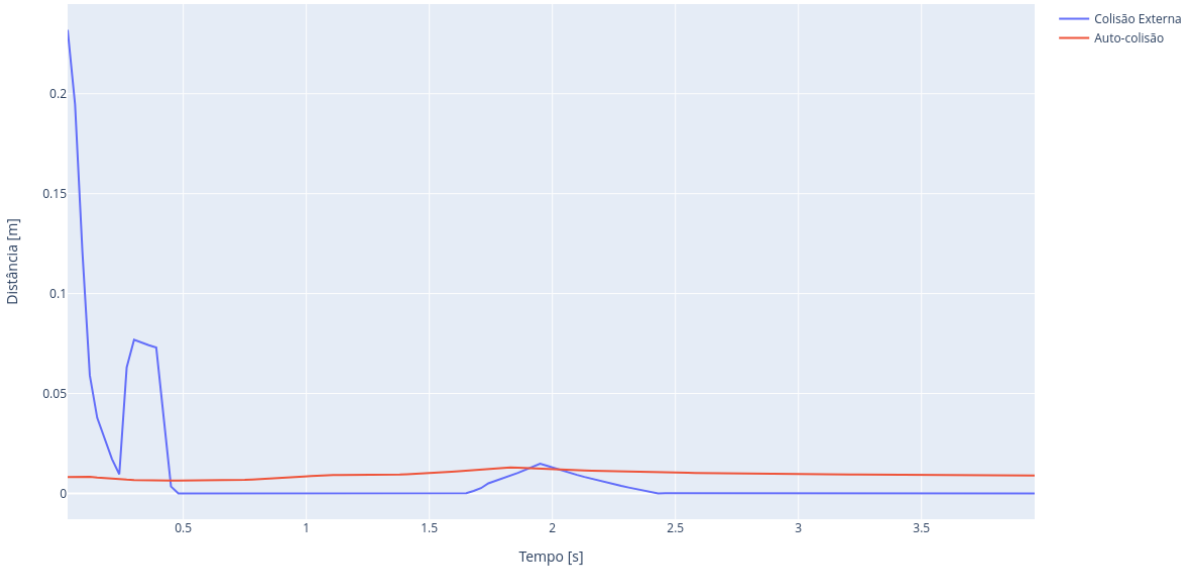
Finalmente, uma vez que o número de restrições consideradas é sempre igual a 22, pode-se avaliar o tempo para construção das restrições de limites de juntas e juntas passivas, incluindo a sua solução, conforme Figura 27. Fica evidente o tempo empregado para o cálculo de todas as distâncias, além da concatenação de diversas matrizes para formar as restrições finais.

Figura 22 – Função de tarefa ao longo do tempo de execução, desprezando-se as restrições de colisão (externa e auto-colisão)



A função de tarefa está decomposta nas três componentes de posição (pos) x, y, z e orientação (ori) x, y, z. Fonte: Elaboração própria

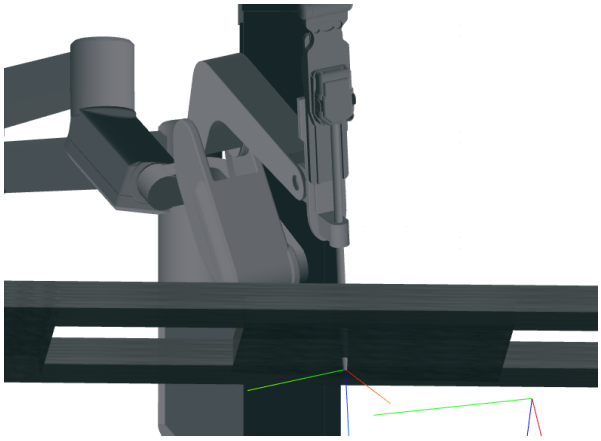
Figura 23 – Distâncias obtidas pela estimativa de colisões externas e auto-colisão ao longo do tempo de execução, desprezando-se as restrições de colisão (externa e auto-colisão)



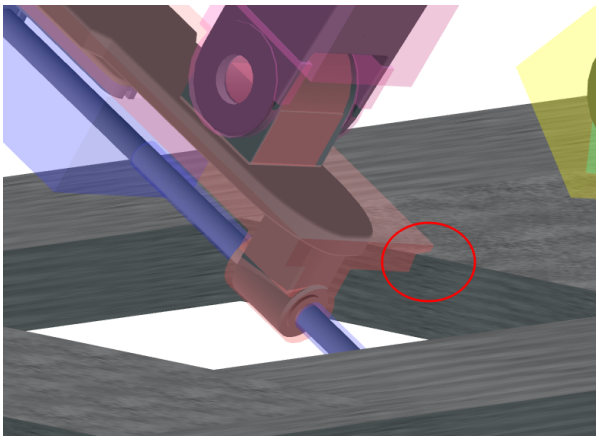
As distâncias são consideradas os valores mínimos de $\Psi_{C_i D_j}$, a distância entre o elo do robô mais próximo e o obstáculo mais próximo (podendo ser também outro elo do robô no caso de auto-colisões). Fonte: Elaboração própria

Figura 24 – Colisões ocorridas para o manipulador ao se desprezar as restrições de colisão

