UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA

ALINE DE CÁSSIA MAGALHÃES

Proposta de algoritmo otimizador de caminhos para manipuladores robóticos em ambientes com presença de obstáculos

Uberlândia

2020

ALINE DE CÁSSIA MAGALHÃES

Proposta de algoritmo otimizador de caminhos para manipuladores robóticos em ambientes com presença de obstáculos

Versão Original

Dissertação apresentada à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia (UFU), como parte das exigências para a obtenção do título de Mestre.

Orientador: Prof. Dr. Aniel Silva de Morais

Uberlândia

2020

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

M188 Magalhães, Aline de Cássia, 1996-
2020 Proposta de algoritmo otimizador de caminhos para manipuladores robóticos em ambientes com presença de obstáculos [recurso eletrônico] / Aline de Cássia Magalhães. - 2020.

Orientador: Aniel Silva de Morais.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Engenharia Elétrica.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2020.515
Inclui bibliografia.
Inclui ilustrações.

1. Engenharia elétrica. I. Morais, Aniel Silva de,1979-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Engenharia Elétrica. III. Título.

CDU: 621.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Engenharia Elétrica
Av. João Naves de Ávila, 2121, Bloco 3N - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4707 - www.posgrad.feelt.ufu.br - copel@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Engenharia Elétrica				
Defesa de:	Dissertação de Mestrado Acadêmico, 740, PPGEELT				
Data:	Quatorze de agosto de dois mil e vinte.	Hora de início:	09:00	Hora de encerramento:	11:00
Matrícula do Discente:	11812EEL013				
Nome do Discente:	Aline de Cássia Magalhães				
Título do Trabalho:	Proposta de algoritmo otimizador de caminhos para manipuladores robóticos em ambientes com presença de obstáculos				
Área de concentração:	Sistemas de Energia Elétrica				
Linha de pesquisa:	Controle e Automação				
Projeto de Pesquisa de vinculação:	Título: Técnicas de identificação de sistemas visando aplicabilidade em processos industriais Agência Financiadora: UNIVERSIDADE FEDERAL DE UBERLÂNDIA - (Programa Institucional de Apoio a Pesquisa (apoio financeiro). Início 02/01/2013 Término __/__/__No. do Projeto na agência: __ Professor Coordenador: Marcio Jose da Cunha				

Reuniu-se por meio de videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Engenharia Elétrica, assim composta: Professores Doutores: Daniel Costa Ramos - FEELT/UFU; Josué Silva de Morais - FEELT/UFU; Gabriela Vieira Lima - UFRPE; Aniel Silva de Morais - FEELT/UFU, orientador(a) do(a) candidato(a).

Iniciando os trabalhos o(a) presidente da mesa, Dr. Aniel Silva de Morais, apresentou a Comissão Examinadora e a candidata, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor(a) presidente concedeu a palavra, pela ordem sucessivamente, aos(às) examinadores(as), que passaram a arguir o(a) candidato(a). Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o(a) candidato(a):

Aprovada.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Aniel Silva de Morais**, **Professor(a) do Magistério Superior**, em 14/08/2020, às 12:12, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Daniel Costa Ramos**, **Professor(a) do Magistério Superior**, em 14/08/2020, às 12:24, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Gabriela Vieira Lima**, **Usuário Externo**, em 14/08/2020, às 12:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Josué Silva de Morais**, **Professor(a) do Magistério Superior**, em 14/08/2020, às 12:40, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2130596** e o código CRC **B89C8E2E**.

Referência: Processo nº 23117.039382/2020-10 SEI nº 2130596

A Deus, meu maior orientador. À minha família, por seu apoio.
Em especial à minha avó Edmunda (in memoriam), por sua imensa importância na minha vida.

Agradecimentos

Agradeço a Deus, por ter me criado, me conceder sabedoria e guiar todos os meus caminhos. A todos os professores que dispenderam seu tempo para me auxiliar e me proporcionar conhecimento. Em especial ao Prof. Dr. Josué Silva de Morais por seu incentivo e apoio e ao Prof. Dr. Aniel Silva de Morais por ter aceitado o desafio de orientar o presente trabalho, pelo suporte e auxílio nas correções. Também aos professores que prontamente se dispuseram a participar desta banca.

Aos meus pais, que não mediram esforços para me dar todo o apoio e auxílio durante esta caminhada acadêmica. Ao meu irmão, minha inspiração de foco e determinação. E a toda a minha família, em especial à minha avó Edmunda (*in memoriam*) com quem eu gostaria de ter compartilhado tantos momentos como este, mas que infelizmente não foi possível realizar este desejo.

Agradeço ainda aos meus amigos e colegas que me apoiaram e me suportaram, com tanto carinho. Em especial ao Rodrigo, que foi fundamental na realização deste trabalho, doando seu tempo, conhecimento e paciência para me auxiliar neste processo.

Agradeço ainda à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte financeiro deste trabalho. A todos que direta ou indiretamente fizeram parte da minha vida e da minha formação, expresso aqui o meu agradecimento.

"O perigo de verdade não é que
computadores passem a pensar como humanos,
mas sim que humanos passem a pensar como computadores".
(Sydney Harris)

Resumo

Baseado na crescente utilização de manipuladores robóticos principalmente nas indústrias, surgiu-se a necessidade de inserir neles a capacidade de auto-otimização e reação ao ambiente. Assim, o presente trabalho propõe a criação de um algoritmo generalizado para diferentes anatomias de manipuladores, que seja capaz de mapear o ambiente, calculando um trajeto mínimo entre dois pontos desejados, utilizando movimentos suaves de junta e evitando colisões com possíveis obstáculos estáticos ou dinâmicos.

Este algoritmo, denominado Otimizador de Caminhos para Manipuladores (POM), utiliza a fusão do método de Denavit-Hartenberg com os algoritmos D* Lite e CCD, com algumas modificações realizadas neles a fim de se adequarem ao funcionamento dos manipuladores. Os resultados obtidos a partir de diversos experimentos relatados neste documento comprovam a eficácia do método, apesar de ainda possuir algumas vulnerabilidades em relação ao mapeamento e definição de nós vizinhos.

Palavras-chave: Inteligência artificial. Manipuladores robóticos. Otimização. Planejamento de caminho.

Abstract

Based on the increasing use of robotic manipulators mainly in industries, the need arose to insert in them the capacity of self-optimization and reaction to the environment. Thus, the present work proposes the creation of a generalized algorithm for different anatomies of manipulators, which is able to map the environment, calculating a minimum path between two desired points, using smooth joint movements and avoiding collisions with possible static or dynamic obstacles.

This algorithm, called Path Optimizer for Manipulators (POM), uses the fusion of the Denavit-Hartenberg method with the D* Lite and CCD algorithms, with some modifications made to them in order to adapt to the manipulators' operation. The results obtained from various experiments reported in this document prove the effectiveness of the method, although it still has some vulnerabilities in relation to the mapping and definition of neighboring nodes.

Keywords: Artificial intelligence. Otimization. Pathfinding. Robotic manipulators.

Lista de figuras

Lista de tabelas

Lista de algoritmos

Lista de abreviaturas e siglas

2D	Que possui duas dimensões
3D	Tridimensional
ABC	Colônia Artificial de Abelhas
ACO	Otimização por Colônia de Formigas
AG	Algoritmo Genético
AHP	Processo Hierárquico analítico
BA	Algoritmo de Morcegos
BAP	*Bottleneck Assignment Problem*
BBO	Otimização baseada em Biogeografia
BGMR	*Bayesian Gaussian Mixture Regression*
BIP	Programação Inteira Binária
BP	*Backpropagation*
CCD	*Cyclic Coordinate Descent*
CS	Algoritmo de busca do pássaro cuco
DH	Método de Denavit-Hartenberg
EPSO	Otimização Evolucionária por Enxame de Partículas
FA	Algoritmo de Vagalumes
G.L.	Graus de Liberdade
GPR	Regressão por Processo Gaussiano
GSA	Algoritmo de busca gravitacional
k-NN	k vizinhos mais próximos
LMA	Método de Levenberg-Marquardt
MAP	*Method of Approximate Programming*
MH	Métodos Heurísticos

MILP	Programação Linear Inteira Mista
MLP	Perceptron Multicamadas
MN	Métodos Numéricos
MTL	*Metric Temporal Logic*
NLP	Processamento de Linguagem Natural
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
POM	*Path Optimizer for Manipulators*
PRM	*Probabilistic Roadmaps*
PSO	Otimização por Enxame de Partículas
RNA	Redes Neurais Artificiais
ROS	Sistema Operacional para Robôs
RRG	*Rapidly-exploring Random Graph*
RRT	*Rapidly-exploring Random Trees*
SLAM	Localização e Mapeamento Simultâneos
TLBO	*Teaching-learning Based Optimization*

Lista de símbolos

α	Valor da junta de torção
θ	Valor da junta de rotação
d	Valor da junta prismática
lb	Valor do limite inferior da junta
ub	Valor do limite superior da junta
$\mathcal{J}$	Valor do deslocamento da junta
dj	Passo de deslocamento da junta
rc	Coeficiente de resolução
U	Lista prioritária do D* Lite
T_i	Junta de torção inserida em i
R_i	Junta de rotação inserida em i
d_i	Junta de deslocamento inserida em i
L_i	Elo em i
p_m	Probabilidade de mutação do AG
p_c	Probabilidade de cruzamento do AG
$\mathcal{E}_p$	Erro de posição
$\mathcal{A}$	Acurácia
s_{start}	Nó atual onde se encontra o efetuador
$\overrightarrow{jt}$	Vetor traçado da junta ao ponto desejável
$\overrightarrow{je}$	Vetor traçado da junta ao efetuador
f_{dist}	Distância total do trajeto
f_{joint}	Deslocamento total das juntas

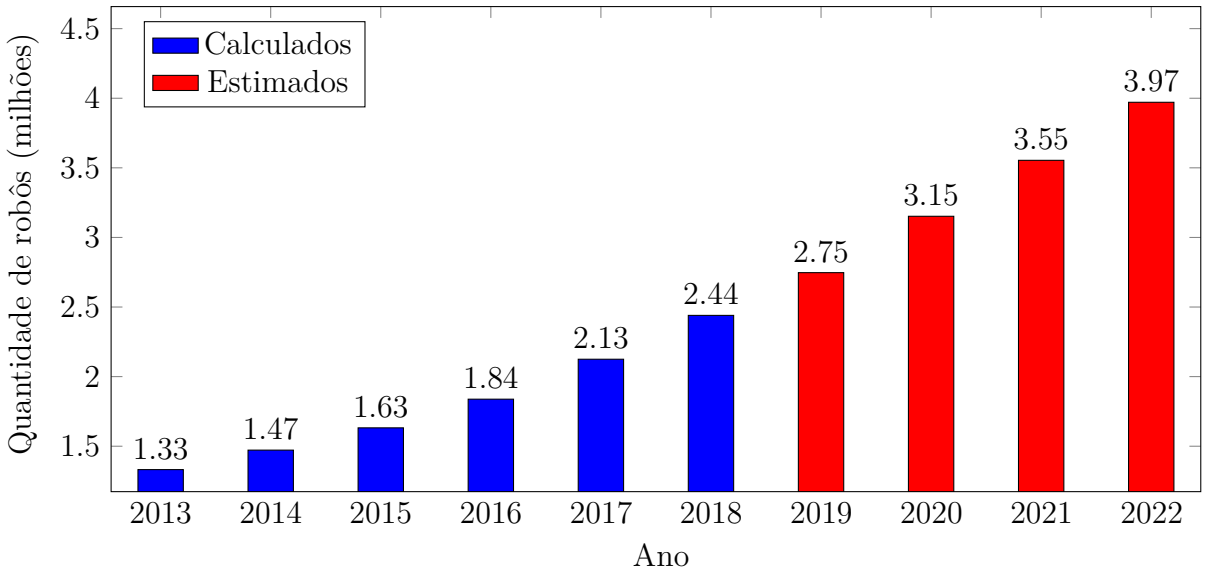
Sumário

1 | Introdução

É inegável a importância da robótica atualmente em diversas áreas, principalmente nas indústrias onde o uso de manipuladores robóticos é essencial a fim de agilizar e padronizar as produções, dentre outras vantagens. O relatório da Federação Internacional de Robótica apresentado em 2019 em sua conferência em Shanghai, afirma que em 2018, existiam 2,440 milhões de robôs em operação em indústrias mundialmente. (International Federation of Robotics, 2019)

A previsão dada por eles, retratada no gráfico da Figura 1, é de que em 2022 este valor tenha aumentado para quase 4 milhões. Ademais, conforme um levantamento realizado pela *Oxford Economics*, estima-se que até o ano de 2030 os robôs se tornem fundamentais na produção alcançando 8,5 % da força de manufatura global. Ainda de acordo com eles, isto aumentará o PIB (produto interno bruto) global em cinco trilhões de dólares.

Figura 1 – Gráfico da quantidade de robôs em operação mundialmente.



Fonte: Adaptado de International Federation of Robotics (2019)

A perspectiva da International Federation of Robotics (2019) para o futuro inclui a inserção de técnicas de aprendizado de máquina e inteligência artificial nos robôs, o

que permitiria que estes sintam o ambiente e respondam a eles. Um tema relacionado ao que a International Federation of Robotics (2019) espera, e que vem sendo estudado nas últimas décadas, como será aprofundado no Capítulo 2, é o planejamento de caminhos utilizando algoritmos de otimização.

A realização do planejamento de caminhos em manipuladores é complexa, iniciando-se na dificuldade de mapeamento em ambientes 3D abordada por Yang et al. (2016). Outro fator que torna a realização deste planejamento complexo é a dificuldade de generalização do algoritmo para funcionar com diferentes anatomias de manipuladores.

Isto ocorre pois a modelagem cinemática de um manipulador, como afirma Pazos (2002), é totalmente dependente de sua anatomia. Assim, ao realizar qualquer tipo de alteração nesta, é necessário modelá-lo novamente, o que é um trabalho que possui um custo relativamente alto em relação ao tempo.

Assim, a generalização do algoritmo para que funcione em qualquer anatomia é uma etapa complexa. Da mesma forma, a capacidade do manipulador de desviar de obstáculos é essencial principalmente em ambientes industriais. O desvio de obstáculos deve abranger tanto obstáculos estáticos quanto dinâmicos, sendo que o último pode auxiliar inclusive em casos onde existem pessoas no ambiente de trabalho do manipulador.

Motivado pela importância da problemática mencionada anteriormente, este trabalho propõe um algoritmo, denominado POM (Otimizador de Caminhos para Manipuladores, do inglês *Path Optimizer for Manipulators*), que dados uma anatomia de manipulador, suas restrições e os pontos de partida e destino, retorna um trajeto entre eles. Este trajeto deve ser mínimo no espaço cartesiano, mantendo a suavidade nos movimentos de junta entre os pontos de controle calculados por ele. O algoritmo deve ainda ser capaz de evitar obstáculos tanto estáticos quanto dinâmicos em ambientes 3D.

Este algoritmo utiliza para a modelagem cinemática o método de Denavit e Hartenberg (1955), calculando a matriz de transformação homogênea a partir dos movimentos de cada elemento inserido na anatomia do manipulador. Para o cálculo do trajeto será analisada a eficácia do Algoritmo Genético, do algoritmo multiobjetivo NSGA-II e do D* Lite, bem como as modificações realizadas para o algoritmo final proposto.

Finalmente, para o cálculo das coordenadas de junta do manipulador a cada movimento, será utilizado o algoritmo de resolução do problema da cinemática inversa, denominado CCD. Algumas alterações foram necessárias para solucionar algumas complicações que surgiram durante o processo de desenvolvimento, as quais serão abordadas no decorrer do documento.

O Capítulo 2 relata as pesquisas realizadas nas últimas décadas relacionadas ao tema abordado neste trabalho. Juntamente com elas são explanados os itens que embasaram a pesquisa aqui proposta e quais autores comprovam a viabilidade de cada etapa do trabalho.

Este capítulo expõe as abordagens utilizadas por cada autor em suas pesquisas. A partir da análise dos benefícios de algumas abordagens e das vulnerabilidades descobertas por cada trabalho exposto, embasa a viabilidade de cada item. Ele também auxilia a elucidar a importância do trabalho aqui proposto.

Em seguida, o Capítulo 3 trata da fundamentação teórica para cada um dos métodos e algoritmos utilizados durante as etapas da realização do trabalho. Estes estágios de desenvolvimentos são descritos no Capítulo 4 em ordem cronológica de realização.

Posteriormente, o Capítulo 5 relata os experimentos realizados e os analisa em relação aos objetivos aqui propostos. Cada experimento se relaciona com algum dos objetivos específicos, testando a efetividade do algoritmo.

Por fim, o Capítulo 6 conclui o trabalho, discorrendo sobre os objetivos atingidos e sobre as vulnerabilidades do algoritmo proposto. Nele ainda são sugeridas atividades futuras relacionadas às vulnerabilidades deste algoritmo.

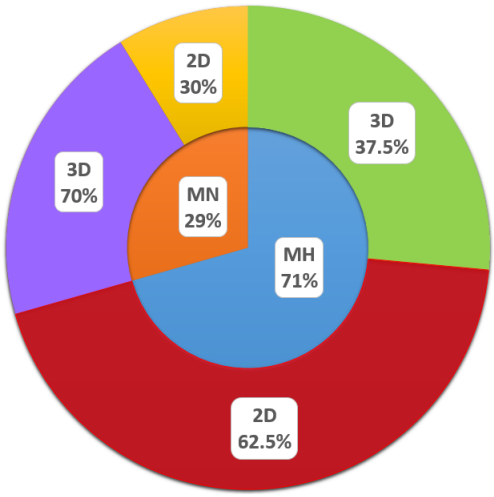
2 | Estado da Arte

Dede o século passado, diversos autores demonstraram interesse pelo estudo de planejamento de caminho em manipuladores, dentre eles Goldenberg e Lawrence (1985), Tan e Potts (1988) e Voliotis, Panopoulos e Christodoulou (1990). Em sua maioria, estes autores utilizavam métodos matemáticos e numéricos clássicos para a estimação do menor caminho, com menor tempo de percurso.

No início dos anos 90, Pobil e Serna (1993) propõem o uso de inteligência artificial com a finalidade de planejar o caminho de um manipulador com desvio de obstáculos. Eles citam Zhu e Latombe (1991) como os pioneiros na utilização dos métodos heurísticos em manipuladores robóticos. Desde então, existem inúmeros estudos com a finalidade de planejar caminhos considerando diversos objetivos e distintos modelos de manipuladores.

O gráfico da Figura 2 relaciona o tipo de método e de ambiente utilizados pelos 54 autores que serão utilizados neste capítulo como forma de embasamento. Nele, pode-se notar que a maioria dos autores que utilizam métodos numéricos (MN), optam por trabalhar no ambiente 3D já que com um tipo fixo de manipulador e devido à base destes métodos ser matemática, trabalhar neste ambiente não apresenta grande complexidade.

Figura 2 – Relação entre os métodos e ambientes utilizados pelos autores abordados neste capítulo.