(a) Primeira colisão

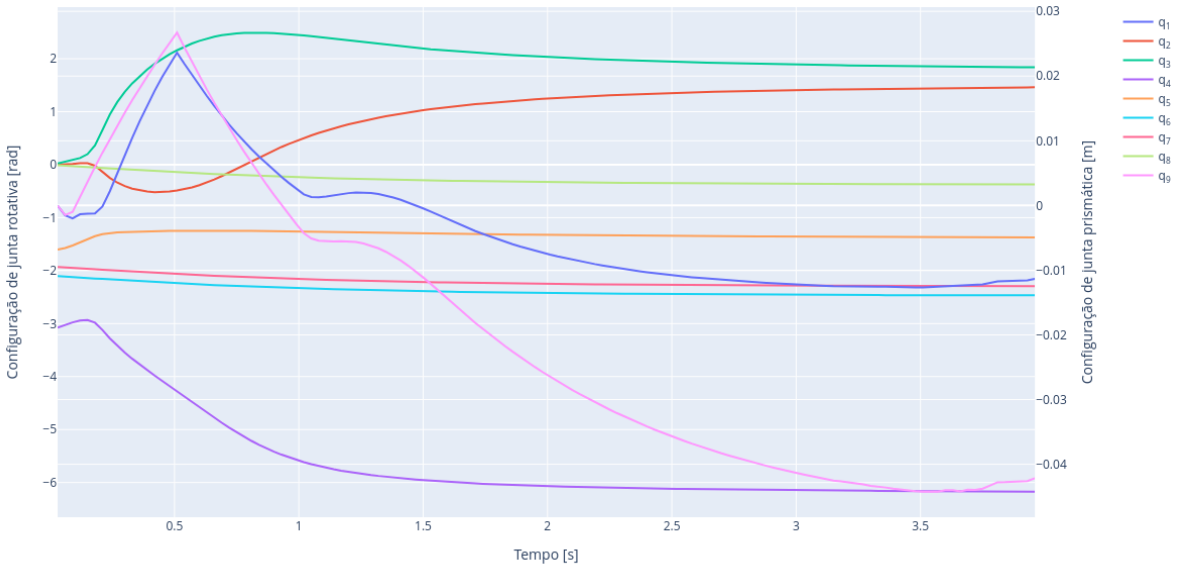


(b) Segunda colisão



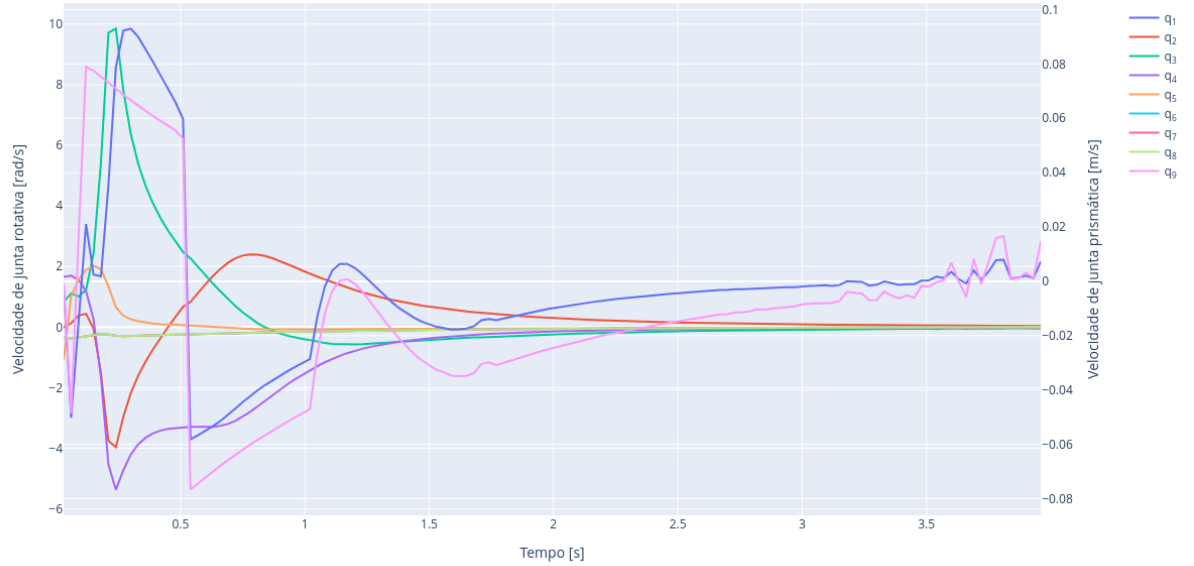
A primeira colisão é fácil notar, a "agulha" do manipulador claramente penetra o obstáculo, já que o referencial do efetuador aparece abaixo do mesmo. A segunda colisão só se torna claro quando se analisa as primitivas de colisão do manipulador, percebe-se que a primitiva colide com o obstáculo no final do movimento, conforme destacado em 24b. Fonte: Elaboração própria

Figura 25 – Configuração das juntas ao longo do tempo de execução, desprezando-se as restrições de colisão (externa e auto-colisão)



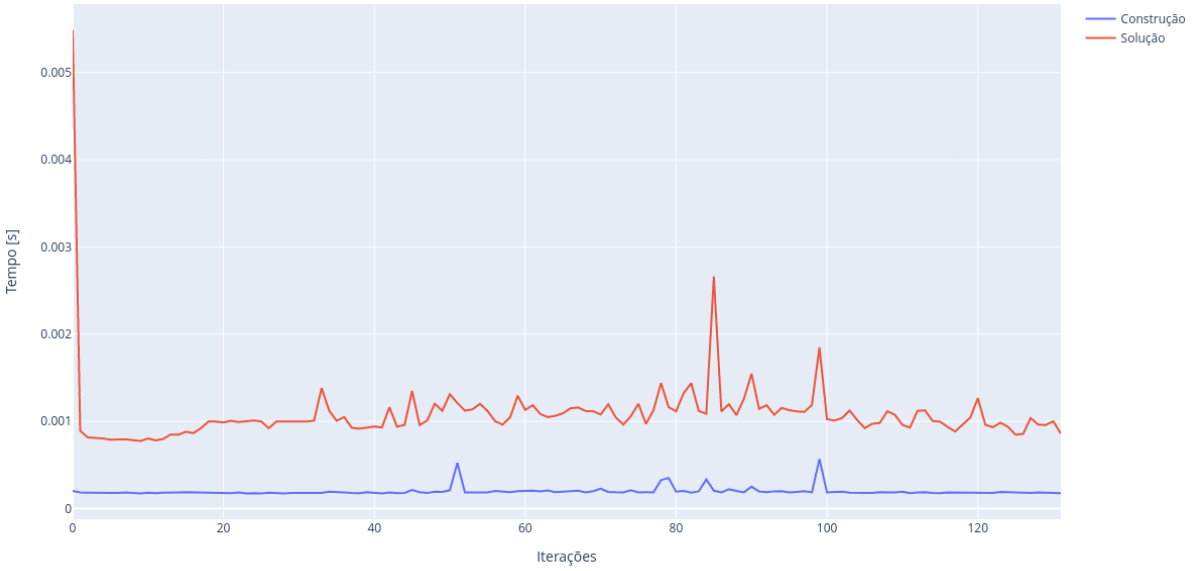
O gráfico apresenta dois eixos verticais para os valores de configuração das juntas, o eixo à esquerda é referência para as juntas rotativas $q_2 - q_8$, enquanto o eixo à direita é referência para as juntas prismáticas q_1 e q_9. Fonte: Elaboração própria

Figura 26 – Velocidade das juntas ao longo do tempo de execução, desprezando-se as restrições de colisão (externa e auto-colisão)



O gráfico apresenta dois eixos verticais para os valores de configuração das juntas, o eixo à esquerda é referência para as juntas rotativas $q_2 - q_8$, enquanto o eixo à direita é referência para as juntas prismáticas q_1 e q_9. Fonte: Elaboração própria

Figura 27 – Tempo gasto para a construção do problema de otimização e para a sua respectiva solução em cada iteração, desprezando-se as restrições de colisão (externa e auto-colisão)



Fonte: Elaboração própria

Uma comparação entre a média de tempo para construção e solução do problema de otimização para os dois casos analisados pode ser observada na Tabela 2.

Tabela 2 – Tempos médios e desvio padrão para a construção e solução do problema de otimização quando há restrições de colisão (externa e auto-colisão) e quando não há

Restrições	**Tempo para Construção**	**Tempo para Solução**
Todas	0,323 48(0,021 79)s	0,002 09(0,000 45)s
Limites de juntas e juntas passivas	0,000 20(0,000 05)s	0,001 09(0,000 44)s

Os dados são apresentados na forma média(desvio padrão). Fonte: Elaboração Própria

5 Trabalhos Futuros

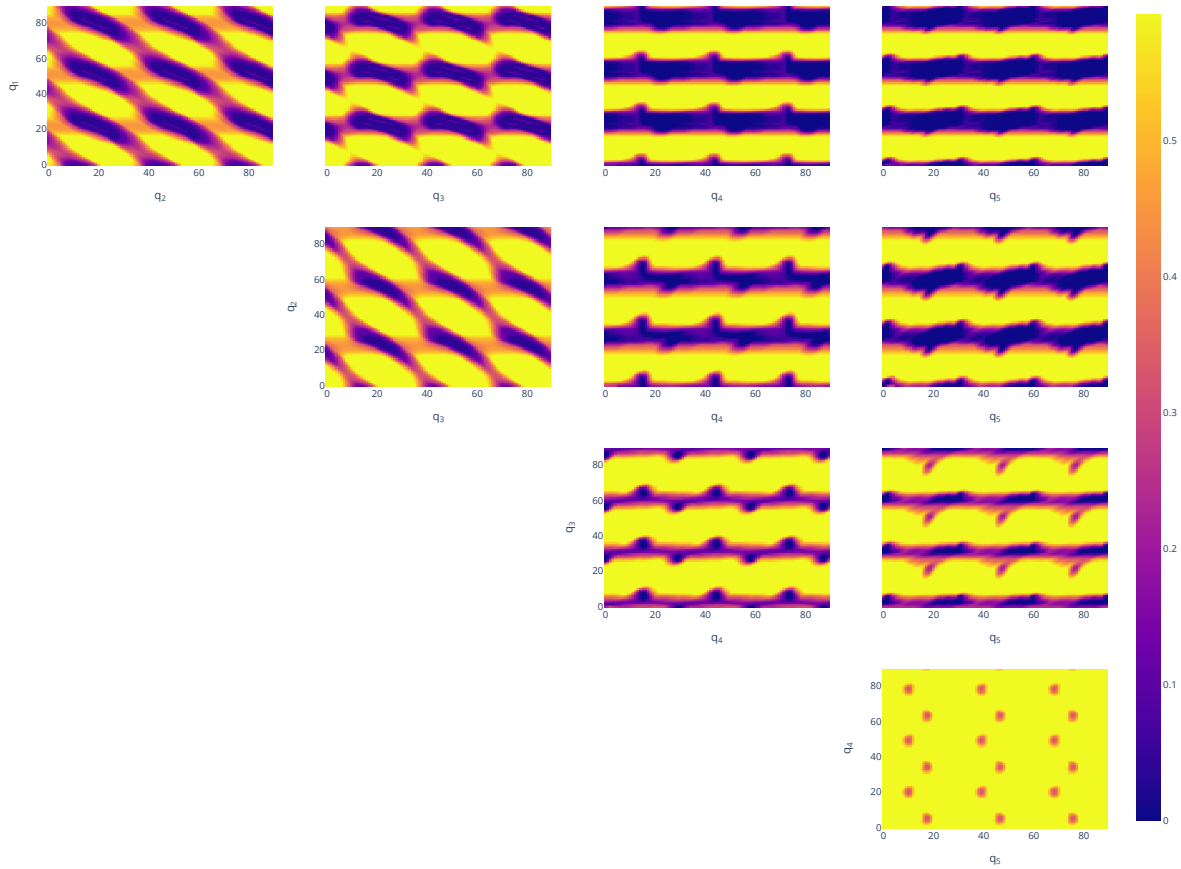
Ao longo da elaboração deste trabalho, notou-se pontos importantes a serem estudados em trabalhos futuros. O primeiro ponto notado foi o alto custo computacional para se obter às matrizes referentes a evitamento de colisão $\bar{A}, \bar{A}_{\text{auto}}$, devido ao custo de se estimar diversas distâncias. Devido à alta dimensão dessas matrizes, o tempo gasto para se estimar todas as distâncias necessárias é assaz elevado, gastando-se 10 vezes mais tempo para se montar o problema de otimização do que solucionar o problema, Tabela 2. Dessa forma, torna-se evidente a necessidade de se estudar a possibilidade de se obter uma nova forma de representação para as restrições e também estudar formas menos custosas de se estimar distâncias entre um robô, composto por peças de variadas geometrias, e obstáculos quaisquer.