Fonte: Autoria própria

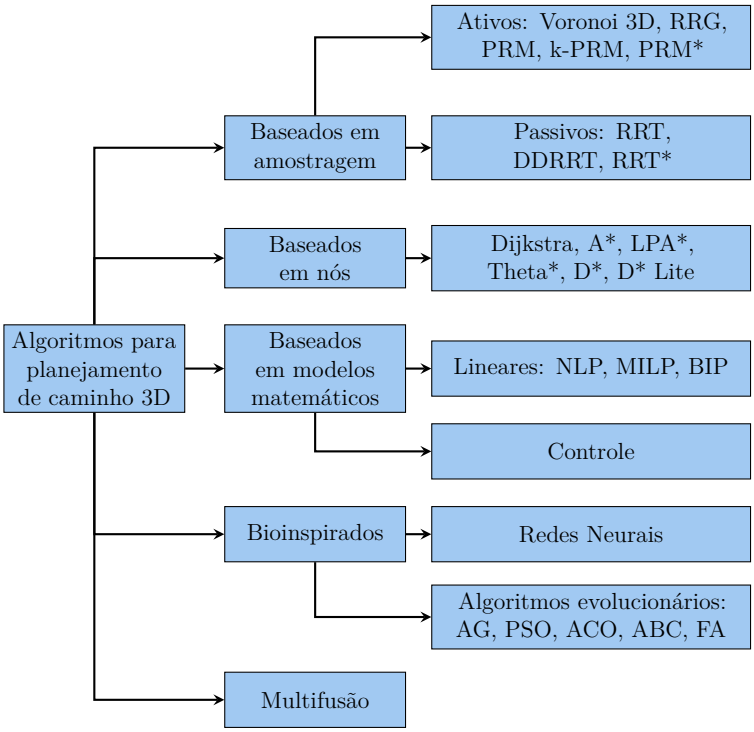
De maneira oposta, ainda retratado pela Figura 2, os autores que trabalham com métodos heurísticos (MH) em sua maioria, optam por trabalhar em ambientes 2D por ser menos complexo. Em alguns casos, ao se utilizar ambiente 3D, uma parte é calculada com métodos numéricos ou matemáticos para ser utilizada como função objetivo do método heurístico, reduzindo sua complexidade. O gráfico da Figura 2 é criado a partir da Tabela 15 do Apêndice A.

Algfoor, Sunar e Kolivand (2015) defendem a importância da utilização de ambientes 3D, já que estes se assemelham mais ao ambiente real. Esta ideia é compartilhada por Yang et al. (2016), os quais alertam que o ambiente 2D não é qualificado para lidar com as complexidades e incertezas do ambiente real.

Porém, ainda de acordo com Yang et al. (2016), ao se utilizar o ambiente 3D, a dificuldade cresce exponencialmente com as restrições cinemáticas. Outra dificuldade apontada pelo autor ao se trabalhar com este ambiente é que ele é NP-difícil, portanto não existe solução comum. Neste trabalho será abordado o planejamento de caminho em ambientes 3D.

Os algoritmos para planejamento de caminhos em ambientes 3D são divididos em cinco tipos principais, como representado no fluxograma da Figura 3. Neste capítulo serão abordados autores que utilizam tipos diferentes de algoritmos para encontrar caminhos ou minimizar erro de posição.

Figura 3 – Representação dos tipos de algoritmos utilizados para o planejamento de caminho em ambientes 3D.



Fonte: Adaptado de Yang et al. (2016).

Os algoritmos baseados em amostragem necessitam de uma representação matemática para descrever o espaço de trabalho, segundo Kim et al. (2019). Com base nela, o ambiente é mapeado ou apenas realiza uma busca aleatória para obter um caminho viável. Os algoritmos baseados em amostragem precisam de todas as informações de amostragem da área de trabalho para escapar dos mínimos locais.

Estes algoritmos baseados em amostragem são divididos no artigo de Yang et al. (2016) em ativos e passivos. De acordo com o autor, os ativos são os que podem alcançar o melhor caminho possível para o objetivo independentemente. Já os passivos necessitam de uma combinação de algoritmos de busca para escolher o melhor caminho possível no mapa da rede, onde existem muitos caminhos possíveis. Qin e Henrich (1996) e Park, Park e Manocha (2018) utilizam este tipo de algoritmo em suas abordagens.

O segundo tipo de algoritmo é o baseado em nós, o qual é utilizado por Son e Lee (2012) e é o principal algoritmo abordado no presente trabalho. Os algoritmos baseados em nós realizam exploração a partir do gráfico decomposto. Este tipo de método sempre pode encontrar um caminho ótimo de acordo com a decomposição certa, de acordo com Yang et al. (2016).

Os algoritmos de Dijkstra, A* e D*, segundo Algfoor, Sunar e Kolivand (2015) são tradicionalmente classificados como planejamento ótimo discreto, algoritmos de mapas de caminhos (*roadmaps*, em inglês), algoritmos de busca e assim por diante. No entanto, todas estas classificações apenas expressam que estes algoritmos lidam com otimização discreta com base na decomposição do *grid*.

A terceira classificação, representada na Figura 3, são os algoritmos baseados em modelos matemáticos. Estes incluem algoritmos lineares e de controle ótimo, e são utilizados por alguns autores como Tan e Potts (1988) e Barakat, Gouda e Bozed (2016) no planejamento de caminho em manipuladores. Estes algoritmos trabalham tanto com a modelagem do ambiente com suas restrições cinemáticas, quanto com o sistema dinâmico. Estas restrições são vinculadas à função de custo para obter uma solução otimizada.

O quarto tipo de algoritmos são os bioinspirados, os quais dominam a literatura tanto na minimização de erro de posição quanto no planejamento de caminhos para manipuladores. Segundo Yang et al. (2016), eles imitam o comportamento biológico para resolver problemas em detrimento do processo de construção de modelos complexos de ambiente para procurar um caminho quase ideal com base em abordagens estocásticas.

Além disto, algoritmos baseados em modelos matemáticos geralmente falham ou caem no mínimos locais em solução de problemas NP-difíceis, com grande número de variáveis e funções objetivas não lineares, o que é solucionado pela utilização de algoritmos bioinspirados. O algoritmo genético (AG) é o método mais famoso, e é abordado em algumas etapas do presente trabalho. Ele é utilizado pela maioria dos autores citados

neste capítulo, como Toogood, Hao e Wong (1995), Jiang, Yao e Zhou (2018), Kazem, Mahdi e Oudah (2008), Yue et al. (2002) e Liu, Chen e Tsai (2007).

Outros algoritmos evolucionários são utilizados por diversos outros autores em suas abordagens, dentre eles Yeasmin, Shill e Paul (2017), Garg e Kumar (2002) e Savsani, Jhala e Savsani (2013). As redes neurais também são combinadas com os algoritmos evolucionários por alguns autores como KöKer (2013) e Varedi-Koulaei e Mokhtari (2018).

A última classificação dos algoritmos de planejamento de caminho em ambientes 3D, retratados pelo fluxograma da Figura 3, são os algoritmos de multifusão. Esta abordagem é classificada por Yang et al. (2016) como a combinação de vários algoritmos para obter um melhor desempenho. Autores como Duka (2014), Zhang et al. (2017) e Park, Park e Manocha (2018) utilizam a abordagem de multifusão.

Em ambientes desconhecidos, com obstáculos dinâmicos ou estáticos, nem todas as abordagens tradicionais de planejamento podem garantir independentemente um custo mínimo de convergência ou eficiência computacional, de acordo com Yang et al. (2016). Os algoritmos baseados em amostragem não podem gerar um caminho ideal próprio ou saltam para o mínimo local; e os algoritmos baseados em nós precisam de informações prévias do ambiente.

Ainda segundo Yang et al. (2016), os algoritmos baseados em modelos matemáticos tendem a ser demorados e incapazes de resolver problemas NP-difíceis com ambientes variados; e os bioinspirados variam seu desempenho com o diagrama do modelo e possuem uma alta complexidade de tempo. Assim, os pesquisadores tentam introduzir uma combinação de diferentes abordagens para formar um algoritmo ideal de busca rápida e global.

No estudo dos manipuladores, os algoritmos baseados em modelos matemáticos são os mais utilizados em ambientes 3D, sendo que suas abordagens em 2D, em geral, datam do fim do século passado. Exemplos disto são o trabalho de Tan e Potts (1988) que utilizam os métodos MAP e NLP para calcular trajeto em ambientes livres de obstáculos e o de Kim e Shin (1985) que propõem um método de minimização de tempo baseado no lagrangiano e DH.

Bessonnet e Lallemand (1990) utilizam o princípio mínimo de Pontryagin, enquanto Chen (1992) opta por utilizar a linearização por realimentação, ambos com a finalidade de traçar um trajeto em ambientes 2D livres de obstáculos. Em ambientes 3D, Yetim (2009) e Mohammed e Sunar (2015) utilizam métodos baseados em modelos matemáticos para minimizar erro de posição também em ambientes livres de obstáculos.

Yetim (2009) emprega métodos da cinemática direta e da inversa, juntamente com a matriz jacobiana. Já Mohammed e Sunar (2015), calculam a matriz de transformação homogênea pelo método de DH e realizam a minimização utilizando o objeto SerialLink

do software Matlab.

Para a geração de trajetos em ambientes livres de obstáculos em 3D, Goldenberg e Lawrence (1985) propõem um método baseado em modelos matemáticos chamado MINIPACK-1. Voliotis, Panopoulos e Christodoulou (1990) também propõem o método Simplex dado pela fusão de métodos deste tipo de algoritmo.

Barakat, Gouda e Bozed (2016) utilizam o método LMA e a matriz jacobiana, enquanto Garriz e Domingo (2019) calculam a matriz de DH e a combinam com o método de Kalman a fim de encontrar um trajeto otimizado no ambiente livre de obstáculos. De forma distinta, Seereeram e Wen (1995) trabalham com obstáculos estáticos no ambiente e utilizam o método de Newton-Raphson para planejar o trajeto.

Em geral, os autores trabalham com um modelo fixo de estrutura, já que ao alterá-la, seria necessário refazer os cálculos para a nova estrutura. Por este motivo, estudos recentes optam por realizar o planejamento da estrutura ideal do manipulador a fim de que realize determinada tarefa, como é o caso de Whitman e Choset (2018). Porém, este tipo de abordagem nem sempre gera um manipulador fisicamente realizável, acarretando diversas desvantagens a este tipo de planejamento.

Devido à robustez dos métodos baseados em modelos matemáticos, existem trabalhos que já abordam robôs mais complexos, como os antropomórficos. Saha e Julius (2017) combinam os métodos MTL, Descida do Gradiente (*Gradient Descent*, em inglês) e MILP para minimizar a quantidade de movimentos realizados por um robô antropomórfico ao realizar determinada tarefa.

Gavilanes et al. (2018) também trabalham com um robô antropomórfico e realizam a modelagem do mesmo utilizando o software SolidWorks. Os autores utilizam a biblioteca SimMechanics do software Matlab no Simulink para minimizar o erro de posição com base na matriz de transformação homogênea de DH. Chu, Hu e Zhang (2017) utilizam interpolação polinomial e a pseudoinversa de Moore-Penrose para realizar o planejamento de trajetória de um robô de manobras espaciais com manipuladores, evitando colisões.

Como já citado anteriormente, os algoritmos bioinspirados sanam algumas falhas dos baseados em modelos matemáticos, por este motivo eles são bastante explorados na literatura. A minimização de erro de posição é o mais simples dos objetivos a serem alcançados para manipuladores em ambientes 3D. Liu, Chen e Tsai (2007) propõem a utilização de AG para otimizar o erro de posição com um menor deslocamento das juntas.

Já KöKer (2013) realiza uma combinação de AG com RNA de forma que o AG encontre os pesos ideais para treinar a rede usada na minimização. Varedi-Koulaei e Mokhtari (2018) combinam os métodos PSO, MLP com BP, para realizar seguimento de trajetória por meio da minimização do erro de posição em cada ponto de um ambiente 2D.

A fim de testar a capacidade de um algoritmo para a geração de trajetos, é comum

utilizar mapas estilo labirintos ou carrinhos dentro de salas, em ambientes 2D. Sudhakara et al. (2016) calculam uma trajetória ótima em labirinto 2D usando o algoritmo bioinspirado Colônia de Formigas melhorado que tem menor esforço computacional. Para suavizar a trajetória, é utilizada a spline cúbica que além de tornar os movimentos do robô mais suaves, diminui significativamente o tempo gasto para chegar ao destino.

Na tese de Roshanineshat (2018) o algoritmo BBO é utilizado para otimizar os parâmetros do método probabilístico de planejamento de trajetória para território 2D desconhecido. O robô tem um radar e um sensor laser e varia a velocidade conforme a presença de obstáculos.

Diversos autores realizam o planejamento de trajeto para manipuladores em ambientes livres de obstáculos com algoritmos bioinspirados utilizando estruturas determinadas. Savsani, Jhala e Savsani (2013) propõem a minimização de trajetória para um robô 3R levando em conta três parâmetros de minimização: o tempo de trajeto, a distância do trajeto e a soma dos deslocamentos de junta. A minimização é realizada usando os algoritmos TLBO e ABC e os resultados são comparados entre si e com os resultados obtidos por Kazem, Mahdi e Oudah (2008) utilizando AG.

Garg e Kumar (2002) otimizam um manipulador 2R com relação ao torque. São dois robôs colaborativos idênticos que carregam juntos determinado objeto. Os autores comparam o AG com o SA, sendo este último constatado como uma opção melhor, já que converge bem mais rápido e com menor esforço computacional. Yano e Toyoda (1999) também utilizam um manipulador 2R e optam por um AG multiobjetivo para garantir um trajeto com mínima trajetória com um mínimo deslocamento das juntas.

Os autores Yue et al. (2002) propõem um novo método generalizado para a geração de trajetória ponto a ponto que analisa diversas variáveis e restrições. O objetivo do AG é gerar uma boa trajetória com mínima vibração. Os indivíduos utilizados são as possíveis trajetórias dados os pontos inicial e final e os limites de ângulo e aceleração.

Outros autores que utilizam um braço 3R, são Sharma, Singh e Singh (2011) que tem como objetivo, assim como Yano e Toyoda (1999), encontrar a melhor trajetória com menores valores de juntas. Porém sua função de aptidão é a minimização da energia dispendida no movimento de cada junta.

Nota-se que todos os autores supracitados que abordam trajeto em manipuladores com algoritmos bioinspirados, optam por trabalhar com manipuladores que possuam seu espaço de trabalho apenas em 2D, devido a sua maior simplicidade. De forma oposta, Števo, Sekaj e Dekan (2014) usam o método da cinemática direta para modelar um robô de 6 links modelo ABB IRB 6400FHD. Eles utilizam a combinação de pelo menos dois parâmetros de otimização como função de aptidão do AG, podendo ser energia consumida, tempo de operação, movimentos de rotações e distância euclideana do ponto desejado.

Alguns autores para garantir um melhor planejamento do caminho, optam pela fusão dos algoritmos bioinspirados com algum outro tipo de algoritmo. Alguns desses autores são Bianco e Piazzi (1999), que propõem um método global de otimização baseado em funções polinomiais cúbicas. Este método encontra a função que será usada como objetivo do algoritmo genético.

Similarmente, Banga, Singh e Kumar (2007) usam um manipulador 3R cujos possíveis movimentos são os indivíduos do AG. A aptidão de cada indivíduo está ligada a três fatores: movimento, fricção e mínima vibração. Os autores utilizam o método AHP para relacionar o grau de importância de cada atributo para cada uma das juntas.

Seguindo a mesma ideia de fusão de algoritmos, Duka (2014) utiliza rede neural do tipo feedforward treinada com o algoritmo LMA para seguir determinada trajetória com um braço robótico 3R. A trajetória é encontrada treinando uma rede neural com diversos ângulos e posições. Para criar os dados de treinamento, o autor usa os valores encontrados na cinemática direta para cada vetor de ângulos. O algoritmo LMA é usado para minimização do erro.

Alguns autores também abordam o planejamento de caminho com obstáculos estáticos. Um exemplo é o trabalho de Tian e Collins (2004) que usam o AG combinado com o método de interpolação de Hermite para gerar possíveis pontos de trajetória de forma que o manipulador 2R desvie de obstáculos.

Com objetivo semelhante, Kazem, Mahdi e Oudah (2008), Savsani, Jhala e Savsani (2014), Yeasmin, Shill e Paul (2017) e Jiang, Yao e Zhou (2018) trabalham com a mesma anatomia de manipulador 3R. O primeiro deles utiliza um AG para otimizar a trajetória em termos de espaço e tempo e sem exceder um torque predeterminado. Já Savsani, Jhala e Savsani (2014) compara os algoritmos ABC, AG, BA, BBO, CS, FA, GSA e TLBO em ambientes livres de obstáculos e com obstáculos estáticos.

Yeasmin, Shill e Paul (2017) propõem o método EPSO para atingir o objetivo mencionado anteriormente, considerando parâmetros como torque e massa. Por fim, Jiang, Yao e Zhou (2018) realizam a modelagem utilizando DH e calculam o caminho usando AG.

Em ambientes 3D, Toogood, Hao e Wong (1995) buscam otimizar um manipulador de 3 graus de liberdade usando AG, com prevenção de colisões. De forma similar, Nearchou (1998) tem como objetivo minimizar o erro de posição e o deslocamento das juntas sem colidir com os possíveis obstáculos dentro do espaço de trabalho. Ele utiliza o AG para fazer a minimização da função em um manipulador do tipo Puma 566.

Por fim, Mahdavian et al. (2015) realizam minimização de energia para um braço de um humanoide, o qual possui 4 graus de liberdade, utilizando a toolbox de AG do software Matlab. A modelagem é feita a parte usando o método de funções polinomiais de Lagrange.

Dentre os algoritmos baseados em nós, os mais conhecidos e utilizados são os que derivam do algoritmo de Dijkstra (1959). Estes algoritmos são comumente utilizados para traçar rotas ou em jogos, conforme Algfoor, Sunar e Kolivand (2015). Na área da robótica, Edan et al. (1991) o utiliza para minimizar tempo de trajeto de um robô para colher frutas na árvore.

O algoritmo A* é derivado do de Dijkstra e é utilizado por Stolle e Atkeson (2006) para encontrar um trajeto subótimo para um labirinto de bola de gude. Os autores combinam trajetos encontrados por qualquer planejador de trajeto em uma biblioteca e utiliza o algoritmo *inflated-heuristic A** para definir o trajeto final.

O algoritmo D* Lite, que é o principal abordado no presente trabalho, permite traçar caminhos em ambientes dinâmicos. Pradeep, Medioni e Weiland (2010) o combina com o método SLAM para criar uma vestimenta de alerta para pessoas com deficiência visual. Eles utilizaram processamento de imagem para identificar objetos e lugares livres para passagem e o D* Lite para calcular caminhos mais rápidos quando está no modo guia.

Outra utilização do D* Lite é na otimização de codons. O trecho de DNA a ser sintetizado é transformado em um gráfico acíclico direcionado ponderado e a seguir é aplicado o algoritmo D* Lite para otimização tendo cinco objetivos como função heurística. O algoritmo é provado robusto e um bom otimizador neste caso.

Rajasekaran et al. (2017) utiliza processamento de imagem para definição de miçangas que serão movimentadas com o uso de pinças ópticas e dos obstáculos. A trajetória é calculada usando controle preditivo e em casos onde há obstáculos ao longo da trajetória calculada, a mesma é recalculada usando D* Lite. A utilização do D* Lite para este tipo de abordagem é constatada satisfatória em relação a outros autores que utilizam outros algoritmos, como Shaw, Chizari e Hopkins (2018) que utilizam BAP.

Para testar a eficácia do algoritmo A* em ambientes 2D, Narayan et al. (2014) propõem o desenvolvimento de um veículo não tripulado com geração de trajetória usando o A*. A presença de obstáculos é determinada por meio de sensor ultrassônico que faz com que o carro desvie do obstáculo e trace a rota novamente.

Igualmente, Gupta, Umrao e Kumar (2016) utilizam o sistema ROS no software Gazebo para fazer testes de planejamento de trajetória usando Relaxed A* e D* Lite. Os obstáculos são mapeados e salvos como um mapa de ocupação com probabilidade de cada ponto estar sendo ocupado. Funciona para o robô determinado e com obstáculos estáticos apenas.

Utilizando o algoritmo D* Lite proposto por Koenig e Likhachev (2002), Barac (2010) apresenta um documento de especificação de um carro projetado para navegação autônoma em locais previamente mapeados. Ele utiliza o D* Lite como algoritmo de

planejamento de trajetórias para adaptação a possíveis obstáculos encontrados no trajeto. Caso haja obstáculos, o robô para e espera até que a nova rota seja encontrada.

Son e Lee (2012) utiliza o algoritmo para um caso com maior complexidade, já que visa traçar trajetórias para robôs cooperativos de forma estática combinado com um modelo R-Object. Portanto, a qualquer mudança como número de robôs na arena ou obstáculos, os robôs param até todos os caminhos serem recalculados.

Na última década foram propostas modificações do algoritmo D* Lite na tentativa de sanar algumas dificuldades advindas da utilização do algoritmo original. Um exemplo é quando se torna necessário alterar os pontos de início e fim do trajeto. Por isto, Sun, Yeoh e Koenig (2010) propõem o algoritmo MT-D* Lite que é um melhoramento do D* Lite em rapidez para casos onde há mudança de pontos de partida e objetivo dinamicamente.

Outro aprimoramento do algoritmo original, é o proposto por Al-Mutib et al. (2011) , que é um algoritmo modificado do D* Lite para multiagente e tempo real. O mapa de obstáculos temporiza cada obstáculo, sendo os obstáculos estáticos com $t = \infty$. Os testes realizados não mostraram piora de desempenho com o aumento do número de robôs.

Com relação ao tratamento de obstáculos, Ganapathy, Yun e Chien (2011) propõe um melhoramento do D* Lite testado via simulação no Matlab e em tempo real usando o Team AmigoBot. Os cinco melhoramentos realizados neste algoritmo são: (i) prevenção de esbarrar na quina de um obstáculo, (ii) desviar de obstáculos complexos, (iii) impedir que o robô passe na junção diagonal de dois obstáculos, (iv) se necessário criar paredes virtuais para não cair em obstáculos em forma de U, e (v) remover partes desnecessárias do caminho para garantir menor trajetória.

Finalmente, Przybylski e Putz (2017) propõem uma modificação do D* Lite com a utilização de árvores com nós pais para que ao invés de existir inconsistência, haja uma reinicialização a cada novo ponto incrementado no caminho. Como pode-se observar, mesmo sendo um algoritmo robusto, o D* Lite possui diversas limitações que podem ser sanadas através da fusão dele com outros algoritmos.

A ideia da multifusão é utilizada em caminho de manipuladores por Tian e Collins (2004) que combinam o AG com polinômio de Hermite, Mahdavian et al. (2015) que usam o AG com Lagrange, e Zhang et al. (2017) que propõem o algoritmo MVN-RMP-INVM-TCOCM para calcular trajetos suaves com robôs colaborativos. Devido à robustez dos algoritmos do tipo multifusão, eles podem ser utilizados em robôs mais complexos.

Um exemplo disto é o trabalho de Hebert et al. (2015) que criam o robô surrogate que tem os braços e o corpo de manipuladores com 7 G.L. em uma base móvel de esteira. O robô detecta os obstáculos por câmera e laser e usa dois algoritmos: o D* Lite que realiza o planejamento de trajetória global e o end-state planning que projeta o estado

final do robô dado o ponto de chegada do trajeto.

A eficácia dos algoritmos do tipo multifusão é comprovada por Lembono et al. (2020) que comparam a performance em trajetórias ótimas entre três métodos que utilizam base de dados de trajetos prévios: k-NN, GPR e BGMR. A partir destes, os autores criam um método multifusão denominado *ensemble*, que é a combinação dos outros três de forma que os resultados obtidos sejam melhores. Os testes são realizados para uma base PR2 com dois braços e para um robô Atlas.

Motivado pelas vantagens e limitações dos algoritmos supramencionados, o presente trabalho busca utilizar alguns dos algoritmos tratados neste capítulo, a fim de avaliar sua eficácia em traçar caminhos ótimos para diferentes anatomias em presença de obstáculos. Alguns autores propuseram soluções para este problema, como Banga, Singh e Kumar (2007), Kazem, Mahdi e Oudah (2008), Savsani, Jhala e Savsani (2013), Yeasmin, Shill e Paul (2017), Sharma, Singh e Singh (2011), Jiang, Yao e Zhou (2018) e Tian e Collins (2004), os quais serão utilizados para comparação com a solução do algoritmo POM proposto neste trabalho.

3 | Referencial Teórico

Um robô pode ser definido como uma máquina automática programável. Estes podem ser divididos em quatro tipos principais: manipuladores, exploradores, máquinas ferramentas e de uso geral (PAZOS, 2002).

O braço mecânico, segundo Rosário (2005), é um manipulador e possui características antropomórficas, além de um programa que controla seus movimentos no caminho a ser seguido. Este consiste, de acordo com Siciliano et al. (2010), em uma sequência de corpos rígidos denominados elos ou links, os quais são conectados por juntas.

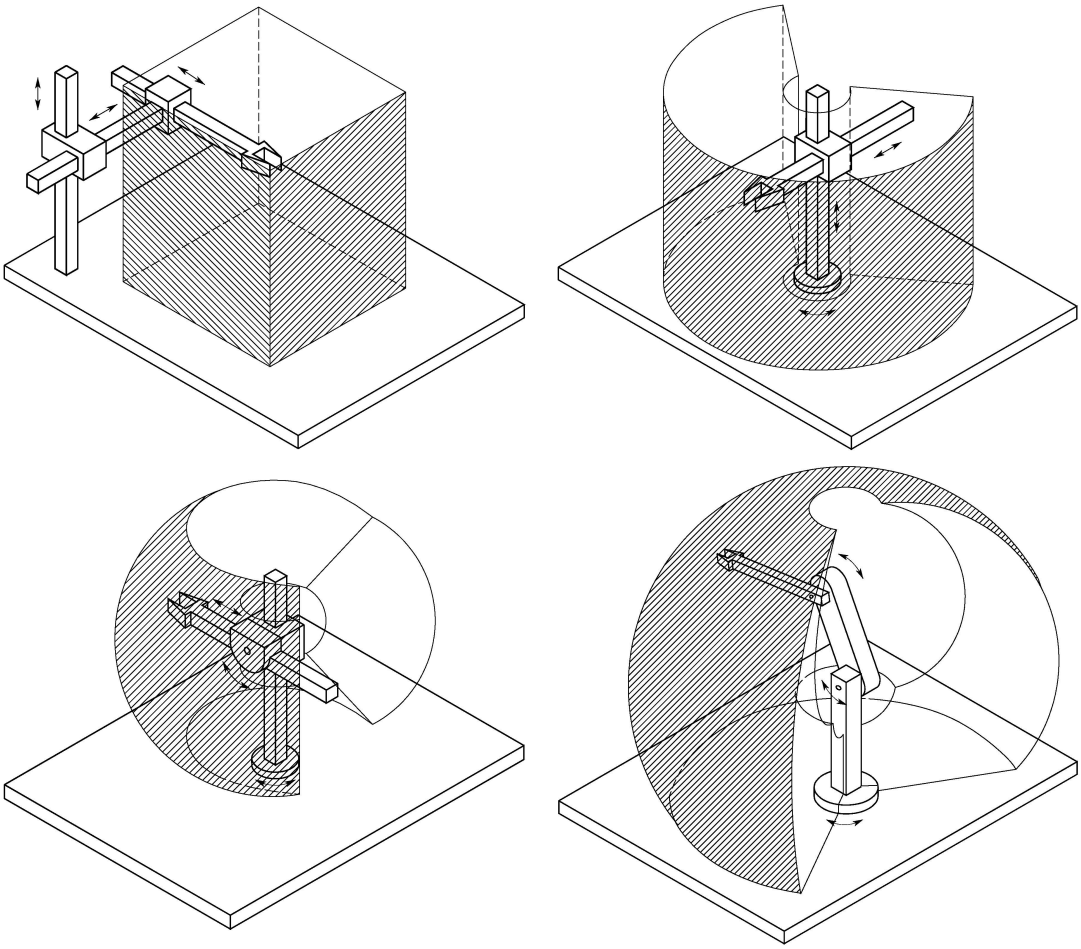
A junta que une os elos lhes permitem movimentar, sendo o braço robótico definido por Pazos (2002) como uma cadeia cinemática aberta de elos interligados por juntas. O autor afirma ainda que o tipo de movimento possível ao elo em relação ao anterior é definido pelo tipo de junta que os interliga. Conforme Siciliano et al. (2010), as juntas podem ser prismáticas ou de revolução, onde a primeira confere movimento de deslocamento linear e a última dependendo da orientação pode ser uma junta de rotação ou de torção.

A quantidade de juntas presentes em um manipulador determina o número de graus de liberdade do robô, segundo Rosário (2005). O autor complementa que cada eixo de movimento do robô é contado como um grau de liberdade. Em conformidade com esta afirmação, Pazos (2002) reitera que quanto maior o número de graus de liberdade, maior é a dificuldade de controle do robô.

A estrutura de juntas e elos do braço robótico determinam o volume do espaço de trabalho deste, como afirmado por Siciliano et al. (2010). O autor define o espaço de trabalho como a denominação da porção do ambiente onde o efetuador alcança, respeitando suas restrições.

A figura 4 representa os espaços de trabalho para quatro diferentes anatomias. No canto superior esquerdo há uma estrutura com três juntas prismáticas que formam um espaço de trabalho em formato de paralelepípedo. Já no canto superior direito, a combinação da junta de torção com duas juntas prismáticas formam um espaço de trabalho cilíndrico.

Figura 4 – Exemplos de espaço de trabalho para diferentes estruturas de manipuladores.



Fonte: Siciliano et al. (2010)

O canto inferior esquerdo da Figura 4 ilustra um modelo esférico gerado pela combinação da junta de rotação com uma de torção e a prismática. O canto inferior direito expõe um espaço de trabalho de um robô antropomórfico que é mais complexo por inserir diferentes tipos de juntas e em maiores quantidades. Nota-se que quanto mais juntas um manipulador possui, mais complexo será seu espaço de trabalho.

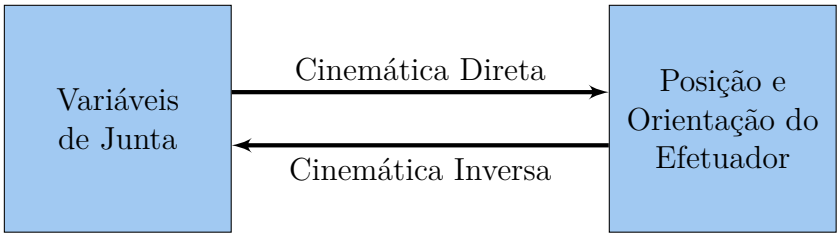
Existe ainda, de acordo com Nof (1999), o chamado espaço de junta. Este é formado por todos os possíveis vetores de valores de junta. O autor afirma ainda que a dimensão deste vetor é igual ao número de graus de liberdade do robô manipulador.

3.1 Cinemática

O estudo dos movimentos de um manipulador é denominado cinemática, como afirma Pazos (2002). Ela é dividida em dois tipos, como elucida a Figura 5: a cinemática direta e a cinemática inversa. Conforme Mohammed e Sunar (2015), a primeira calcula a

posição e orientação no espaço com base nos valores de junta, enquanto a última delas encontra os deslocamentos para as juntas tendo a posição e orientação do efetuador.

Figura 5 – Representação das entradas e saídas para cada método de estudo da Cinemática.



Fonte: Autoria própria

Ambos tipos serão explanados nas subseções 3.1.1 e 3.1.2, bem como alguns métodos associados a cada um deles.

3.1.1 Cinemática Direta

Como mencionado anteriormente e salientado por Nof (1999), a cinemática direta encontra os dados do efetuador com base nas coordenadas de junta. Isto é possível, de acordo com Schilling (1996), devido à chamada matriz de transformação homogênea (T) advinda das rotações e translações realizadas, bem como da anatomia do manipulador.

Segundo Ranky e Ho (1985), o conceito da matriz de transformação foi introduzido por Denavit e Hartenberg (1955) e posteriormente aplicado por outros autores. Craig (2009) complementa afirmando que a matriz T é encontrada através de cálculos algébricos, como representação da posição e orientação do robô com relação ao eixo de coordenadas de referência.

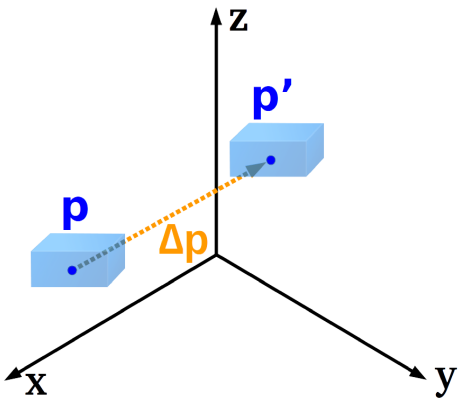
Siciliano et al. (2010) define translação como a relação de certo ponto com o sistema de referência, sendo ela a responsável por revelar a posição de um corpo rígido no espaço. Schilling (1996) endossa esta definição, além de estabelecer o termo rotação como a orientação do corpo rígido advindo dos componentes do vetor unitário entre o eixo de referência deste corpo com relação ao eixo base de referência.

A Equação 3.1 é dada por Ranky e Ho (1985) como a matriz de translação entre um ponto i e o ponto consecutivo $i+1$, sendo p_x, p_y e p_z a posição no sistema de coordenadas cartesiano x, y e z, respectivamente.

$$p_i^{i+1} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{3.1}$$

Na Figura 6 há a representação de um ponto $p = (p_x, p_y, p_z)$ que será transladado em $\Delta p = (\Delta x, \Delta y, \Delta z)$, a fim de chegar ao ponto $p' = (p'_x, p'_y, p'_z)$. Em conformidade com McCloy (2013), para encontrar o ponto p', basta somar a coordenada de p com o valor do deslocamento relativo à coordenada, como descreve a Equação 3.2.

Figura 6 – Representação do movimento de translação no gráfico.



Fonte: Autoria própria

$$p' = \begin{bmatrix} p_x + \Delta x \\ p_y + \Delta y \\ p_z + \Delta z \end{bmatrix} \tag{3.2}$$

Já o processo de rotação é um pouco mais complexo que o anterior e, segundo Schilling (1996) pode ser realizado com o auxílio da regra da mão direita e os ângulos de Euler. Deste modo, o autor afirma que será definida uma matriz de rotação R de dimensão 3x3, que descreve com base nos ângulos de Euler a rotação realizada entre determinado sistema de coordenadas e o sistema de coordenadas de referência.

Para encontrar a matriz R, aplica-se a transformação na matriz identidade também de dimensão 3x3, de acordo com Ranky e Ho (1985). O autor ainda estabelece que as colunas da matriz são as transformações ocorridas com relação ao eixo x, y e z, respectivamente.

Em 3.3 está representada a matriz de rotação do ângulo θ em torno do eixo x. Se o eixo y for rotacionado em sentido anti-horário com um ângulo θ, a transformação ocorrida é representada por 3.4. Ao aplicar a rotação de θ em torno de z, a transformação é descrita pela matriz 3.5, conforme assegura Schilling (1996).

$$R(x, \theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{3.3}$$

$$R(y,\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.4}$$

$$R(z,\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

De acordo com Siciliano et al. (2010), conhecendo as matrizes de translação e rotação, pode-se definir a matriz de transformação homogênea T que mapeia a cinemática direta. Nof (1999) complementa que esta matriz é representada pela Equação 3.6, onde R_i^{i+1} é a matriz 3x3 de rotação e p_i^{i+1} a matriz 3x1 de posição contendo p_x, p_y e p_z. Cada matriz é analisada entre o elemento i e o elemento $i+1$ do manipulador, portanto a notação para a matriz de transformação homogênea é T_i^{i+1} e a notação para a matriz de rotação será utilizada como R_i^{i+1}.

$$T_i^{i+1} = \begin{bmatrix} R_i^{i+1} & p_i^{i+1} \\ [0] & 1 \end{bmatrix} \tag{3.6}$$

Um robô manipulador possui $n+1$ ligamentos, sendo a base numerada como zero e o efetuador como n, de acordo com Pazos (2002). Cada combinação de rotações e translações ocorridas em um dado ligamento, dá origem a uma transformação homogênea. Isto é demonstrado por Tsai (1999), o qual embasa o fato de que quando um manipulador se movimenta, ocorrem n transformações homogêneas.

Segundo Crane e Duffy (2008), para compor o movimento total do manipulador, multiplica-se as matrizes T_i^{i+1} na ordem em que aparecem da base ao efetuador. Isto é representado pela Equação 3.7.

$$T_0^n = \prod_{i=0}^{n-1} T_i^{i+1} \tag{3.7}$$

O método proposto por Denavit e Hartenberg (1955) consiste em descrever o eixo de coordenadas de um elo em relação ao anterior, é um método conciso que resulta em quatro parâmetros: a, α, d e θ. Estes parâmetros são melhor detalhados em Siciliano et al. (2010), Ranky e Ho (1985) e Radavelli et al. (2012).

- a: Comprimento da normal comum;
- α: Ângulo em torno da normal comum de z_{i-1} a z_i;
- d: Distância ao longo de z_{i-1} até a norma comum;

- θ: Ângulo em torno de z_{i-1}, do x_{i-1} até x_i

O método de Denavit-Hartenberg segue o algoritmo 1, definido por Schilling (1996).

Algorithm 1 Algoritmo de Denavit-Hartenberg

1: Numere as juntas de 0 até n desde a base até a ferramenta
2: Determine uma origem de sistema de coordenadas dextrogiras O_0 para a base, alinhando o z_0 com o eixo da junta 1. Escolha arbitrariamente x_0 e complete com y_0 o sistema dextrogiro
3: Incremente o contador i=1
4: **while** i < n **do**
5: Alinhe z_i com o eixo da junta $i + 1$.
6: Localize a origem O_i na interseção dos eixos z_i e z_{i-1}
7: **if** Não se interceptam **then**
8: Use a interseção de z_i com a normal comum entre z_i e z_{i-1}
9: **end if**
10: **if** z_i e z_{i-1} estão no mesmo elo **then**
11: A origem O_i pode ser alocada em qualquer local ao longo deste elo
12: **end if**
13: Selecione a direção de x_i tal que esta seja ortogonal tanto a z_i e z_{i-1}.
14: **if** z_i e z_{i-1} são paralelos **then**
15: x_i tem a direção de z_{i-1} para z_i
16: **end if**
17: Complete o sistema dextrogiro de coordenadas i com o vetor y_i, seguindo a regra da mão direita.
18: i = i+1
19: **end while**
20: Coloque a origem O_n na ferramenta.
21: Alinhe z_n com o z_{n-1}, escolha x_n de forma a ser perpendicular a z_{n-1} e z_n e complete o sistema dextrogiro com o y_n.
22: Coloque i=1.
23: **while** $i \leq n$ **do**
24: Compute θ_i como o ângulo de rotação de x_{i-1} até x_i em torno do eixo z_{i-1}
25: Compute di como distância entre as origens atual e anterior ao longo do eixo $zi - 1$.
26: Compute a_i como a distância entre as origens atual e anterior medida ao longo do eixo x_i.
27: Compute α_i como ângulo de rotação de z_{i-1} até z_i medida em torno de x_i.
28: i= i+1
29: **end while**

A partir do algoritmo 1, é possível encontrar a tabela dos parâmetros de Denavit-Hartenberg. De acordo com Siciliano et al. (2010), ao substituir as incógnitas da matriz T pelos parâmetros da tabela, é possível encontrar a posição e orientação do efetuador.