O segundo ponto observado foi a possibilidade de se convergir a mínimos locais por meio da estratégia de controle utilizada, minimizando-se uma função objetivo com restrições de inequação por meio de programação quadrática convexa. Dependendo da pose alvo escolhida, o manipulador convergia a mínimos locais de tal forma que as velocidades das juntas se tornavam nulas, com a função de tarefa não minimizada.

Para avaliar se esse comportamento é consequência da complexidade do obstáculo no espaço de configurações, fez-se testes para se compreender superficialmente essa representação. Visto que a configuração do robô é 7-dimensional, é impossível visualizar a representação perfeita do obstáculo no espaço de configurações. Dessa forma, tomou-se cortes bidimensionais do espaço de configurações variando-se as juntas rotativas em pares e observando os resultados da função de distância entre robô e obstáculo. O resultado das distâncias obtidas para cada par de configurações variadas é mostrado na forma de mapas de calor na Figura 28. Nota-se que o obstáculo é composto de quinas, que causam problemas de diferenciabilidade, além da geometria complexa no espaço de configurações.

Por essas observações, sabe-se que algoritmos de planejamento de movimento seriam capazes de contornar o obstáculo, entretanto esses algoritmos demoram o suficiente para não serem ideais para aplicações em robótica cirúrgica. Torna-se necessário pensar em formas reativas de evitar mínimos locais, permitindo-se o uso dessa estratégia em ambientes onde decisões devem ser tomadas rapidamente perante empecilhos não previstos.

Figura 28 – Representação do obstáculo no espaço de configurações por meio de mapas de calor bidimensionais



Cada mapa de calor representa o resultado do cálculo de distâncias entre robô e obstáculo ao se variar pares de configuração. Variou-se somente as juntas rotativas atuáveis $q_1 - q_5$. Fonte: Elaboração própria

6 Conclusões

Os parâmetros de Denavit-Hartenberg estimados visualmente para o da Vinci se mostraram satisfatórios, uma vez que os movimentos observados para as juntas não demonstraram erros perceptíveis e ocasionam nos movimentos esperados para o robô real. A composição de primitivas de colisão escolhida também se mostrou eficaz, visto que permitiram o perfeito evitamento de colisão nos testes realizados. As primitivas também demonstraram eficazes quanto à distância de segurança à elas intrínseca, como pôde ser notado na segunda colisão no teste de controle sem restrições de colisão, para o qual o algoritmo somente aponta uma colisão devido ao maior volume da primitiva, em comparação com o elo em si.

Ainda quanto às primitivas de colisão, notou-se a importância e eficácia do conceito de hierarquia de primitivas, ou hierarquia de volumes delimitadores. Devido à hierarquia formada intrinsecamente pelo simulador, por meio de AABBs, tornou-se possível reduzir a dimensão das matrizes que representam as restrições e consequentemente, reduzir o tempo necessário para a construção do problema de otimização. Entretanto essa hierarquia se demonstrou mais uma decisão de projeto, visto que a imposição de valores μ muito altos não reduzem o custo computacional do problema e valores baixos se mostraram causar a convergência para mínimos locais, somente existentes pela ignorância, durante grande parte do movimento, do manipulador quanto aos obstáculos próximos.

Observou-se a capacidade da solução de problemas de controle, em especial para robótica em um ambiente bastante restrito, por meio de algoritmos de otimização voltados para programação quadrática convexa. Tornou-se possível realizar a tarefa desejada realizando, concomitantemente, evitamento de colisões com o ambiente e auto-colisões. Ainda, sob o mesmo problema de otimização, foi possível considerar restrições de limites de juntas e restrições de juntas passivas. Uma vez que as três primeiras restrições citadas envolvem não igualdades, mas inequações, percebe-se a vantagem clara no uso desse tipo de otimização. Ressalta-se ainda a observação feita quanto ao gargalo para o uso desse tipo de solução para problemas no âmbito de robótica cirúrgica ser a construção do problema de otimização e não obtenção de sua solução. Devido à alta dimensão imposta pelas diversas restrições, torna-se lento o processo de calcular todas as linhas das matrizes de restrições, sendo assim alvo de novas pesquisas uma melhor forma de representação ou obtenção de todas as restrições que governam um problema.

Durante as simulações feitas, notou-se a possibilidade de convergência a mínimos locais pelo algoritmo de programação quadrática utilizado. Devido à localidade das soluções encontradas, é possível que o método convirja a mínimos locais devido à obstáculos

complexos, não sendo possível atingir a pose alvo. Dessa forma, torna-se alvo de trabalhos futuros encontrar soluções para situações desse tipo, sem que se torne necessário utilizar algoritmos de planejamento de movimento que, devido ao seu olhar global sobre o problema, demandam tempo o suficiente para se tornarem inviáveis para o controle em tempo real.