3.1.2 Cinemática Inversa

A cinemética inversa se propõe, segundo Nof (1999), tendo a posição e orientação do efetuador, encontrar vetores de deslocamento de junta que podem ser usados para chegar ao local desejado. Siciliano et al. (2010) afirma que ela é bem mais complexa que a cinemática direta já que as equações da cinemática inversa são, em sua maioria, não lineares; podem existir múltiplas ou infinitas soluções; além de existirem soluções não admissíveis devido á estrutura do robô.

A primeira ação necessária ao usar a cinemática inversa, de acordo com Craig (2009), é restringir os pontos ao espaço de trabalho do robô, garantindo que seja alcançável. Segundo Schilling (1996), o problema da existência de múltiplas soluções pode usar diferentes critérios na escolha da solução, e reitera que em geral são usadas soluções mais próximas da anterior ou prioriza-se o elo de menor tamanho.

Craig (2009) afirma que este problema cinemático pode ser resolvido por solução analíticas ou numéricas. As analíticas, segundo o autor, podem ser algébricas ou geométricas e são aplicáveis em problemas mais simples. Já os métodos numéricos, ainda em conformidade à Craig (2009), são computacionalmente custosos, porém encontram resultados com aproximação precisa com mais facilidade.

Os métodos numéricos, segundo Aristidou e Lasenby (2009), podem ser divididos pela forma de abordagem, sendo os principais o uso da matriz jacobiana, de sistemas de controle ou funções de otimização. Fêdor (2003) afirma que o uso da matriz jacobiana em inversa ou pseudoinversa consegue resolver o problema da cinemática inversa, porém sofre com as singularidades.

O método cíclico de coordenadas descendentes (CCD) proposto por Wang e Chen (1991) é considerado por Almeida (2016) como um dos métodos de convergência mais rápida para a proximidade do desejado, porém lento para altas precisões e gera movimentos bruscos.

3.2 Planejamento de Caminho

Um caminho é definido por Siciliano et al. (2010) como um conjunto de pontos (posição e orientação) que o manipulador deve seguir na execução do movimento entre um

ponto inicial e outro onde se deseja chegar. É uma descrição puramente geométrica do movimento.

Segundo Craig (2009), os pontos entre o ponto inicial e o final são denominados pontos de passagem. Geralmente, estes pontos de passagem são determinados pelo usuário e o manipulador passa o mais perto possível dos pontos escolhidos, como afirma Vukobratovic (2012).

De acordo com Craig (2009), os pontos de passagem relacionam posição e orientação do efetuador no ponto, relacionado à base. Siciliano et al. (2010) complementa que eles podem ser analisados no espaço de junta ou no espaço operacional, porém devem pertencer ao espaço de trabalho do manipulador.

Siciliano et al. (2010) e Craig (2009) afirmam que quando o caminho é definido a partir do espaço de juntas, o formato dele se torna complexo no espaço cartesiano, apesar de possuir movimento suave das juntas. Ao defini-lo no espaço operacional, ele possui forma suave no espaço cartesiano, porém pode haver movimentos bruscos por parte das juntas, lugares não alcançáveis, além de enfrentar problemas de singularidades.

3.3 Métodos Heurísticos

Métodos heurísticos são algoritmos exploratórios que buscam resolver determinado problema, como sugere Colin (2007). Ainda segundo o autor, a solução final é a solução subótima mais próxima possível à ótima sob as condições estabelecidas.

Aqui serão tratados quatro algoritmos heurísticos distintos: algoritmo genético (AG), Non-dominated Sorting Genetic Algorithm II (NSGA-II), D* Lite e o método cíclico de coordenadas descendentes (CCD). Estes serão descritos nas subseções desta seção.

3.3.1 Algoritmos Genéticos

O algoritmo genético segundo Goldberg (1989) é um método baseado no processo darwiniano de evolução dos seres vivos. Ele possui definições para indivíduos, cromossomos, genes, mutação e cruzamento (interação entre dois indivíduos a fim de formar indivíduos para a nova geração).

O Algoritmo 2 mostra que inicialmente a população é gerada aleatoriamente. Posteriormente, a cada geração são escolhidos os indivíduos mais aptos entre os quais ocorre o cruzamento dos cromossomos e finalmente alguns indivíduos são mutados, gerando assim os indivíduos da nova população (WHITLEY, 1994).

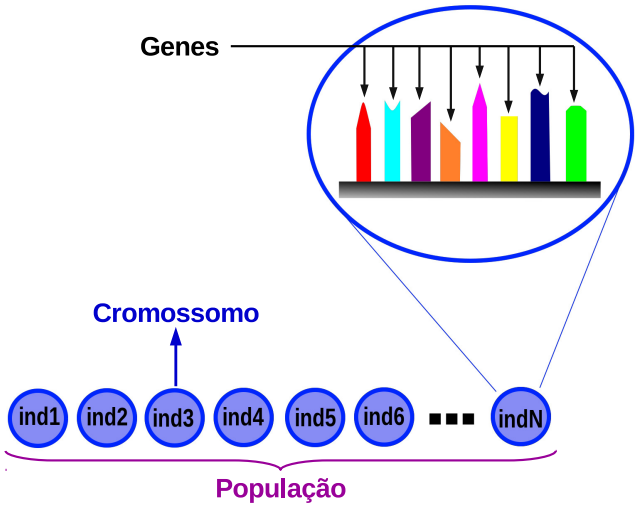
Algorithm 2 Pseudocódigo do Algoritmo Genético

```
1: procedure Algoritmo Genético
2:     t ← 0
3:     Gerar População Inicial P(t)
4:     Avaliar P(t)
5:     while not CriterioDeParada do
6:         for i ← 1 to N_pop/2 do
7:             Selecionar dois indivíduos de P(t)
8:             Aplicar cruzamento aos dois indivíduos com probabilidade p_c
9:             Mutar os novos indivíduos gerados com probabilidade p_m
10:            Inserir os novos indivíduos em P(t+1)
11:        end for
12:        t ← t + 1
13:    end while
14: end procedure
```

Alguns conceitos são necessários para compreender com eficácia o algoritmo genético. Estes são listados a seguir, de acordo com Miranda (2007) e alguns deles são ilustrados na Figura 7.

- **Cromossomo:** Conjunto de valores que representam uma solução candidata. É a representação do indivíduo.
- **Gene:** Um parâmetro contido no cromossomo
- **População:** Conjunto de indivíduos (cromossomos)
- **Geração:** Cada repetição do procedimento do AG
- **Aptidão:** Valor da função objetivo que avalia cada indivíduo da população

Figura 7 – Representação dos conceitos de gene, cromossomo e população.



Fonte: Autoria própria

A função objetivo avalia o grau de adequação do indivíduo ao problema a ser solucionado, penalizando as que não forem admissíveis, conforme Linden (2008). Esta função, segundo Lacerda e Carvalho (1999), será avaliada para todos os indivíduos de todas as populações até que o critério de parada seja satisfeito.

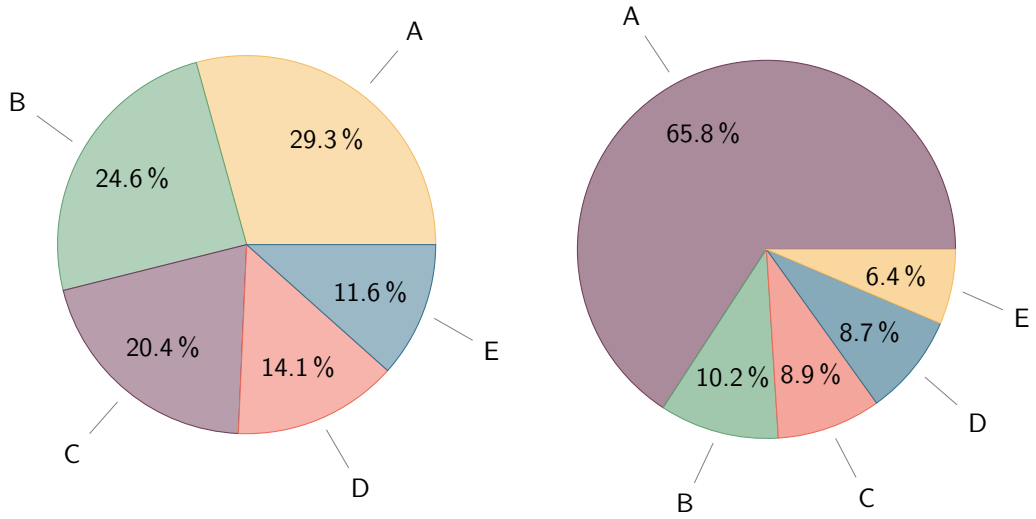
Pacheco et al. (1999) define alguns critérios de parada, sendo um deles o número máximo de gerações, o qual se for atingido finaliza o algoritmo. Outro possível critério de parada sugerido por Miranda (2007), além de um valor de erro mínimo, é a proximidade com o valor de aptidão desejado.

Antes de iniciar o procedimento do algoritmo genético, de acordo com Pacheco et al. (1999) é necessário definir três parâmetros: o tamanho da população (N_{pop}) — que é a quantidade de indivíduos gerados a cada geração —, a taxa de cruzamento (p_c) que é a probabilidade de um indivíduo ser recombinado, e a taxa de mutação (p_m) que é a probabilidade de um gene ser alterado. Ao definir estes parâmetros pode-se inicializar a população gerando aleatoriamente N_{pop} indivíduos.

Posteriormente é iniciada a fase de seleção, a qual segundo Pacheco et al. (1999) se assemelha à seleção natural dos seres vivos onde os indivíduos mais aptos tem maior chance de se reproduzir, porém os menos aptos também podem gerar descendentes. Um dos métodos utilizados, conforme Miranda (2007), é o método da roleta.

Neste método, considera-se um círculo dividido em N_{pop} partes, como ilustra a Figura 8 do lado esquerdo. Ainda de acordo com Miranda (2007), cada pedaço da roleta tem o tamanho de acordo com a aptidão do indivíduo.

Figura 8 – Representação do método de seleção da roleta.



Fonte: Autoria própria

Lacerda e Carvalho (1999) afirma que neste método, gera-se um número aleatório entre zero e o máximo valor das aptidões acumuladas. O indivíduo que estiver mais próximo acima do valor gerado é selecionado. Assim, o indivíduo mais apto tem mais chance de ser selecionado que os menos aptos.

Porém, Linden (2008) alerta que este método pode levar à perda de diversidade e convergência prematura no caso ilustrado à direita na Figura 8. Este efeito é chamado de convergência genética e pode ser minimizado selecionando também indivíduos menos aptos.

Um método que pode ser utilizado para garantir a diversidade dos indivíduos selecionados, segundo Pacheco et al. (1999), é a seleção por torneio. Nele, são escolhidos randomicamente na população k indivíduos para formar uma subpopulação. Dentro desta subpopulação, o indivíduo mais apto é selecionado.

De posse dos indivíduos selecionados, inicia-se a etapa de cruzamento (*crossover*, em inglês). Segundo Fogel e Atmar (1990) a princípio, o método de recombinação não modificava os genes, ocorrendo apenas uma troca entre eles. Um dos métodos pioneiros a implementar o processo de mutação foi o chamado método do cruzamento médio de Davis, em que Jr e Davis (1991) propuseram calcular a média para modificar alguns genes.

Posteriormente, Radcliffe (1992) propôs um método no qual a população teria valores uniformes em relação aos pais. Neste método, um valor β randômico entre zero e um é usado para calcular os novos indivíduos conforme as Equações 3.8 e 3.9. Nelas, p_1 e p_2 são os indivíduos selecionados para gerar os indivíduos c_1 e c_2 para a nova geração.

$$c_1 = \beta p_1 + (1 - \beta)p_2 \tag{3.8}$$

$$c_2 = \beta p_2 + (1 - \beta)p_1 \tag{3.9}$$

Outro método de cruzamento popular foi proposto por Wright (1991) e contorna casos onde o cruzamento resulta em uma população com baixa aptidão. Os indivíduos c_1, c_2 e c_3 são os candidatos para a nova população e são calculados usando as Equações 3.10, 3.11 e 3.12, nas quais p_1 e p_2 são os indivíduos selecionados da população. Os indivíduos da nova população são os dois mais aptos entre os três gerados.

$$c_1 = \frac{1}{2}p_1 + \frac{1}{2}p_2 \tag{3.10}$$

$$c_2 = \frac{3}{2}p_1 - \frac{1}{2}p_2 \tag{3.11}$$

$$c_3 = \frac{1}{2}p_1 + \frac{3}{2}p_2 \tag{3.12}$$

Finalizado o processo de geração dos novos indivíduos, inicia-se o processo de mutação, o qual garante a diversidade da população de acordo com Lacerda e Carvalho (1999). Na mutação, conforme Miranda (2007), um número aleatório é gerado entre zero e um e comparado à taxa de mutação. Se este número for menor que a taxa de mutação predeterminada, que segundo Linden (2008) deve ser extremamente baixa, então o valor do gene é alterado.

Para garantir que o algoritmo genético sempre evolua de uma geração para outra sem perder o melhor cromossomo, Jong e Brandon (1975) propôs a estratégia chamada elitismo. Esta consiste, segundo Lacerda e Carvalho (1999), em transferir pelo menos um dos melhores indivíduos da população atual para a próxima população. Assim, como afirma Linden (2008), no pior dos casos o melhor indivíduo de uma população será igual ao melhor indivíduo da anterior.

3.3.2 NSGA-II

Em alguns problemas de otimização, de acordo com Souza, Dias e Oliveira (2019), deseja-se encontrar soluções que consigam sanar mais de um objetivo. Estes são denominados por Li e Zhang (2008) problemas multiobjetivos, e encontram um conjunto de melhores soluções ao invés de uma única solução. O conjunto de soluções ótimas é chamado, segundo Emmerich e Deutz (2018), curva de Pareto.

Assim, Filho, Ferreira e Vergilio (2018) reiteram que algoritmos multiobjetivos buscam encontrar um vetor de soluções que minimizem ou maximizem N_{obj} funções,

respeitando certas restrições. O Non-dominated Sorting Genetic Algorithm II (NSGA-II) é um dos algoritmos multiobjetivos mais populares na literatura, de acordo com Vargas (2018).

Deb et al. (2002) define o algorito NSGA-II como um algoritmo genético voltado para problemas com mais de um objetivo. Conforme pode ser analisado no Algoritmo 3 de Pimenta et al. (2014), ele difere do AG apenas no método de seleção dos indivíduos.

Algorithm 3 NSGA-II

procedure NSGAII
    $P \leftarrow$ População pai
    $Q \leftarrow$ População filha
    $T \leftarrow$ Tamanho fixo para P e Q
    $F_i \leftarrow$ Conjunto de soluções na fronteira j
    $n \leftarrow$ Número da geração atual
    $N \leftarrow$ Número máximo de gerações
    Gerar a população inicial P_0 aleatoriamente
    $Q_0 \leftarrow \{\}$
    $n \leftarrow 0$
    while n < N **do**
        Realizar seleção, cruzamento e mutação em P_n para gerar a população filha Q_n
        $R_n \leftarrow P_n \cup Q_n$
        Ordenar por não dominância R_n, gerando F_i, $i = \{1, ..., v\}$ em R_n
        $P_{n+1} \leftarrow \{\}$
        while $|P_{n+1}| + |F_i| \leq N$ **do**
            Copiar as soluções de F_i em P_{n+1}
            $i \leftarrow i + 1$
        end while
        Calcular a distância de multidão (d_{mult}) para cada solução em F_i
        Ordenar F_i decrescentemente de acordo com d_{mult} de cada solução
        Copiar as primeiras $N - |P_{n+1}|$ soluções ordenadas de F_i para P_{n+1}
        $n \leftarrow n + 1$
    end while
end procedure

Segundo Deb (2001), o critério de dominância de Pareto é geralmente o mais adotado para classificação de soluções em contexto multiobjetivo. De acordo com este critério, entre duas soluções x_1 e x_2, pode-se dizer que x_1 domina x_2 ($f(x_1) \prec f(x_2)$) se a Equação 3.13 é verdadeira como assegura Batista et al. (2011), onde os operadores

relacionais $\leq$ e $\neq$ são definidos pelas relações das Equações 3.14 e 3.15.

$$f(x_1) \leq f(x_2) \;\; \& \;\; f(x_1) \neq f(x_2) \tag{3.13}$$

$$f(x_1) \leq f(x_2) \Leftrightarrow f_i(x_1) \leq f_i(x_2) \quad \forall i \in \{0, 1, \cdots, m\} \tag{3.14}$$

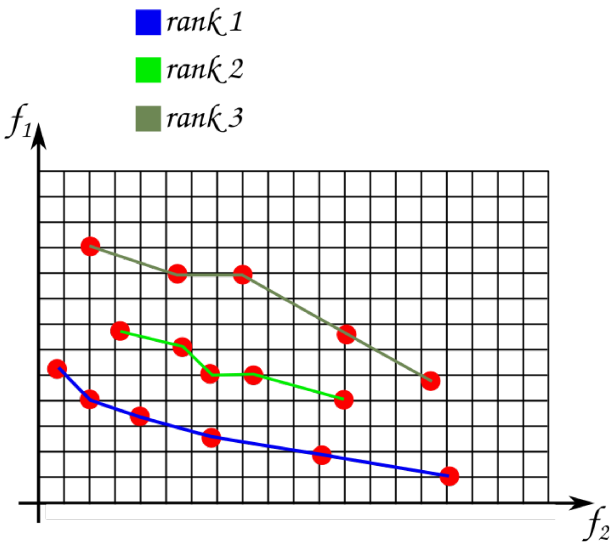
$$f(x_1) \neq f(x_2) \Leftrightarrow \exists i \in \{0, 1, \cdots, m\} : f_i(x_1) \neq f_i(x_2) \tag{3.15}$$

As soluções que não possuem outra solução que as domine são denominadas soluções não-dominadas. A curva de Pareto P é então um conjunto de soluções não dominadas descrita matematicamente pela Equação 3.16, como endossa Voorneveld (2003).

$$P = \{x^* \in \Omega \mid \nexists x \in \Omega : f(x) \prec f(x')\} \tag{3.16}$$

A primeira etapa do processo de seleção é a classificação dos indivíduos em fronteiras de dominância (*fronts* ou *ranks*, em inglês) por ordem descendente de não dominação, conforme Vargas (2018). A classificação é representada pelo gráfico da Figura 9, onde $\mathcal{F}_i$ é a fronteira que corresponde à curva de Pareto local.