Referências

AMES, A. D. et al. *Control Barrier Functions: Theory and Applications*. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1903.11199>. Citado na página 31.

BAI, L. et al. Solving the Time-Varying Inverse Kinematics Problem for the Da Vinci Surgical Robot. *Applied Sciences*, MDPI AG, v. 9, n. 3, p. 546, feb 2019. ISSN 2076-3417. Disponível em: <http://www.mdpi.com/2076-3417/9/3/546>. Citado 2 vezes nas páginas 23 e 24.

BARTELT, F. *UAIBot Fork: algoritmo de controle para o da Vinci*. [S.l.]: GitHub, 2022. <https://github.com/fbartelt/uaibot/blob/master/uaibot/demo/control_davinci.py>. Citado na página 65.

BOYD, S.; DATTORRO, J. *Alternating Projections*. 2003. Disponível em: <https://web.stanford.edu/class/ee392o/alt_proj.pdf>. Acesso em: 4 nov. 2022. Citado na página 30.

BRADSHAW; GARETH. *Bounding volume hierarchies for level-of-detail collision handling*. 176 p. Tese (Doutorado) — Trinity College, 2002. Disponível em: <http://www.tara.tcd.ie/handle/2262/86786>. Citado 3 vezes nas páginas 29, 39 e 40.

CHANG, J.-W.; WANG, W.; KIM, M.-S. Efficient collision detection using a dual obb-sphere bounding volume hierarchy. *Computer-Aided Design*, v. 42, n. 1, p. 50–57, 2010. ISSN 0010-4485. Advances in Geometric Modelling and Processing. Disponível em: <https://www.sciencedirect.com/science/article/pii/S001044850900102X>. Citado na página 29.

CHENEY, W.; GOLDSTEIN, A. A. Proximity maps for convex sets. *Proceedings of the American Mathematical Society*, American Mathematical Society (AMS), v. 10, n. 3, p. 448–450, 1959. ISSN 0002-9939. Disponível em: <https://www.ams.org/proc/1959-010-03/S0002-9939-1959-0105008-8/>. Citado na página 30.

DESAI, J. P. et al. *The Encyclopedia of Medical Robotics*. [S.l.]: World Scientific, 2017. v. 1. Citado 3 vezes nas páginas 22, 24 e 25.

DINAS, S.; BAÑÓN, J. M. Revisión de literatura de jerarquía volúmenes acotantes enfocados en detección de colisiones. *INGENIERÍA Y COMPETITIVIDAD*, v. 17, n. 1, p. 49–62, jun. 2015. Disponível em: <https://revistaingenieria.univalle.edu.co/index.php/ingenieria_y_competitividad/article/view/2200>. Citado 3 vezes nas páginas 29, 39 e 40.

ESCALANTE, R.; RAYDAN, M. *Alternating Projection Methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2011. Disponível em: <https://epubs.siam.org/doi/abs/10.1137/9781611971941>. Citado 2 vezes nas páginas 29 e 30.

ESCANDE, A.; MANSARD, N.; WIEBER, P.-B. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, SAGE Publications Inc., v. 33, n. 7, p. 1006–1028, 2014. ISSN

17413176. Disponível em: <https://hal.archives-ouvertes.fr/hal-00751924https://hal.archives-ouvertes.fr/hal-00751924/document>. Citado na página 32.

FAVERJON, B.; TOURNASSOUD, P. A local based approach for path planning of manipulators with a high number of degrees of freedom. In: *Proceedings. 1987 IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 1987. v. 4, p. 1152–1159. Citado na página 31.

FONTANELLI, G. A. et al. Modelling and identification of the da Vinci Research Kit robotic arms. In: *IEEE International Conference on Intelligent Robots and Systems*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2017. v. 2017-September, p. 1464–1469. ISBN 9781538626825. ISSN 21530866. Citado na página 24.

GONÇALVES, V. M. *Manipuladores Robóticos*. [S.l.]: GitHub, 2022. <https://viniciusmgn.github.io/aulas_manipuladores/>. Citado 2 vezes nas páginas 36 e 37.

GONÇALVES, V. M. *UAIBot*. [S.l.]: GitHub, 2022. <https://github.com/viniciusmgn/uaibot_vinicius>. Citado 3 vezes nas páginas 25, 26 e 33.

GONÇALVES, V. M. et al. Stable-by-Design Kinematic Control Based on Optimization. *IEEE Transactions on Robotics*, Institute of Electrical and Electronics Engineers Inc., v. 36, n. 3, p. 644–656, jun 2020. ISSN 19410468. Citado na página 32.

GONÇALVES, V. M. et al. Parsimonious Kinematic Control of Highly Redundant Robots. *IEEE Robotics and Automation Letters*, Institute of Electrical and Electronics Engineers Inc., v. 1, n. 1, p. 65–72, jan 2016. ISSN 23773766. Citado na página 32.

HAIDEGGER, T. Autonomy for Surgical Robots: Concepts and Paradigms. *IEEE Transactions on Medical Robotics and Bionics*, Institute of Electrical and Electronics Engineers (IEEE), v. 1, n. 2, p. 65–76, apr 2019. Citado 2 vezes nas páginas 21 e 22.

IEC/TR 60601-4-1:2017(E). Standard. *Medical electrical equipment - Part 4-1: Guidance and interpretation - Medical electrical equipment and medical electrical systems employing a degree of autonomy*. Geneva, CH: International Organization for Standardization, 2017. Disponível em: <https://www.iso.org/standard/70755.html>. Citado na página 21.

Intuitive Surgical. *Da Vinci Stapling*. 2022. Disponível em: <https://www.intuitive.com/en-us/products-and-services/da-vinci/stapling>. Acesso em: 23 out. 2022. Citado na página 25.