Figura 9 – Representação gráfica da classificação por fronteiras de dominância.



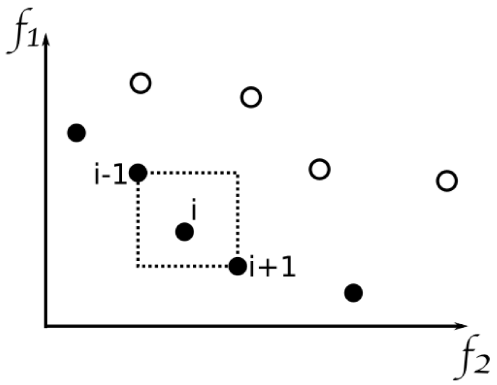
Fonte: Autoria própria

Com todos os indivíduos classificados, de acordo com Bekele e Nicklow (2007), inicia-se o método de seleção por torneio, no qual k indivíduos são randomicamente selecionados para formar uma subpopulação temporária. Kannan et al. (2008) complementa que o indivíduo selecionado desta subpopulação é o de menor valor de fronteira.

Porém, há casos em que existe mais de um indivíduo com o valor mais baixo de fronteira. Neste cenário, conforme Asgari et al. (2013) é necessário utilizar o método denominado distância da multidão para selecionar um indivíduo apenas entre eles. Este método, segundo Deb et al. (2002) garante a diversidade da população através da densidade estimada do indivíduo.

A Figura 10 representa a distância da multidão calculada usando a Equação 3.17 conforme o Algoritmo 4, de Fraccaroli (2010). O método consiste em calcular o valor do cuboide traçado entre os indivíduos vizinhos ao indivíduo i (i-1 e i+1), de acordo com Fortin e Parizeau (2013).

Figura 10 – Representação gráfica do método de distância da multidão.



Fonte: Autoria própria

$$\mathcal{F}_{dist}[i] = \mathcal{F}_{dist}[i] + \frac{\mathcal{F}[i+1].m - \mathcal{F}[i-1].m}{f_m^{max} - f_m^{min}} \tag{3.17}$$

Algorithm 4 Método de distância da multidão

```
1:  procedure DistanciaMultidao(F_i)
2:      l ← Número de soluções em F_i
3:      for each solução em F_i do
4:          d_i = 0
5:      end for
6:      for each objetivo m do
7:          Ordenar de forma decrescente as soluções por f_m na lista I^m
8:      end for
9:      for m = 0; m = M; m++ do
10:         for each solução extrema (mínimo e máximo) do
11:             Fazer d_{I_1^m} = d_{I_l^m} = inf
12:         end for
13:     end for
```

14: **for** $i = 1; i = l - 1; i++$ **do**
15: $d_{I_i^m} = d_{I_i^m} + (f_m^{I_{i+1}^m} - f_m^{I_{i-1}^m})/(f_m^{max} - f_m^{\min})$
16: **end for**
17: **end procedure**

Na Equação (3.17), $\mathcal{F}_{dist}[i]$ é o valor da distância de multidão para o indivíduo i, $\mathcal{F}[i-1].m$ é o valor do m^{th} objetivo do i^{th} indivíduo na fronteira de dominância $\mathcal{F}$, f_m^{max} é o máximo valor para o objetivo m e f_m^{min} é o mínimo valor para o objetivo m. Assim, o indivíduo com o maior valor desta distância será o indivíduo selecionado para a etapa de cruzamento, como afirma Jensen (2003).

O restante do algoritmo é semelhante aos procedimentos utilizados no algoritmo genético. O que difere é que ao invés de uma única solução, será encontrada uma curva de soluções ótimas, de acordo com Deb (2001).

3.3.3 D* Lite

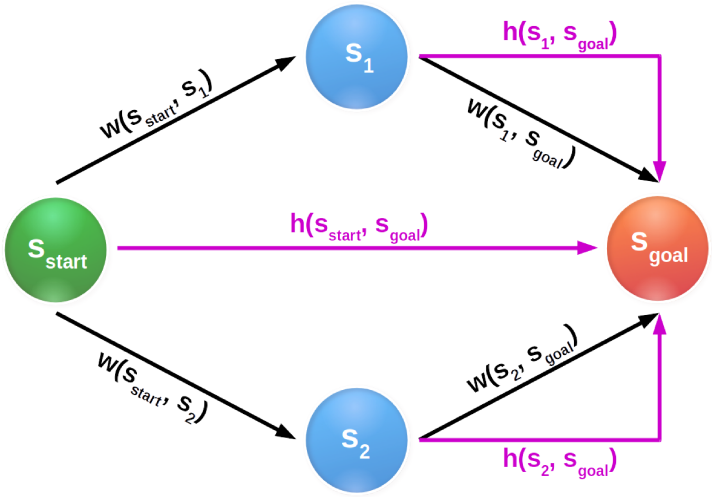
Este algoritmo foi inicialmente proposto por Koenig e Likhachev (2002) e é um método heurístico de busca incremental baseado em algoritmos derivados de Dijkstra. Ele utiliza mapeamento em grids e é adaptável à mudanças ocorridas durante o trajeto.

Nas buscas incrementais, como abordado por Lu et al. (2011), são realizadas buscas no mapa através da repetição do planejamento do caminho, seguido pelo reconhecimento das mudanças ocorridas e atualização do resultado na busca com base nas alterações. A repetição destes três procedimentos ocorre até que se encontre o objetivo desejado.

Para um melhor entendimento, é necessário compreender alguns conceitos fundamentais da busca por grafo, como é o caso do D* Lite. Um grafo é definido por Imai e Asano (1986) como um conjunto de objetos relacionados por arcos direcionados.

Na busca por grafo, representada na Figura 11, existem os nós (*nodes*, em inglês) representados pelos círculos os quais são conectados por um arco de peso ω calculado por uma função peso, como descrito por Mani e Bloedorn (1997). Esta função calcula o custo para ir de um nó s a outro s' ($\omega(s, s')$).

Figura 11 – Representação de um grafo.



Fonte: Autoria própria

Existe ainda a função heurística abordada por Riesen, Fankhauser e Bunke (2007), que estima o custo para ir do nó atual até o nó desejado (s_{goal}). Devem ser definidos o nó inicial (s_{start}) e o nó onde se deseja chegar. Ainda de acordo com o autor, a saída do problema deve ser o caminho de menor custo.

Outro conceito fundamental é o relaxamento das arestas, que segundo Goldberg (2001) consiste em um processo para verificar se é possível melhorar o caminho mais curto passando por outro nó. O Algoritmo 5 de Biswas, Alam e Doja (2013) é referente ao relaxamento de uma aresta (u, v), onde $v.\pi$ é o nó antecessor de v.

Algorithm 5 Relaxamento das arestas

```
1: procedure RELAXAMENTO(u, v, ω)
2:     if d[v] > d[u] + ω(u, v) then
3:         d[v] ← d[u] + ω(u, v)
4:         v.π ← u
5:     end if
6: end procedure
```

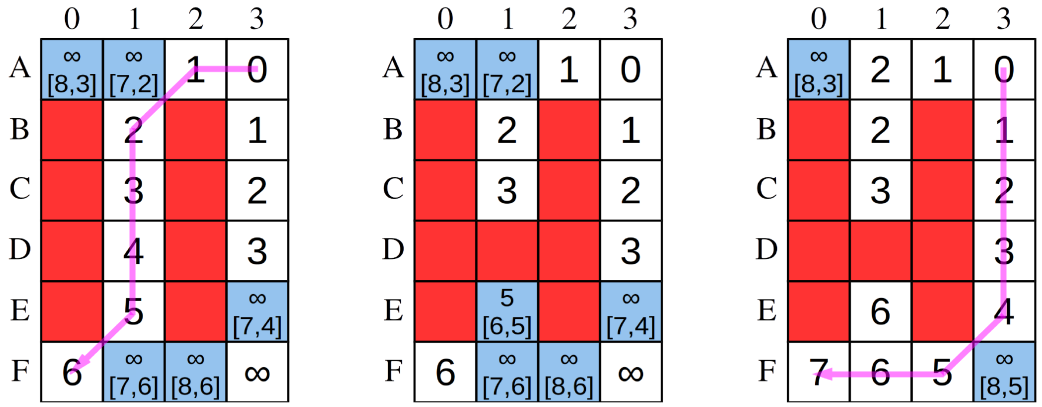
A partir da compreensão dos métodos e parâmetros explicitados anteriormente, será abordado como utilizar os resultados prévios de uma busca em uma nova busca. Para isto, é necessário salvar os melhores resultados de uma busca prévia. Neste algoritmo, são salvos o menor caminho e os valores para $g(s)$ que é o custo atual do caminho para o nó s, como esclarece Koenig e Likhachev (2002).

Para calcular o menor caminho, é necessário encontrar as inconsistências locais que ocorrerem no grafo. Então os dados salvos anteriormente e as inconsistências serão usadas

conforme Marques (2012) para relaxar as arestas na busca de uma nova solução ideal.

Este procedimento é descrito por Koenig e Likhachev (2002) e ilustrado pela Figura 12, onde à esquerda é representada uma busca inicial e o melhor caminho encontrado por ela. Na representação localizada ao centro da Figura 12, há um obstáculo em um local que anteriormente era acessível. Isto irá resultar na atualização dos custos das arestas.

Figura 12 – Representação da busca dinâmica.



Fonte: Adaptado de Koenig, Likhachev e Furcy (2004).

Os valores dentro dos espaços do grid representam os valores de $g(s)$ e serão atualizados no caminho representado à direita na Figura 12, nos locais próximos ao obstáculo inserido. Após a atualização é possível encontrar o novo caminho ótimo.

O algoritmo D* Lite, de acordo com Ferguson, Likhachev e Stentz (2005), combina a otimalidade e eficiência dos métodos heurísticos e a capacidade de replanejamento com rapidez dos métodos de busca incrementais. Por isto, ele é largamente utilizado por robôs exploradores e em desenvolvimento de jogos (Stentz e Hebert (1995), Hebert, MacLachlan e Chang (1999), Matthies et al. (2002), Thayer et al. (2001), Zlot et al. (2002), Likhachev (2003)).

O Algoritmo 6 é um pseudocódigo guia referente ao D* Lite. O vetor de pesos de prioridade para determinado nó é calculado, o qual é importante quando existe mais de um nó com mesmo valor de prioridade. Quando se usa um vetor de prioridades, caso a primeira delas seja igual para mais de um nó, a segunda será a decisiva.

Algorithm 6 Pseudocódigo do D* Lite

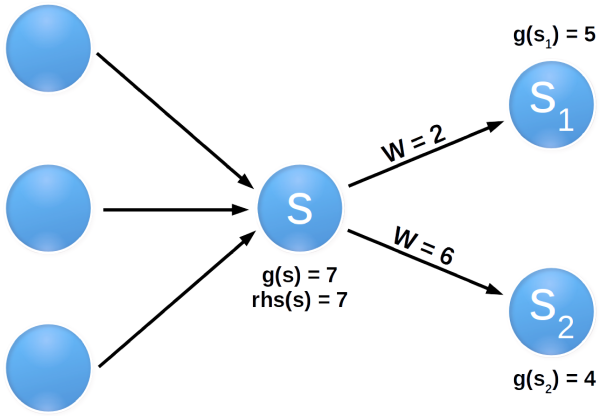
```
1: Inicializa todos os nodes como não expandidos
2: while not CriterioDeParada do
3:     Busca melhor-primeiro até que s_start seja consistente com os vizinhos e expandido
4:     Move até o próximo melhor nó
5:     if mudou custo de alguma aresta then
6:         Acompanhar como as heurísticas mudaram
```

7: Atualizar os nós de origem das arestas alteradas
8: **end if**
9: **end while**

Koenig e Likhachev (2002) define o valor $rhs(s)$ que é o custo do nó s levando em consideração um passo a frente, e é utilizado para eficiência computacional. Quando $g(s) \neq rhs(s)$, diz-se que há inconsistência local. O autor ainda afirma que existem dois tipos de inconsistência: a superconsistência (*overconsistency*, em inglês) e a subconsistência (*underconsistency*).

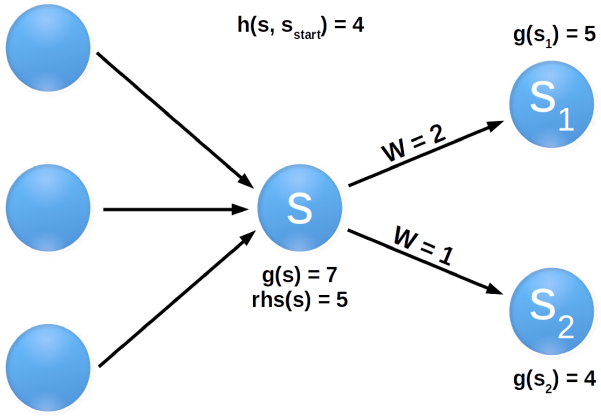
Oral e Polat (2012) afirma que há uma superconsistência quando $g(s) > rhs(s)$ e subconsistência quando $g(s) < rhs(s)$. As Figuras 13, 14 e 15 representam o processo de tratamento de uma superconsistência. Inicialmente, os pesos de cada aresta são mostrados na Figura 13, sendo que o menor custo é $g(s) = 7$. Os custos neste caso estão consistentes.

Figura 13 – Representação dos nós consistentes.



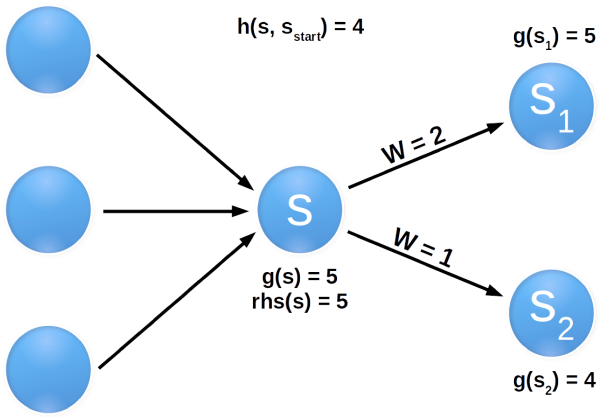
Fonte: Autoria própria

Figura 14 – Representação de variação de peso da aresta gerando superconsistência.



Fonte: Autoria própria

Figura 15 – Representação do tratamento da superconsistência.



Fonte: Autoria própria

Então o valor do peso de uma das arestas, como representado na Figura 14 e o valor $rhs(s)$ são atualizados. O nó agora é superconsistente, já que $g(s) > rhs(s)$. O nó é então inserido na lista prioritária U.

Os valores de prioridade são, conforme Koenig e Likhachev (2002) advindos da Equação 3.18, onde o primeiro valor é o prioritário e o segundo é comparado apenas quando houver mais de um nó com o primeiro valor idêntico. São inseridos na lista prioritária U apenas os nós inconsistentes, e após lidar com a inconsistência eles são removidos dela.

$$[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))] \tag{3.18}$$

Marques (2012) afirma que em casos de superconsistência o novo caminho é melhor que o anterior, portanto basta igualar $g(s) = rhs(s)$, conforme a Figura 15. Oral (2012) complementa que ao igualar, o nó s é removido da lista prioritária U, já que se tornou novamente consistente, e seus nós antecessores são atualizados.

O caso de subconsistência é mais complexo de acordo com Marques (2012), pois o custo do caminho antigo era melhor que o novo. O valor de $g(s)$ é então definido como ∞, conforme Mengana (2013), e o nó s e seus antecessores são atualizados. Ainda em conformidade com a publicação do autor, a atualização dos nós irá ocorrer até que todos os nós sejam consistentes novamente.

Quando o nó s_{start} é atualizado, todas as heurísticas mudam, como alerta Oral e Polat (2012). Para ser mais eficiente computacionalmente, Koenig e Likhachev (2002) usa a variável modificadora k_m (*key modifier*, em inglês). Esta variável é definida pela Equação 3.19 e atualizada sempre que o s_{start} é atualizado.

$$k_m = k_m + h(s_{last}, s_{start}) \tag{3.19}$$

Utilizando esta variável não é necessário recalcular a heurística para todos os nós, já que segundo Mengana (2013), k_m está inserida no cálculo da prioridade. Conhecendo todos os métodos e variáveis que compõem o algoritmo D* Lite, pode-se compreender o Algoritmo 7 completo publicado por Koenig e Likhachev (2002) em seu artigo.

Algorithm 7 D* Lite

```
1: function CalculateKey(s)
2:     return [min(g(s), rhs(s)) + h(s_start, s) + k_m; min(g(s), rhs(s))]
3: end function
4: procedure Initialize( )
5:     U ← [ ]
6:     k_m ← 0
7:     for all s ∈ S do
8:         rhs(s) ← g(s) ← ∞
9:     end for
10:    rhs(s_goal) ← 0
11:    U.Insert(s_goal, [h(s_start, s_goal), 0])
12: end procedure
13: procedure UpdateVertex(u)
14:    if g(u) ≠ rhs(u) & u ∈ U then
15:        U.Update(u, CalculateKey(u))
16:    else if g(u) ≠ rhs(u) & u ∉ U then
17:        U.Insert(u, CalculateKey(u))
18:    else if g(u) = rhs(u) & u ∈ U then
19:        U.Remove(u)
20:    end if
21: end procedure
22: procedure ComputeShortestPath( )
23:    while U.TopKey() < CalculateKey(s_start) || rhs(s_start) > g(s_start) do
24:        u ← U.Top()
25:        k_old ← U.TopKey()
26:        k_new ← CalculateKey(u)
27:        if k_old < k_new then
28:            U.Update(u, k_new)
29:        else if g(u) > rhs(u) then
30:            g(u) ← rhs(u)
31:            U.Remove(u)
32:            for all s ∈ Pred(u) do
33:                if s ≠ s_goal then
34:                    rhs(s) ← min (rhs(s), c(s, u) + g(u))
```

```
35:                 end if
36:                 UpdateVertex(s)
37:             end for
38:         else
39:             g_old ← g(u)
40:             g(u) ← ∞
41:             for all s ∈ Pred(u) ∪ {u} do
42:                 if rhs(s) = c(s, u) + g_old then
43:                     if s ≠ s_goal then
44:                         rhs(s) ← min_{s'∈Succ(s)}(c(s, s') + g(s'))
45:                     end if
46:                 end if
47:                 UpdateVertex(s)
48:             end for
49:         end if
50:     end while
51: end procedure
52: procedure Main( )
53:     s_last ← s_start
54:     Initialize( )
55:     ComputeShortestPath( )
56:     while s_start ≠ s_goal do
57:         if rhs(s_start) = ∞ then
58:             Não existe caminho
59:         end if
60:         s_start ← arg min_{s'∈Succ(s_start)}(c(s_start, s') + g(s'))
61:         Move para s_start
62:         Busca por arestas que tenham custos alterados
63:         if alguma aresta mudou de custo then
64:             k_m ← k_m + h(s_last, s_start)
65:             s_last ← s_start
66:             for all arestas direcionadas (u, v) que tenham mudado os custos do
67:                 c_old ← c(u, v)
68:                 Atualiza o custo da aresta c(u, v)
69:                 if c_old > c(u, v) then
70:                     if u ≠ s_goal then
71:                         rhs(u) ← min(rhs(u), c(u, v) + g(v))
72:                     end if
73:                 else if rhs(u) = c_old + g(v) then
```

```
74:                 if u ≠ s_goal then
75:                     rhs(u) ← min_{s'∈Succ(u)}(c(u, s') + g(s'))
76:                 end if
77:             end if
78:             UpdateVertex(u)
79:         end for
80:         ComputeShortestPath( )
81:     end if
82: end while
83: end procedure
```

3.3.4 CCD

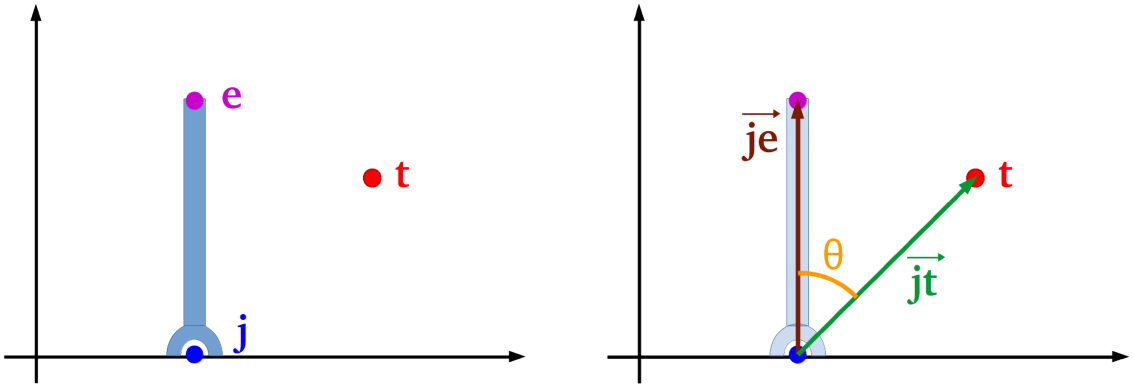
O método cíclico de coordenadas descendentes (CCD) é um método de busca direcionada e foi proposto inicialmente por Wang e Chen (1991) e posteriormente Chris (1993) o usou para descrever um método que resolve o problema da cinemática inversa. Quando se deseja calcular a cinemática inversa, conforme Lander (1998b) principalmente em manipuladores de alta complexidade, surgem alguns empecilhos, já abordados na subseção 3.1.2.

Este algoritmo, de acordo com Kenwright (2012), busca solucionar estas complicações otimizando cada junta da ponta à base ou da base à ponta para chegar o mais próximo possível do alvo. As juntas são alternadamente otimizadas, como afirma Zeng et al. (2014), repetindo até que a solução seja encontrada ou que se extrapole o número máximo de iterações.

O objetivo é minimizar a distância entre o alvo e o efetuador através da otimização dos ângulos, como defendem Boomsma e Hamelryck (2005) em seu artigo. Dependendo do comportamento da junta, diferentes equações devem ser usadas para encontrar a solução ideal, segundo Olsen e Petersen (2011).

A Figura 16 mostra à esquerda uma junta rotacional posicionada em j, a posição atual e do efetuador, e a posição t do alvo desejado. Para encontrar o valor θ que a junta deve rotacionar, calcula-se o vetor $\vec{je}$ que vai da junta ao efetuador e o vetor $\vec{jt}$ traçado entre a junta e o alvo, como ilustrado do lado direito da Figura 16.

Figura 16 – Representação do CCD para juntas de rotação.



Fonte: Autoria própria

Assim, θ é o valor que a junta precisa rotacionar para que $\vec{je}$ se assemelhe o máximo possível a $\vec{jt}$. Lander (1998a) afirma que calculando o arco cosseno do produto escalar como na Equação 3.20, obtem-se o valor do ângulo entre os vetores porém sem garantir sua direção.

$$\theta' = \arccos\left(\frac{e-j}{||e-j||} \cdot \frac{t-j}{||t-j||}\right) \tag{3.20}$$

Quando se utiliza a operação do produto vetorial, um vetor perpendicular aos dois vetores é criado, segundo Martin, Barrientos e Cerro (2018). A Equação 3.21 mostra o cálculo do produto vetorial no espaço. O sinal proveniente da somatória entre os elementos deste cálculo define a orientação do ângulo θ. Assim, para cada junta, calcula-se as Equações 3.20 e 3.21.

$$\vec{dir} = \vec{je} - \vec{jt} = \frac{e-j}{||e-j||} \times \frac{t-j}{||t-j||} \tag{3.21}$$

Portanto, de acordo com Lander (1998a), o ângulo θ que precisa ser deslocado pela junta é dado pela relação da Equação 3.22, na qual o ângulo θ' é ajustado de acordo com o vetor de direção $\vec{dir}$. Este procedimento é realizado, conforme Martin, Barrientos e Cerro (2018) para cada junta partindo do efetuador até chegar à base, ou da base até chegar ao efetuador dependendo do tipo de algoritmo utilizado, e repete até encontrar o alvo desejado.

$$\theta = \begin{cases} \theta' & se\ dir > 0 \\ -\theta' & se\ dir < 0 \end{cases} \tag{3.22}$$

No caso da junta prismática, Wang e Chen (1991) definem a Equação 3.23 do erro de posição $\Delta p(\lambda)$. Nela, λ é o deslocamento da junta prismática e $\delta P = P_{id} - P_{ih}$, onde

P_{id} é o tamanho do vetor que vai da junta ao alvo e P_{ih} é o tamanho do vetor da junta ao efetuador. z_i é um vetor unitário na direção do eixo z da junta i.

$$\Delta p(\lambda) = (P_{id} - (P_{ih} + z_i\lambda)) \cdot (P_{id} - (P_{ih} + z_i\lambda)) = \delta P^2 - 2(\delta P \cdot z_i)\lambda + \lambda^2 \tag{3.23}$$

Derivando a Equação 3.23 em relação a λ, encontra-se a Equação 3.24. Sabe-se por Wang e Chen (1991) que para minimizar, o valor da derivada deve ser nulo. Portanto, o erro de posição é mínimo quando a relação da Equação 3.25 é satisfeita.

$$\frac{\partial \Delta p(\lambda)}{\partial \lambda} = 2(\lambda - \delta P \cdot z_i) \tag{3.24}$$

$$\lambda = \delta P \cdot z_i \tag{3.25}$$

Estes equacionamentos serão realizados para cada junta até que o critério de parada seja atingido. O Algoritmo 8, escrito por Kenwright (2012), segue a proposta do CCD quando inicia do efetuador e é percorrido até a base, denominado *CCD backward*.

Algorithm 8 CCD do efetuador à base

```
1:  procedure CCD_Backward(error, k_max, n)
2:      k ← 0                                              ▷ Contador de iterações
3:      while k < k_max do
4:          for i = n − 1; i < 0; i − − do                 ▷ Contador de juntas
5:              Calcular je, jt
6:              Calcular valor do ângulo      ▷ Eq. 3.20 e 3.21 se α ou θ e Eq. 3.25 se d
7:              Deslocar a junta conforme o ângulo calculado
8:              Calcular a nova posição
9:              if |e − t| < error then                    ▷ Chegou ao destino
10:                 return Novos ângulos
11:             end if
12:         end for
13:         k ← k + 1
14:     end while
15: end procedure
```

O Algoritmo 9, também escrito por Kenwright (2012), trata do algoritmo do CCD da base ao efetuador, denominado *CCD Forward*. Este distingue do anterior apenas pelo laço de repetição incrementar o index das juntas ao invés de decrementar.

Algorithm 9 CCD da base ao efetuador

```
1: procedure CCD_FORWARD(error, k_max, n)
2:     k ← 0                                              ▷ Contador de iterações
3:     while k < k_max do
4:         for i = 0; i = n; i++ do                       ▷ Contador de juntas
5:             Calcular je, jt
6:             Calcular valor do ângulo   ▷ Eq. 3.20 e 3.21 se α ou θ e Eq. 3.25 se d
7:             Deslocar a junta conforme o ângulo calculado
8:             Calcular a nova posição
9:             if |e − t| < error then                    ▷ Chegou ao destino
10:                return Novos ângulos
11:            end if
12:        end for
13:        k ← k + 1
14:    end while
15: end procedure
```

Tanto o Algoritmo 8 quanto o Algoritmo 9 recebem três variáveis: *error* que é o valor de precisão desejada do erro de posição, k_{max} que é o número máximo de iterações e n que é o index de elos do manipulador. Elas retornam um vetor com a coordenada de junta ideal calculada para atingir o ponto cartesiano desejado.

Segundo Gurbuzbalaban et al. (2017) o algoritmo CCD possui boa performance e rápida convergência. Kenwright (2012) complementa que o tipo de algoritmo melhor entre o *CCD Forward* e o *CCD Backward* depende do problema abordado.

4 | Metodologia

Este capítulo trata das etapas realizadas durante o desenvolvimento do trabalho aqui proposto, as quais são apresentadas em ordem cronológica. O primeiro tópico (4.1) aborda o desenvolvimento da interface gráfica, seguido por dois outros tópicos (4.2 e 4.3) que retratam a utilização de dois algoritmos - AG e NSGA-II - na tentativa de obter um comportamento satisfatório, até chegar no algoritmo que atendeu às expectativas almejadas.

Ao desenvolver o algoritmo adequado à proposta deste trabalho, foi necessário realizar algumas modificações no algoritmo original do D* Lite. Os impasses encontrados e as maneiras como foram elucidados encontram-se na seção 4.4. A combinação deste algoritmo modificado com a inserção do CCD, bem como as alterações realizadas para a criação do algoritmo POM são abordadas na seção 4.5.

4.1 Interface Gráfica

Um dos objetivos do presente trabalho é que o algoritmo consiga encontrar a trajeto subótimo[1] entre dois pontos, para diferentes anatomias de manipulador robótico. Para isto, foi necessário utilizar um software que permitisse a manipulação da interface de forma a se adequar ao desejado. Assim, foi utilizado o software *Unity3D* que é uma *game engine* bastante utilizada na criação de jogos, programado em linguagem C#.

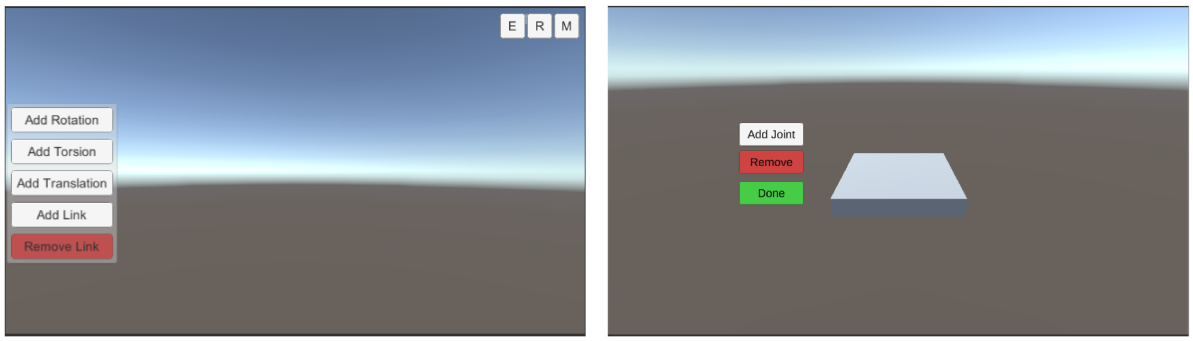
Dois modelos de interface serão apresentados ao longo deste documento, sendo uma utilizada com o AG e o NSGA-II e a outra a partir da utilização do D* Lite. Elas diferem apenas esteticamente, possuindo as mesmas funcionalidades em ambas versões.

A tela ilustrada na Figura 17, sendo à esquerda a primeira versão da interface e à direita a segunda, é onde o usuário pode inserir as juntas e elos por tipo (rotação, torção, deslocamento e elo) e seus limites máximo e mínimo para as juntas e tamanho para o elo. O botão verde com a escrita "*Done*" na nova versão, representada à direita, inicia o

[1] Algoritmos de otimização podem cair em mínimos locais. Por não ser necessariamente o mínimo global obtido pela minimização, será utilizado neste trabalho a nomenclatura "trajeto subótimo" para se referir ao trajeto minimizado.

programa para a anatomia inserida.

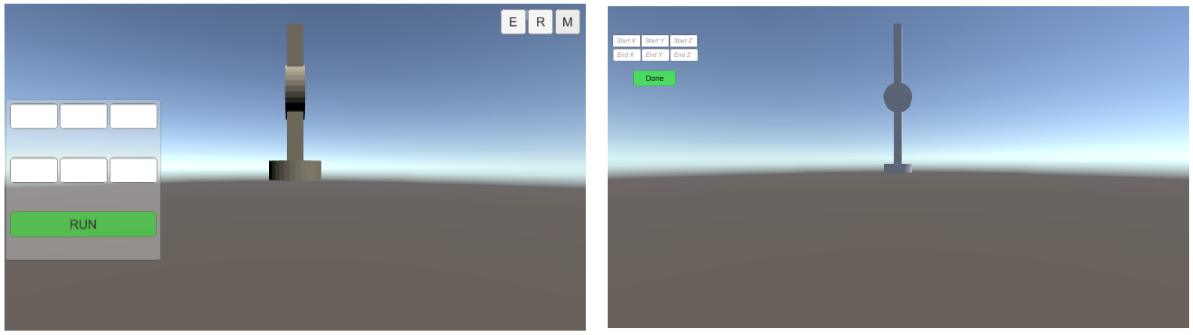
Figura 17 – Tela principal da interface.



Fonte: Autoria própria

Após a montagem do manipulador, basta inserir na tela retratada na Figura 18 os pontos de início (caixas de texto superiores) e de destino (caixas de texto inferiores) do trajeto. À esquerda é exibida a tela de seleção dos pontos na primeira versão e à direita é apresentada a tela da nova versão. Na nova versão, após iniciado o algoritmo, o ponto inicial é exibido na cor verde e o final na cor vermelha.

Figura 18 – Seleção dos pontos de partida e objetivo.



Fonte: Autoria própria

4.2 Minimização de erro de posição com Algoritmos Genéticos

O capítulo 2, o qual trata do estado da arte sobre o tema proposto, referencia diversos autores que utilizam como algoritmo para trajetórias em braços robóticos o Algoritmo Genético. Por este motivo, inicialmente optou-se por utilizá-lo neste trabalho. Para comprovar sua eficácia em casos onde a anatomia do manipulador pode variar, foi implementada uma rotina de minimização de erro de posição em braços robóticos.

O indivíduo gerado pelo algoritmo genético é um vetor com valores de deslocamentos para cada junta inserida no robô. Estes valores são utilizados para calcular a aptidão do indivíduo através da matriz de transformação do braço robótico encontrada pelo método de Denavit-Hartenberg. Quanto menor a distância euclidiana entre o ponto desejado inserido pelo usuário e o ponto proveniente da matriz de transformação de DH, mais apto é o indivíduo analisado.

O problema de otimização abordado nesta seção é dado pela Equação 4.1, sendo que para cada junta inserida pelo usuário são geradas duas equações de restrição. Estas restrições são calculadas conforme as Equações 4.2 e 4.3, sendo lb o limite inferior, ub o limite superior e par o parâmetro que pode ser α, θ ou d, dependendo do tipo de junta.

$$min(pos_{desejada} - pos_{AG}) \tag{4.1}$$

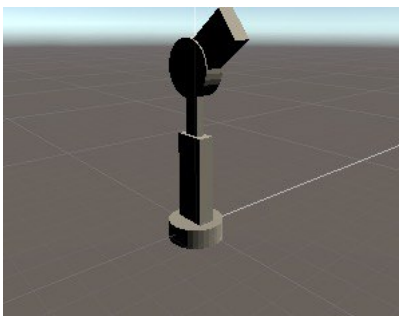
$$g_{min} = lb - par \tag{4.2}$$

$$g_{max} = par - ub \tag{4.3}$$

A definição dos melhores parâmetros de otimização foi realizada a partir da análise de sensibilidade envolvendo o número de gerações, a quantidade de indivíduos em cada população, a probabilidade de cruzamento e a probabilidade de mutação. Foram realizados 972 testes, com as combinações para três valores de cada uma das variáveis a serem analisadas e o valor da semente de 0 a 6 para cada combinação de parâmetros.

Para a análise da sensibilidade foi utilizado o manipulador com a anatomia ilustrada na Figura 19, com a posição desejada ajustada para o ponto $(-4.6\text{ cm}, 56.9\text{ cm}, 13.2\text{ cm})$ e os valores dos elos e juntas listados na Tabela 1.

Figura 19 – Geometria usada para análise de sensibilidade do AG.



Fonte: Autoria própria

Tabela 1 – Tabela de parâmetros utilizados para análise de sensibilidade do AG.

Junta	**Valores setados**
T_0	$0° \leq \alpha \leq 360°$
a_0	30 cm
d	$0 \leq d \leq 25$
R_1	$-120° \leq \theta \leq 120°$
L_1	20 cm

Os dados obtidos na análise de sensibilidade são salvos em um arquivo de extensão csv. Foi implementado um script em linguagem Python que lê este arquivo de dados e realiza a seleção dos casos que obedecem ao requisito de erro na posição. Dentre os dados selecionados, é realizado o cálculo estatístico da moda para cada parâmetro. A Tabela 2 retrata a tabela gerada pelo script com os melhores parâmetros obtidos para o AG destacados.

Tabela 2 – Melhores parâmetros da análise de sensibilidade do AG.

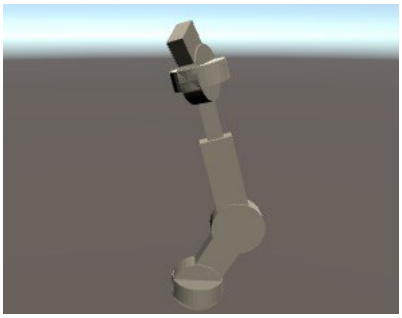
Semente	**Tipo de Cruzamento**	**Gerações**	**Indivíduos**	p_m	p_c
0	radcliffe	15000	15	0.05	0.5
1	radcliffe	5000	15	0.1	0.5
1	radcliffe	15000	15	0.1	0.5
2	radcliffe	10000	15	0.1	0.7
2	wright	10000	15	0.1	0.7
2	radcliffe	15000	15	0.05	0.7
2	radcliffe	15000	15	0.1	0.7
2	radcliffe	15000	15	0.1	0.9
2	wright	15000	15	0.1	0.7
3	radcliffe	5000	15	0.1	0.9
3	wright	5000	15	0.1	0.7
3	radcliffe	10000	15	0.1	0.9
3	wright	10000	15	0.1	0.7
3	radcliffe	15000	15	0.1	0.9
3	wright	15000	15	0.1	0.7
4	wright	5000	15	0.1	0.7
4	radcliffe	10000	15	0.05	0.5
4	radcliffe	10000	15	0.1	0.9
4	wright	10000	15	0.1	0.7
4	radcliffe	15000	15	0.05	0.5
4	radcliffe	15000	15	0.05	0.9
4	radcliffe	15000	15	0.1	0.9

continua para próxima página

continuação da página anterior					
4	wright	15000	15	0.1	0.7
5	radcliffe	15000	15	0.1	0.9
5	wright	15000	15	0.1	0.7
		1500	**15**	**0.1**	**0.7**

De posse dos parâmetros obtidos como frequentemente melhores pela análise de sensibilidade, foi realizado um novo teste com um manipulador possuindo anatomia distinta, ilustrada pela Figura 20. Esta foi escolhida devido à quantidade de graus de liberdade ser significativamente maior que a utilizada para análise de sensibilidade e, portanto, mais complexa.