KANOUN, O.; LAMIRAUX, F.; WIEBER, P.-B. Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, v. 27, n. 4, p. 785–792, aug 2011. ISSN 15523098. Citado 3 vezes nas páginas 31, 32 e 39.

KHATIB, O. Real-time obstacle avoidance for manipulators and mobile robots. In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 1985. v. 2, p. 500–505. Citado na página 31.

LOMBARD, B.; CÉRUSE, P.; FUMAT, C. *Robotics and Digital Guidance in ENT-H&N Surgery: Rapport SFORL 2017*. Elsevier Health Sciences, 2017. ISBN 9782294756719. Disponível em: <https://books.google.com.br/books?id=RyQmDwAAQBAJ>. Citado 3 vezes nas páginas 23, 24 e 25.

MARINHO, M. M. et al. A unified framework for the teleoperation of surgical robots in constrained workspaces. In: *Proceedings - IEEE International Conference on Robotics and Automation*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2019. v. 2019-May, p. 2721–2727. ISBN 9781538660263. ISSN 10504729. Citado na página 32.

MARINHO, M. M. et al. Dynamic Active Constraints for Surgical Robots Using Vector-Field Inequalities. *IEEE Transactions on Robotics*, Institute of Electrical and Electronics Engineers Inc., v. 35, n. 5, p. 1166–1185, oct 2019. ISSN 19410468. Citado 2 vezes nas páginas 32 e 37.

MOUSTRIS, G. P. et al. Evolution of autonomous and semi-autonomous robotic surgical systems: a review of the literature. *The International Journal of Medical Robotics and Computer Assisted Surgery*, John Wiley & Sons, Ltd, v. 7, n. 4, p. 375–392, dec 2011. ISSN 14785951. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1002/rcs.408>. Citado 2 vezes nas páginas 21 e 22.

OKAN, K. *Da Vinci Surgical Robot*. 2021. Disponível em: <https://grabcad.com/library/da-vinci-surgical-robot-1/details?folder_id=10427669>. Acesso em: 5 jul. 2022. Citado na página 33.

O'SULLIVAN, S. et al. Legal, regulatory, and ethical frameworks for development of standards in artificial intelligence (AI) and autonomous robotic surgery. *The International Journal of Medical Robotics and Computer Assisted Surgery*, John Wiley and Sons Ltd, v. 15, n. 1, p. e1968, feb 2019. ISSN 14785951. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1002/rcs.1968>. Citado na página 22.

PANESAR, S. et al. Artificial Intelligence and the Future of Surgical Robotics. *Annals of Surgery*, Lippincott Williams and Wilkins, v. 270, n. 2, p. 223–226, aug 2019. ISSN 0003-4932. Disponível em: <https://journals.lww.com/00000658-201908000-00007>. Citado na página 21.

ROBOTS (IEEE Spectrum). *Da Vinci*. 2018. Disponível em: <https://robots.ieee.org/robots/davinci/?gallery=interactive1>. Acesso em: 23 out. 2022. Citado na página 23.

SICILIANO, B. et al. *Robotics: Modelling, Planning and Control*. 1. ed. London, England: Springer, Cop, 2009. Citado na página 31.

SICILIANO, B.; SLOTINE, J.-J. A general framework for managing multiple tasks in highly redundant robotic systems. In: *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*. [S.l.: s.n.], 1991. p. 1211–1216 vol.2. Citado na página 31.

Apêndices

APÊNDICE A – Códigos Fonte

Algoritmo A.1 – Código principal, que executa o Algoritmo A.2

```
import plotly.graph_objects as go
from demo.control_test import _control_demo_davinci

sim, (build_ts, solve_ts, size_restrictions, plot_tuple) = _control_demo_davinci(arm
    =1)

fig = go.Figure(go.Scatter(y=build_ts, name='Construcao'))
fig.add_scatter(y=solve_ts, name='Solucao')
fig.update_xaxes(title='Iteracoes')
fig.update_yaxes(title='Tempo [s]')
fig.update_layout(width=1200, height=600, margin=dict(t=10))
fig.show()
```