Figura 20 – Anatomia usada para teste dos parâmetros do AG.



Fonte: Autoria própria

O erro mínimo de posição encontrado utilizando o algoritmo genético foi de 0,05277 cm para o método de Radcliffe (1992), enquanto o método de Wright (1991) encontrou um erro mínimo de 0,03955 cm. Porém, o valor de erro médio é de 0,70001 cm para o método proposto por Radcliffe (1992) e de 0,46766 cm para o método proposto por Wright (1991).

De posse destes dados, pode-se afirmar que o erro obtido utilizando este método não foi satisfatoriamente pequeno, principalmente em robôs com mais graus de liberdade. Por este motivo, como será abordado na seção 4.3, decidiu-se migrar para um algoritmo multiobjetivo em busca de assegurar um mínimo valor de erro de posição, juntamente com uma melhor eficiência.

4.3 Minimização de erro de posição usando NSGA-II

O NSGA-II é um algoritmo popular na resolução de problemas com mais de um objetivo, como abordado na subseção 3.3.2. Na etapa abordada nesta seção, ele possui duas funções de aptidão do indivíduo e busca um mínimo deslocamento das juntas enquanto garante um valor mínimo desejado do erro de posição.

Assim, a primeira função de aptidão, dada pela Equação 4.4 é idêntica à utilizada com o AG, e refere-se à minimização da diferença entre a posição desejada e a posição dada pela matriz de transformação homogênea de Denavit-Hartenberg a partir do deslocamento do indivíduo. A outra função de aptidão, descrita pela Equação 4.5 é a minimização dos ângulos de deslocamento das juntas.

$$f_1 = min(pos_{desejada} - pos_{AG}) \tag{4.4}$$

$$f_2 = min\left(\frac{\sum_{j=1}^{num} \mathcal{J}}{num}\right) \tag{4.5}$$

Na Equação 4.4, $pos_{desejada}$ é a posição que foi inserida pelo usuário como final para o manipulador e pos_{AG} é a posição obtida através da matriz de transformação homogênea para o indivíduo gerado pelo algoritmo. Na Equação 4.5, j é o index do indivíduo atual, num é a quantidade de juntas inseridas e $\mathcal{J}$ é o valor de deslocamento da $j^{ésima}$ junta. Portanto, f_1 e f_2 são os objetivos a serem otimizados.

Para cada junta inserida pelo usuário são geradas duas equações de restrição para que o valor da junta permaneça entre os valores solicitados, da mesma forma como realizado usando o algoritmo genético. Estas restrições são calculadas conforme as Equações 4.6 e 4.7, sendo lb o limite mínimo, ub o limite máximo e $\mathcal{J}$ o valor gerado para a junta, que pode ser α, θ ou d, se a junta for de torção, rotação ou deslocamento, respectivamente.

$$g_{min} = lb - \mathcal{J} \tag{4.6}$$

$$g_{max} = \mathcal{J} - ub \tag{4.7}$$

Outra restrição que o indivíduo deve obedecer é o valor de acurácia do erro de posição, que neste trabalho foi utilizado como 1.0. Portanto, o indivíduo é penalizado se o

erro de posição ($\mathcal{E}_p$) for maior que o valor da acurácia ($\mathcal{A}$), conforme a Equação 4.8.

$$g_{acc} = \mathcal{E}_p - \mathcal{A} \tag{4.8}$$

Desta maneira, existirão $2j + 1$ equações de restrições, sendo j a quantidade de juntas no manipulador. A penalidade $\mathcal{P}$ é dada pela Equação 4.9, onde r_p é a constante de penalidade que neste trabalho possui valor de 10^{10} e $g(p)$ é o valor da $p^{ésima}$ função de restrição.

$$\mathcal{P} = \sum_{p=1}^{2j+1} max(0, g(p))r_p \tag{4.9}$$

O valor final da função para o objetivo m (F_m) é o valor da minimização somada com a penalidade, de acordo com a Equação (4.10).

$$F_m = f_m + \mathcal{P} \tag{4.10}$$

A escolha do algoritmo NSGA-II é devida a três principais fatores: existência de elitismo, a enfatização das soluções não-dominadas, e por seu mecanismo de preservação de diversidade denominado distância da multidão. O indivíduo gerado é um vetor de tamanho j, sendo j a quantidade de juntas inseridas no manipulador. A ordem dos deslocamentos no vetor segue a mesma ordem de inserção das juntas, da base ao efetuador.

A seguir é realizada a seleção dos indivíduos seguindo os critérios explicitados na subseção 3.3.2, separando-os em fronteiras de dominância, realizando o torneio e por fim utilizando o cálculo da distância de multidão. Após selecionados, os indivíduos tem seus cromossomos cruzados utilizando os métodos de Radcliffe e Wright, e em seguida passam pelo processo de mutação dos genes.

A definição dos melhores parâmetros de otimização foi realizada a partir da análise de sensibilidade envolvendo o número de gerações, a quantidade de indivíduos em cada população, a probabilidade de cruzamento e a probabilidade de mutação. Os valores utilizados na análise de sensibilidade encontram-se descritos na Tabela 3 e foram testados para cada um dos dois tipos de cruzamento.

Tabela 3 – Valores utilizados na análise de sensibilidade para os parâmetros de otimização.

Parâmetro	**Valores para teste de sensibilidade**
Número de gerações	1000, 3000, 5000, 7000, 9000, 11000
Número de indivíduos	10, 20, 30, 40, 50
Probabilidade de cruzamento (p_c)	0.6, 0.7, 0.8
Probabilidade de mutação (p_m)	0.05, 0.15, 0.25, 0.35, 0.45, 0.55

Foram utilizados quatro modelos distintos de manipulador para a análise de sensibilidade, cujas anatomias e restrições se encontram no apêndice B, totalizando 4320 testes com a semente randômica igual a 42. Os dados obtidos na análise de sensibilidade são salvos em arquivos de extensão json contendo todos os parâmetros de otimização e os valores dos objetivos.

Foi implementado um script em linguagem Python que lê este arquivo de dados e realiza o cálculo estatístico da moda entre os parâmetros do melhor indivíduo de cada manipulador. Os resultados obtidos por ele como os melhores parâmetros de otimização, são utilizados para análise dos resultados.

O número de indivíduos e gerações diferiram na análise de sensibilidade, então nos testes para resultados foram utilizados os maiores valores para estes parâmetros. Assim, foram utilizados os seguintes parâmetros de otimização: 9000 gerações, 50 indivíduos, probabilidade de cruzamento de 0,6 e probabilidade de mutação de 0,55.

Nos testes realizados, o método proposto por Radcliffe (1992) possui um erro médio de posição de $0,41557$ cm, enquanto o método proposto por Wright (1991) tem como erro médio $0,36327$ cm. Apesar do método de Radcliffe apresentar maiores erros de posição, ele prioriza a minimização das juntas, sendo seus valores de junta aproximadamente 15 % menores que utilizando o método de Wright.

Nota-se que o erro de posição obtido em muitos casos foi maior que o ideal. Isto ocorre devido à variação dos parâmetros conforme a anatomia do braço robótico, o que pode ser claramente observado nos gráficos do apêndice B. Com o intuito de sanar este problema, seria necessário implementar uma rotina para encontrar os parâmetros ideais para a anatomia escolhida. Porém, isto poderia acarretar em efeito sistemático no erro, que em alguns casos o tornaria ainda maior.

Desta maneira, optou-se por utilizar um novo método que seja independente de parâmetros voláteis a fim de minimizar a ação do erro por escolha de parâmetros. Na seção 4.4 será descrita a abordagem utilizada com o novo algoritmo.

4.4 Minimização de trajeto com D* Lite

Da subseção 3.3.3 sabe-se que é necessário realizar o mapeamento para conhecimento de pontos vizinhos antes de iniciar o cálculo da trajetória. A forma de mapeamento é descrita na subseção 4.4.1 e a lógica para encontrar o trajeto é especificada na subseção 4.4.2.

4.4.1 Mapeamento

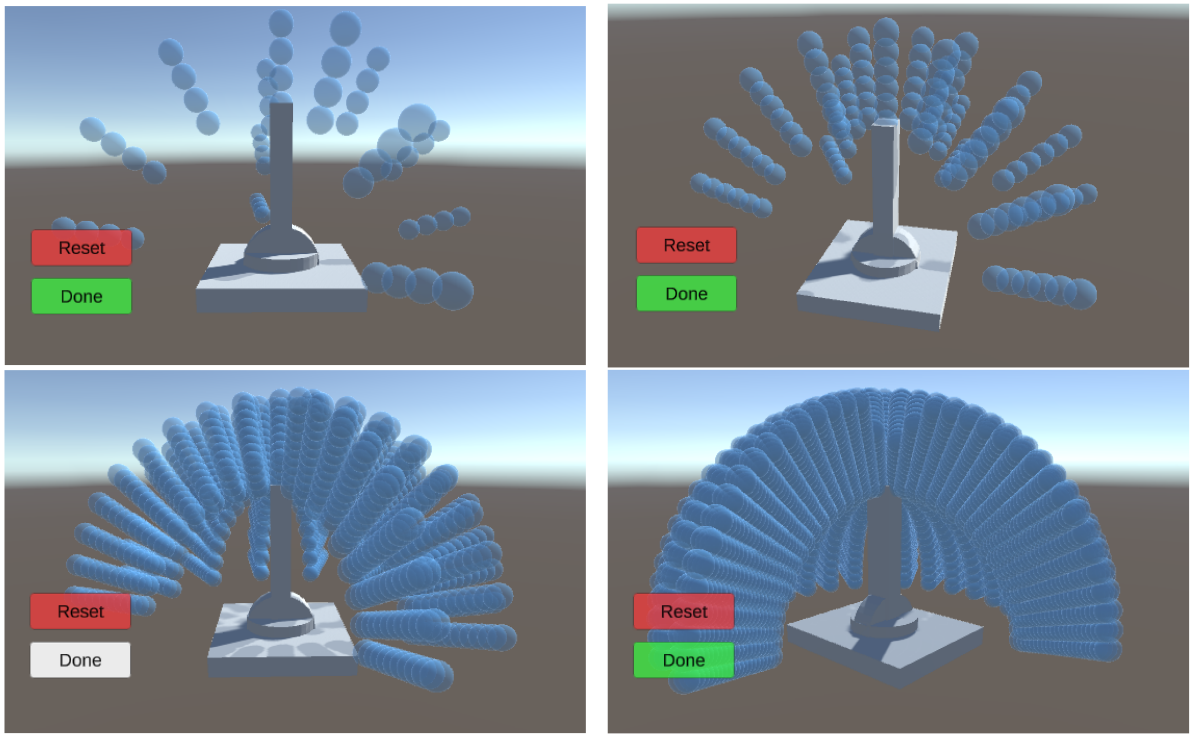
O algoritmo do D* Lite é baseado em grids, o que permite que se conheça os pontos vizinhos a cada nó do grid. Geralmente, o ambiente todo é dividido em uma malha com espaçamentos iguais, porém neste trabalho gerou-se o grid apenas dentro do espaço de trabalho do robô a partir das coordenadas de junta a fim de se garantir já no mapeamento que o robô trabalhe apenas dentro do espaço permitido.

Para criar os possíveis pontos de controle do trajeto, define-se o valor de resolução (rc), o qual influenciará na quantidade de pontos gerados no espaço das juntas e a distância entre eles. Menores valores garantem mais pontos, que levam a trajetos mais suaves mas com maior esforço computacional.

O tamanho do passo de deslocamento para cada junta (dj) é dado pela equação 4.11. A figura 21 ilustra como a variação do coeficiente de resolução influencia nos pontos do mapeamento. Na imagem do canto superior esquerdo foi utilizado $rc = 30$, na do canto superior direito $rc = 20$, na figura do canto inferior esquerdo $rc = 10$ e na do canto inferior direito o coeficiente $rc = 5$.

$$dj = (rc/100.0) * (ub - lb) \tag{4.11}$$

Figura 21 – Influência do coeficiente de resolução.



Fonte: Autoria própria

Os pontos gerados serão, desta maneira, as combinações de valores de juntas entre lb e ub espaçados de dj para cada uma das juntas do manipulador. De posse destes pontos, é necessário ainda definir os pontos vizinhos a cada um deles.

A função que encontra os vizinhos recebe o vetor de valores de juntas do ponto atual (v_{ref}). Para cada junta que não seja um link e tenha $dj > 0$, se o valor de junta $\mathcal{J} + i \cdot dj \leq ub$, então $\mathcal{J} + i \cdot dj$ é adicionado ao vetor de valores da junta atual v_j. Da mesma forma, se $\mathcal{J} - i \cdot dj \geq lb$, então $\mathcal{J} - i \cdot dj$ é adicionado a v_j, sendo i um número inteiro começando em zero até chegar nos limites de junta.

Tendo realizado o procedimento supracitado para cada uma das juntas existentes no braço robótico, é calculado o produto cartesiano entre os vetores, gerando todas as combinações possíveis de deslocamento das juntas para a anatomia analisada com o cr determinado. Daí, para cada coordenada de junta dada pelo cálculo do produto cartesiano, é calculada a posição no espaço cartesiano à qual elas são referentes, utilizando-se a matriz T de DH.

Se a coordenada da posição não existir no vetor de coordenadas do espaço de trabalho, ela é adicionada ao vetor, o nó é gerado, e a coordenada de junta é adicionada ao vetor de posições de juntas existente na classe do nó com seu valor total de deslocamento como custo. Caso contrário, apenas a coordenada de junta é inserida no vetor do nó já existente para a coordenada cartesiana.

Ao final deste procedimento, são analisados os pontos de partida e destino do caminho inseridos pelo usuário. Se estes não existirem no espaço de trabalho, os mesmos são adicionados e seus nós são criados.

Assim, todos os pontos cartesianos do espaço de trabalho do manipulador desejado estão contidos em um vetor (v_{et}). Para cada ponto, existe um nó referente a ele, com seus devidos atributos, que serão abordados no tópico 4.4.2.2. A partir deles é possível calcular os nós vizinhos.

Para isto, inicialmente são escolhidos aleatoriamente metade das coordenadas armazenadas em v_{et} e armazenadas em um novo vetor ($v_{amostras}$). Em seguida é calculado um valor de raio (r_{viz}), o qual é dado pelo valor da menor distância entre um ponto de v_{et} e os de $v_{amostras}$.

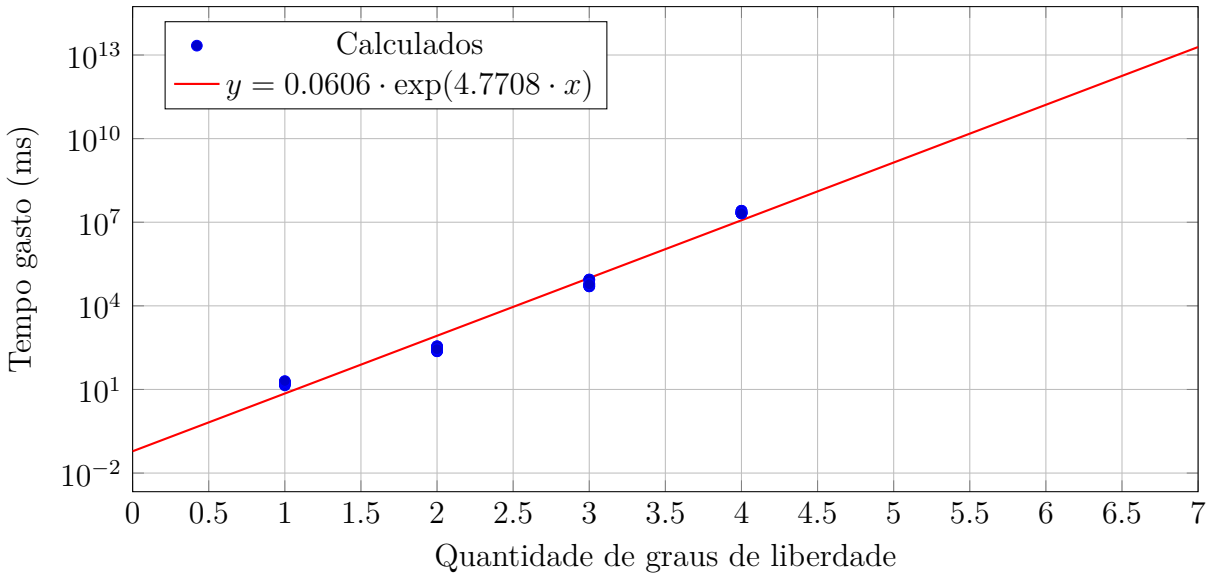
Para cada coordenada p_i existente em v_{et}, são calculados todos os outros pontos que estejam dentro de uma circunferência de raio r_{viz} centrada em p_i. Se a quantidade de pontos encontrados for maior que a quantidade de juntas, estes são definidos como os pontos vizinhos do nó referente a p_i. Senão, o raio é recalculado e o mapeamento refeito para p_i.

Este procedimento é realizado até que todos os nós possuam pelo menos o mínimo número de nós vizinhos. Ao finalizar este procedimento, a malha do mapa que será

utilizado pelo algoritmo se encontra completa.

Os nós inicialmente eram armazenados em vetores, os quais eram percorridos por laços do tipo *for* para encontrar os nós vizinhos conforme sobredito. Porém, o custo computacional deste método era muito alto, como pode-se notar pelo gráfico da Figura 22.

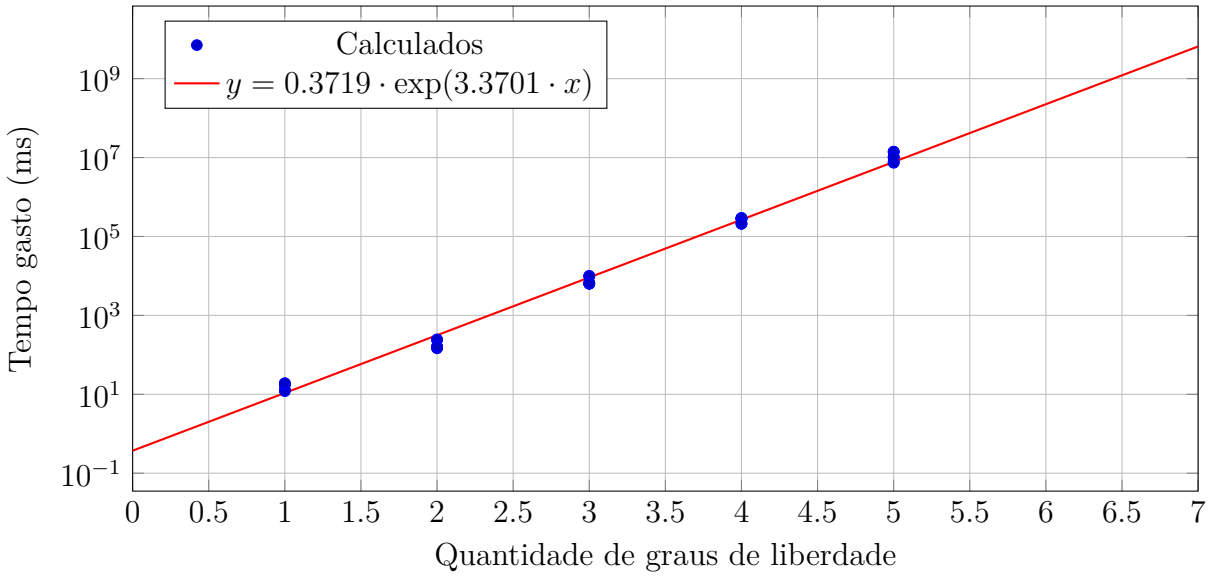
Figura 22 – Gráfico de tempo em milissegundos por quantidade de graus de liberdade usando laço de repetição do tipo for.