Algoritmo A.2 – Trechos principais do algoritmo da estratégia de controle (BARTELT, 2022)

```
import sys
import time
import numpy as np
import robot as rb
from simulation import Simulation
from utils import Utils
from simobjects.box import Box
from simobjects.ball import Ball
from simobjects.frame import Frame
from cvxopt import matrix, solvers
from graphics.meshmaterial import MeshMaterial, Texture



right_foot = Box(htm=Utils.trn([0.26, -0.26, 0.09]),
                 width=0.25, height=0.19, depth=1.4, opacity=0.5)
left_foot = Box(htm=Utils.trn([-0.46, -0.26, 0.09]),
                width=0.25, height=0.19, depth=1.4, opacity=0.5)
center_foot = Box(htm=Utils.trn(
    [-0.1037, -0.52, 0.21]), width=0.62, height=0.44, depth=0.8, opacity=0.5)
```

```
cable_rest = Box(htm=Utils.trn([—0.1037, —0.93, 0.4]),
                 width=0.35, height=0.82, depth=0.35, opacity=0.5)
tower = Box(htm=Utils.trn([—0.10, —0.275, 0.9]),
            width=0.2, height=1.7, depth=0.19, opacity=0.5)
stat_parts = [right_foot, left_foot, center_foot, cable_rest, tower]


def auto_collision_helper(robot_, q=None, stat_parts=stat_parts, old_struct=[None,
    None, None, None, None], max_dist=np.inf):
    """Computes autocollision between joints and stationary part (chest) of davinci
        """
    dist_vect, gradD_list, struct = [], [], []
    n = len(robot_.links)

    for i, obj in enumerate(stat_parts):
        dist_struct = robot_.compute_dist(
            obj, q=q, old_dist_struct=old_struct[i], max_dist=max_dist)
        dist, grad = np.array(dist_struct.dist_vect), np.array(
            dist_struct.jac_dist_mat)
        idxs_aux = 1 — np.zeros_like(dist, dtype=bool).ravel()

        # Skip first joint distances
        for k, d in enumerate(dist_struct):
            if d.link_number == 0:
                idxs_aux[k] = False
            elif d.link_number == 1 and d.link_col_obj_number == 2:
                idxs_aux[k] = False

        dist = dist[idxs_aux.astype(bool)]
        grad = grad[idxs_aux.astype(bool)]
        if dist.size > 0:
            dist_vect.append(dist)
            gradD_list.append(grad)
        struct.append(dist_struct)

    dist_vect = np.array(dist_vect).reshape(—1, 1)
    gradD_list = np.array(gradD_list).reshape(—1, n)

    return dist_vect, gradD_list, struct
```

```
def _control_demo_davinci(arm=0):
    solvers.options['show_progress'] = False
    np.set_printoptions(precision=4, suppress=True, linewidth=150)

    robot_arm1, robot_arm2, robot_arm3, robot_arm4, chest = rb.Robot.create_davinci
        ().list_of_objects
    arms = [robot_arm1, robot_arm2, robot_arm3, robot_arm4]
    robot = arms[arm]

    texture_wall = Texture(
        url='https://raw.githubusercontent.com/viniciusmgn/uaibot_content/master/
            contents/Textures/rough_metal.jpg',
        wrap_s='RepeatWrapping', wrap_t='RepeatWrapping', repeat=[4, 4])

    material_wall = MeshMaterial(texture_map=texture_wall, clearcoat=0.3, roughness
        =0.8,
                                 metalness=0.7, opacity=0.9, color='silver', side="
                                     DoubleSide")

    wall1 = Box(name="wall1", htm=Utils.trn([0.6 - 0.9, 0.5, 0.8]) @ Utils.roty(-np.
        pi/2), width=0.05, depth=0.2, height=1.4,
                mesh_material=material_wall)
    wall2 = Box(name="wall2", htm=Utils.trn([0.6 - 0.9, 1.1, 0.8]) @ Utils.roty(-np.
        pi/2), width=0.05, depth=0.2, height=1.4,
                mesh_material=material_wall)
    wall3 = Box(name="wall3", htm=Utils.trn([0. - 0.9, 0.75, 0.8]) @ Utils.roty(-np.
        pi/2), width=0.05, depth=0.6, height=0.2,
                mesh_material=material_wall)
    wall4 = Box(name="wall4", htm=Utils.trn([0.7 - 0.9, 0.75, 0.8]) @ Utils.roty(-np
        .pi/2), width=0.05, depth=0.6, height=0.5,
                mesh_material=material_wall)
    wall5 = Box(name="wall5", htm=Utils.trn([1.4 - 0.9, 0.8, 0.8]) @ Utils.roty(-np.
        pi/2), width=0.05, depth=0.8, height=0.2,
                mesh_material=material_wall)

    def evaluate_error(r, tol_pos=5e-3):
        error_pos = max(abs(r[0:3]))
```

```
    error_ori = (180 / np.pi) * max(abs(np.arccos(1 - r[3:6])))
    ok1 = error_pos < tol_pos
    ok2 = error_ori < 5 if len(r.tolist()) > 3 else True

    return ok1 and ok2, error_pos, error_ori

def dist_computation(q, old_struct, obstacles=[wall1, wall2, wall3, wall4, wall5
    ], max_dist=np.inf):
    n = len(robot.links)
    dist_vect, gradD_list, struct = 5, [5,]*n, []

    for i, obstacle in enumerate(obstacles):
        dist = robot.compute_dist(
            obj=obstacle, q=q, old_dist_struct=old_struct[i], max_dist=max_dist)
        struct.append(dist)
        if dist.dist_vect.size > 0:
            if isinstance(dist_vect, np.ndarray):
                dist_vect = np.vstack(
                    (dist_vect, np.array(dist.dist_vect).reshape(-1, 1)))
                gradD_list = np.vstack(
                    (gradD_list, np.array(dist.jac_dist_mat).reshape(-1, n)))
            else:
                dist_vect = np.array(dist.dist_vect).reshape(-1, 1)
                gradD_list = np.array(dist.jac_dist_mat).reshape(-1, n)

    dist_vect = np.array(dist_vect).reshape(-1, 1)
    gradD_list = np.array(gradD_list).reshape(-1, n)

    return dist_vect, gradD_list, struct

# Target pose definition
pose_tg = []
pose_tg.append(Utils.trn([0.3 - 0.8, 0.7, .7]) @ Utils.rotx(np.deg2rad(130))
               @ Utils.rotz(np.deg2rad(-100)) @ Utils.roty(np.deg2rad(-12)))

# Create simulation
sim = Simulation.create_sim_grid(
    [robot, wall1, wall2, wall3, wall4, wall5, chest])
```

```
for k in range(len(pose_tg)):
    sim.add(Frame(name="pose_tg_" + str(k), htm=pose_tg[k]))

# Parameters
dt = 0.03
K = 2
eta = 0.5
sigma = 0.2
dist_safe = 0.01
max_dist = 0.5
xi = 1
q = robot.q.astype(float)
n = len(robot.links)
Ntest = 4
qdot = np.zeros((n, 1)).astype(float)

# Initializations
obstacles = [wall1, wall2, wall3, wall4, wall5]
struct = [None,] * len(obstacles)
auto_struct = None
auto_struct2 = [None,] * len(stat_parts)

i = 1
imax = round((47.4/2)/dt)

hist_dist = []
hist_time = []
hist_r = np.matrix(np.zeros(shape=(6, 0)))
hist_q = np.matrix(np.zeros(shape=(n, 0)))
hist_qdot = np.matrix(np.zeros(shape=(n, 0)))
build_ts, solve_ts = [], []
size_restrictions = []

error_qp = False
iteration_end = False

# Changing Initial configuration
qaux = q.copy()
qaux[3] = —np.deg2rad(179)
```

```
qaux[4] = -np.pi/2
qaux[5] = -np.deg2rad(120)
qaux[6] = -np.deg2rad(110)
qaux[7] = 0
robot.add_ani_frame(0, qaux)
q65_c = qaux[6] - qaux[5]
q75_c = qaux[7] - qaux[5]
q = robot.q.astype(float)

# Main loop
for k in range(len(pose_tg)):
    converged = False
    while not converged and not error_qp and not iteration_end:

        qdot_max = robot.joint_limit[:, 1] - q.reshape(-1, 1)
        qdot_min = robot.joint_limit[:, 0] - q.reshape(-1, 1)
        qdot_max[1:-1] *= 10
        qdot_min[1:-1] *= 10

        # This is just for showing progress when the simulation is run
        if i % 50 == 0 or i == imax - 1:
            sys.stdout.write('\r')
            sys.stdout.write(
                "[%-20s] %d%%" % ('=' * round(20 * i / (imax - 1)), round(100 *
                    i / (imax - 1))))
            sys.stdout.flush()

        # Compute r
        [r, jac_r] = robot.task_function(pose_tg[k], q=q)

        # Compute dist_vect, dist_vect_dot and the distance Jacobian
        build_t0 = time.time()
        dist_vect, gradD_list, struct = dist_computation(
            q, struct, obstacles=obstacles, max_dist=max_dist)

        # Create the quadratic program parameters
        A = -gradD_list
        b = eta * (dist_vect - dist_safe)
        #idx = np.argsort(dist_vect.ravel())
```

```
# AutoCollision
auto_struct = robot.compute_dist_auto(max_dist=max_dist/2)
dist_vect_auto, jac_dist_auto = auto_struct.dist_vect, auto_struct.
    jac_dist_mat
dist_vect_auto, jac_dist_auto = np.array(
    dist_vect_auto), np.array(jac_dist_auto)
idx_auto = np.argsort(dist_vect_auto.ravel())
b_auto = eta * \
    (np.array(dist_vect_auto).reshape(-1, 1) - dist_safe)
b_auto = b_auto[idx_auto]
A_auto = -np.array(jac_dist_auto).reshape(-1, n)
A_auto = A_auto[idx_auto, :]
b_auto2, A_auto2, auto_struct2 = auto_collision_helper(
    robot, old_struct=auto_struct2, max_dist=max_dist)
b_auto2 = eta * (b_auto2 - dist_safe)
A_auto2 = -A_auto2

H = 2 * (jac_r.T * jac_r) + 0.001 * np.identity(n)
f = (2 * K * r.T @ jac_r).T
q6_rest = np.array([0, 0, 0, 0, 0, -1, 1, 0, 0]).reshape(1, -1)
q7_rest = np.array([0, 0, 0, 0, 0, -1, 0, 1, 0]).reshape(1, -1)
A = np.block([[A], [A_auto], [A_auto2], [np.identity(
    n)], [-np.identity(n)], [q6_rest], [q7_rest], [-q6_rest], [-q7_rest
        ]])
b = np.block([[b], [b_auto], [b_auto2], [xi * qdot_max], [-xi * qdot_min
    ], [-sigma * (q[6] - q[5] - q65_c)],
              [-sigma * (q[7] - q[5] - q75_c)], [sigma * (q[6] - q[5] -
                  q65_c)], [sigma * (q[7] - q[5] - q75_c)]])
deltaT_build = time.time() - build_t0

if dist_vect.size <= 0:
    dist_vect = np.array([[4]])
size_restrictions.append(b.shape[0])

# Solve the quadratic program
solve_t0 = time.time()
try:
    qdot = solvers.qp(matrix(H), matrix(
```

```
                f), matrix(A), matrix(b))['x']
        except:
            qdot = np.matrix(np.zeros((n, 1)))
            error_qp = True

        deltaT_solve = time.time() - solve_t0
        qdot = np.array(qdot).reshape(n, 1)

        # First order explicit Euler simulation
        q += qdot * dt

        # Add animation to simulation
        robot.add_ani_frame(i * dt, q)

        # Store data for showing the graphs later
        hist_time.append(i * dt)
        hist_dist.append(np.amin(dist_vect))
        hist_r = np.block([hist_r, r])
        hist_q = np.block([hist_q, q])
        hist_qdot = np.block([hist_qdot, qdot])
        build_ts.append(deltaT_build)
        solve_ts.append(deltaT_solve)

        # Continue the loop, check if converged
        i += 1
        converged, error_pos, error_ori = evaluate_error(r, 5e-4)
        if converged:
            print(error_pos, error_ori)

        iteration_end = i > imax
print(converged, error_qp, iteration_end, i)
# Show collision primitives
# for i in robot.links:
#     for j in i.col_objects:
#         sim.add(j[0])
# robot.update_col_object(0)
# Run simulation
sim.run()
```

```
    # Plot graphs
    Utils.plot(hist_time, hist_dist, "", "Time (s)",
               "True distance (m)", "dist")
    Utils.plot(hist_time, hist_q, "", "Time (s)",
               "Joint configuration (rad)", "q")
    Utils.plot(hist_time, hist_qdot, "", "Time (s)",
               "Joint speed (rad/s)", "qdot")
    fig = Utils.plot(hist_time, hist_r, "", "Time (s)", "Task function",
                     ["posx", "posy", "posz", "orix", "oriy", "oriz"])

    return sim, (build_ts, solve_ts, size_restrictions, (hist_time, hist_q,
        hist_dist, hist_qdot, hist_r))
```