Fonte: Autoria própria

Assim, escolheu-se armazenar os nós em um dicionário, pois este trabalha com chave-valor. Isto significa que é possível pesquisar se existe um valor e atualizá-lo mais rapidamente do que com um array, já que neste último é necessário percorrer N elementos para encontrar um valor específico. Por este motivo, o tempo gasto para encontrar os nós é otimizado com o uso do dicionário. Os novos valores de tempo gastos são retratados no gráfico da Figura 23.

Figura 23 – Gráfico de tempo em milissegundos por quantidade de graus de liberdade usando dicionário.



Fonte: Autoria própria

Os gráficos das figuras 22 e 23 foram traçados com base em testes realizados utilizando um computador Intel(R) Core(TM) i7-3770 CPU @ 3.4GHz rodando sistema operacional Windows 10 Pro de 64 bits com 2GB de memória RAM livres. Foram realizados três testes para cada modelo de braço robótico, sendo três modelos para cada quantidade de graus de liberdade (1, 2, 3, 4 e 5 G.L.). Os valores obtidos para cada teste em milissegundos e os modelos utilizados são descritos no Apêndice C.

Apesar de haver uma melhora significativa na utilização do dicionário, o tempo de mapeamento ainda é grande conforme aumenta a quantidade de graus de liberdade do robô. Assim, pode-se afirmar que o mapeamento possui limitações que se tornam maiores quanto mais complexa for a anatomia do manipulador.

4.4.2 Cálculo da trajetória

Após mapear o espaço de trabalho, pode-se iniciar o algoritmo do D* Lite para cálculo da trajetória. Inicialmente, foi implementado o algoritmo para traçar trajetórias entre dois pontos em situações livres de obstáculos. A seguir, foram inseridos obstáculos estáticos e por fim obstáculos dinâmicos.

O algoritmo do D* Lite utiliza o conceito de *priority queue*. Esta é uma maneira de realizar inserções e modificações nas listas de indivíduos a partir de suas prioridades e armazená-los. Para maior rapidez, os conjuntos (Node, prioridade) são armazenados em uma *hash table* que é uma lista ordenada por prioridades em ordem crescente.

O algoritmo foi programado em linguagem Python e utiliza duas classes muito importantes, que são a classe *Node* e a *PriorityQueue*. A primeira delas é onde podem ser acessados atributos próprios a um determinado nó e calculados os valores de custo e vizinhança de cada nó, os quais serão abordados na sub-subseção 4.4.2.2. Já a classe *PriorityQueue*, cujos métodos são explanados na sub-subseção 4.4.2.1, contém os métodos responsáveis por armazenar e fazer alterações na lista ordenada de prioridades.

4.4.2.1 Priority Queue

A referida classe possui um atributo próprio denominado U que é uma *hash table* referente à lista de prioridades do D* Lite. Os seis métodos próprios da classe manipulam a lista prioritária U e são abordados a seguir.

- **Empty():**
  Retorna *True* se U estiver vazia e *False* caso contrário.
- **Exists(state):**
  Retorna *True* se o nó *state* está em U e *False* caso contrário.
- **Insert(vertex, priority):**
  Insere em U um tuple contendo o nó (vertex) e o vetor de valores das prioridades referentes ao nó (priority).
- **Pop():**
  Retira de U o nó com menor prioridade e retorna o nó eliminado.
- **Remove(state):**
  Exclui o nó *state* de U.
- **Top():**
  Retorna o nó que possui menor valor de prioridade. Caso a lista U esteja vazia, esta função retorna *False*.
- **TopKey():**
  Retorna a menor prioridade dentre as inseridas em U. Se U estiver vazia, retorna um vetor de infinitos.
- **Update(state, priority):**
  Atualiza o vetor de prioridades do nó *state* com o vetor recebido.

Com o uso dos métodos explicitados acima, a manipulação da lista prioritária é mais simples e rápida de se realizar. Os nós armazenados em U, possuem todas as informações necessárias do estado, as quais são explanadas na sub-subseção 4.4.2.2.

4.4.2.2 Node

A classe Node é a que manipula operações e atributos próprios aos nós. Primeiramente serão descritos os atributos que cada nó possui e o que cada uma deles representa.

- **k:**
  Vetor de custos.
- **g:**
  Valor de custo do nó.
- **rhs:**
  Valor de custo que integra informações da vizinhança imediata do nó.
- **neighbours:**
  Vetor contendo os nós vizinhos.
- **coordinate:**
  Tuple da coordenada cartesiana do nó.
- **walkable:**
  Variável booleana utilizada para checar a existência de obstáculos. Se o nó for obstruído pelos obstáculos, ela é setada em *False*. Do contrário, ela permanece como *True*.
- **jointVector:**
  Atributo referente à *hash table* que é usado para manipular as coordenadas de junta e seus custos para o nó. Possui as mesmas funções da *PriorityQueue*.
- **cost_multiplier:**
  Utilizado como multiplicador do custo, usado para penalização de nós próximos aos obstáculos. No momento da geração do nó, ele tem valor igual a 1.

Além dos atributos supramencionados, a classe *Node* possui também métodos que manipulam atributos relacionados ao nó. Estes métodos são descritos abaixo.

- **cost(neighbour):**
  Calcula o custo, que é a distância euclideana entre as coordenadas do nó e do nó vizinho recebido (*neighbour*) multiplicada por *cost_multiplier*, e retorna este valor. Após a inserção de obstáculos, se existe obstáculos entre os dois nós, o custo será infinito.

- **costForAllNeighbours():**
  Calcula o custo entre o nó e cada vizinho dele e os armazena no vetor k.

- **costToMove(neighbour):**
  Calcula o custo para mover do nó para o nó vizinho (*neighbour*), que é o custo g do *neighbour* somado ao retorno do método *cost* entre o nó e o vizinho. Este método retorna o valor da soma calculada.

- **findBestNeighbour():**
  Encontra o nó vizinho com menor custo para mover do atual para ele, e o retorna.

- **heuristic(startState):**
  Calcula o valor da heurística , que é a distância euclideana entre a coordenada do nó e a coordenada que o efetuador se encontra, e o retorna.

- **rhs_value():**
  Calcula o valor rhs do nó, que é o custo para mover até o vizinho com menor custo somado ao custo g deste vizinho.

Conhecendo os atributos e métodos dos nós, pode-se então compreender o algoritmo D* Lite utilizado e as modificações realizadas no algoritmo original, as quais são explanadas na sub-subseção 4.4.2.3.

4.4.2.3 D* Lite

O algoritmo POM implementado com base no D* Lite segue o algoritmo 7 com exceção de algumas modificações necessárias para aperfeiçoar o método de prevenção de colisões. Todas as modificações e sua importância serão explicitadas nesta sub-subseção.

A primeira destas alterações é necessária pois no algoritmo original só existe checagem da existência de obstáculos após o primeiro movimento. Isto gera um problema no caso em que há inserção de obstáculo após a inicialização do algoritmo e antes do primeiro movimento do manipulador. Por este motivo, assim que a etapa de inicialização é finalizada, inseriu-se uma checagem de possíveis alterações nos custos, sanando este problema.

Outra necessidade no caso de trabalho com robôs manipuladores é evitar colisões do obstáculo com o corpo do robô. A fim de prevenir que isto ocorra, são traçados cilindros entre a posição no espaço cartesiano de uma junta j e a da junta $j + 1$ e realiza-se a checagem se o ponto no qual o obstáculo se encontra (p_{obs}) coincide com alguma parte do cilindro de raio r.

Para isto, inicialmente verifica-se se 4.12 e 4.13 são verdadeiras. Isto irá confirmar que o obstáculo fica entre os planos das duas facetas circulares do cilindro.

$$(p_{obs} - p_j) \cdot (p_{j+1} - p_j) \geq 0 \tag{4.12}$$

$$(p_{obs} - p_{j+1}) \cdot (p_{j+1} - p_j) \leq 0 \tag{4.13}$$

Verifica-se então se 4.14 é verdadeira, o que irá garantir que p_{obs} está dentro da superfície curva do cilindro.

$$\frac{|(p_{obs} - p_j) \cdot (p_{j+1} - p_j)|}{|p_{j+1} - p_j|} \leq r \tag{4.14}$$

Se todas as verificações acima forem corretas, então o obstáculo colide com o corpo do robô. Portanto, o custo para mover neste caso é setado para infinito, garantindo que o robô não realize o trajeto utilizando esta configuração.

Outra mudança realizada foi o vetor de prioridades. No algoritmo original, o vetor é conforme a Equação 4.15. Na lista prioritária, os nós são ordenados conforme estes valores. O primeiro valor do vetor é prioritário, sendo o segundo valor parte da análise apenas quando há mais de um caso com mesmo valor no primeiro índice do vetor.

$$[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m;\ \min(g(s), rhs(s))] \tag{4.15}$$

Porém, decidiu-se inserir um terceiro valor de prioridade, como mostra a Equação 4.16. Este novo valor adiciona ao primeiro a somatória dos valores de juntas multiplicada por 0.1. Assim, se o primeiro e o segundo valor das prioridades forem idênticas para dois pontos distintos, o nó com menor deslocamento de junta é o que prevalece.

$$\begin{aligned}[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m;\ \min(g(s), rhs(s));\\ \min(g(s), rhs(s)) + h(s_{start}, s) + k_m + jointValue(s)]\end{aligned} \tag{4.16}$$

4.4.3 Resultado da utilização do algoritmo

Ao migrar para o algoritmo D* Lite realizando as alterações supracitadas, consegue-se atingir uma precisão da ordem de 10^{-3} para anatomias de até cinco graus de liberdade. Para anatomias mais complexas, o custo de mapeamento se torna elevado, impossibilitando a realização de testes para estes valores, considerando os recursos utilizados. Além disso, os pontos de passagem calculados garantem um trajeto suave no espaço cartesiano.

Porém, ainda assim o algoritmo sofre com os problemas de singularidades, realizando movimentos bruscos em alguns casos, ou mesmo sendo incapaz de mover. Isto ocorre pois a associação da escolha de juntas em casos de singularidades à que realiza menor esforço, leva o manipulador a muitas vezes a priorizar o valor do deslocamento de junta em detrimento do melhor caminho.

Por este motivo, como deseja-se encontrar um caminho que seja suave tanto no espaço cartesiano como no espaço de juntas, optou-se por inserir um novo método na escolha dos deslocamentos de junta. Na seção 4.5 será abordada a combinação do método modificado do D* Lite com o CCD.

4.5 Minimização de trajeto com deslocamento de juntas suaves

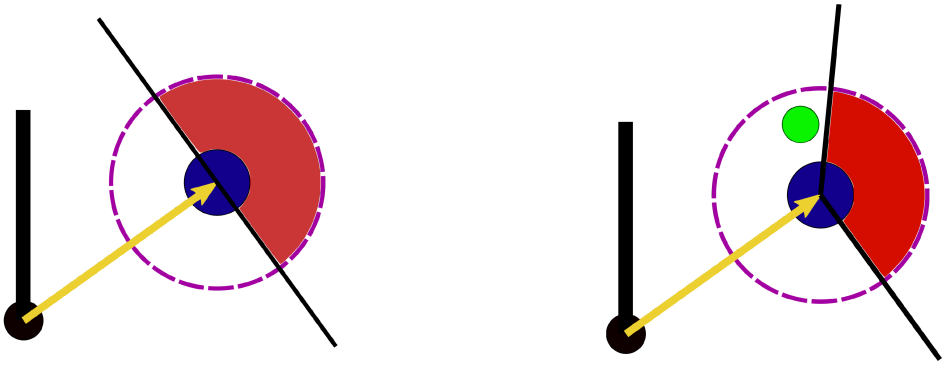
As modificações no algoritmo original do D* Lite relacionadas à prevenção de colisões antes do primeiro movimento e com o corpo do robô são mantidas. Já o vetor de prioridades, que na Seção 4.4 havia sido modificado, agora retorna ao modelo original do algoritmo, descrito pela Equação 4.17.

$$[\min\left(g(s), rhs(s)\right) + h(s_{start}, s) + k_m;\ \min\left(g(s), rhs(s)\right)] \tag{4.17}$$

Ainda nos esforços para evitar colisões, foi inserida a utilização da variável *cost_multiplier* para evitar que o trajeto seja calculado passando por cima do obstáculo, o que poderia forçar o manipulador a ter movimentos de colisão. Neste caso, a modificação realizada anteriormente para impedir a colisão do objeto com o corpo não permite o movimento, causando impossibilidade de movimentação.

A fim de sanar este problema, para cada obstáculo presente no espaço de trabalho do manipulador, é traçado um vetor da base do manipulador ao centro do obstáculo ($\overrightarrow{v_{BO}}$), conforme representado na cor amarela na Figura 24. Em seguida é calculado usando a Equação 4.18 o tamanho do raio (r) que será utilizado posteriormente para a penalização. Nela, *obs_sizes* é um *array* que contém o tamanho do obstáculo nos eixos x, y e z, e h é a somatória dos tamanhos dos elos do manipulador.

Figura 24 – Representação da escolha dos nós que serão penalizados.



Fonte: Autoria própria

$$r = \frac{\exp\left(\max(obs\_sizes\,[i])\right)}{h} \tag{4.18}$$

De posse do raio, é realizada a checagem se os pontos de início (p_{inicio}) ou de fim (p_{final}) do trajeto se encontram dentro da esfera com o raio calculado, centrada no ponto central do obstáculo. A esfera é representada na Figura 24 pela linha tracejada lilás. Na ilustração à esquerda os pontos não se encontram no interior da circunferência, ao contrário da imagem à direita, onde o ponto retratado na cor verde situa-se no interior da circunferência analisada.

Neste último caso, um vetor é traçado entre o centro do obstáculo e o ponto que se encontra no interior da circunferência. Em seguida é calculado o ângulo formado entre $\overrightarrow{v_{BO}}$ e esse vetor, sendo θ_{BI} o ângulo ao vetor do ponto de início e θ_{BF} o ângulo ao vetor do ponto final. Caso o ângulo encontrado esteja entre 90° e 270°, o valor mais próximo será substituído pelo valor do ângulo defasado de 10°. Estes serão os valores de limites de ângulos, os quais seguem as condições abaixo.

- Se p_{inicio} e p_{final} não se encontram dentro da circunferência: $v_{limites} = [90°, 270°]$
- Se p_{inicio} está dentro da circunferência e $90° \leq \theta_{BI} \leq 270°$
  - Se θ_{BI} está mais próximo a 90°: $v_{limites} = [(\theta_{BI} - 10)°, 270°]$
  - Se θ_{BI} está mais próximo a 270°: $v_{limites} = [90°, (\theta_{BI} + 10)°]$
- Se p_{final} está dentro da circunferência e $90° \leq \theta_{BF} \leq 270°$
  - Se θ_{BF} está mais próximo a 90°: $v_{limites} = [(\theta_{BF} + 10)°, 270°]$
  - Se θ_{BF} está mais próximo a 270°: $v_{limites} = [90°, (\theta_{BF} - 10)°]$

Após o cálculo dos limites de ângulos, todos os pontos cartesianos são analisados e os que se encontram dentro da circunferência de raio r e não são pontos intrespassáveis

passam pela análise de ângulo. Nela, é calculado o ângulo θ_{BP} entre $\overrightarrow{v_{BO}}$ e um vetor traçado do centro do obstáculo ao ponto analisado. Se θ_{BP} estiver entre os ângulos em $v_{limites}$, então a variável *cost_multiplier* do nó é recalculada de acordo com a Equação 4.19.

$$cost\_multiplier = \exp\left(\frac{\theta_{BP}}{10}\right) \tag{4.19}$$

Desta maneira, os nós cuja variável *cost_multiplier* foi modificada é ilustrada na Figura 24 pela área vermelha. Os nós fora desta área permanecem com $cost\_multiplier = 1$ a menos que seja intrespassável, e neste caso $cost\_multiplier = \infty$.

Assim, o novo código do POM segue a ideia do Algoritmo 10, onde a cada iteração, atualiza-se o s_{start} com o melhor nó vizinho ao nó onde o manipulador se encontra, utilizando o D* Lite. Em seguida realiza-se a checagem da existência de obstáculos no ambiente e a variável K_m é atualizada.

Posteriormente, é averiguado se existem nós que na iteração anterior eram obstruídos por obstáculos mas foram desobstruídos na iteração atual. Caso isto seja verdadeiro, a variável *walkable* retorna para *True*, o valor de *rhs* é recalculado e o custo g é igualado ao novo custo rhs para cada um destes nós.

Em seguida, se houver obstáculos no ambiente, o algoritmo é semelhante às linhas 66 a 80 do Algoritmo 7, que é o proposto por Koenig e Likhachev (2002). Então, a coordenada cartesiana para onde o efetuador deve se mover é enviada ao CCD, que retorna as coordenadas de junta que o manipulador deve deslocar para atingir o ponto.

Algorithm 10 Código principal do algoritmo POM.

```
1:  procedure MAIN( )
2:      s_start ← s_last.FINDBESTNEIGHBOUR()
3:      newObstacles ← FINDOBSTACLES(True)      ▷ Refaz o pedido dos nós em colisão
4:      Km ← Km + heuristic(s_last, s_start)
5:      if newObstacles ≠ None then
6:          freeAgain ← Nós que eram obstáculos e não são mais
7:          for all nodeFree ∈ freeAgain do
8:              nodeFree.walkable ← True
9:              nodeFree.RHS_VALUE()
10:             nodeFree.g ← nodeFree.rhs
11:         end for
12:         obstacles ← newObstacles
13:     end if
14:     for all u ∈ obstacles do
15:         Manipula os custos dos obstáculos          ▷ Linhas 67:78 do Algoritmo 7
16:     end for
```

17: ComputeShortestPath()
18: $went = \text{Move}(s_{start})$
19: **if** *went* **then**
20: $s_{last} \leftarrow s_{start}$
21: $path \leftarrow [path, s_{start}]$
22: **end if**
23: **end procedure**

O CCD recebe o nó s_{start} e conhece a posição das juntas e as variáveis de junta anteriores, com as quais calcula usando a matriz de DH a posição em que o efetuador se encontra. O alvo é a coordenada cartesiana de s_{start}.

Enquanto a distância euclideana entre o alvo e a coordenada cartesiana do efetuador for maior que a precisão desejada, percorre-se cada elemento inserido no manipulador. A precisão utilizada é de 10^{-3} e existe uma variável denominada *maxCount*, que inicialmente é dada pela quantidade de elementos inseridos no manipulador multiplicado por 2. Há outra variável nomeada *deltaCount* iniciada em zero.

O algoritmo é iniciado do efetuador à base até que o contador atinja o valor de *maxCount*, então o contador é zerado e iniciado da base ao efetuador. Então o valor de *maxCount* é incrementado conforme a Equação 4.20.

$$maxCount = maxCount + deltaCount * 5 \tag{4.20}$$

Para cada um deles, primeiramente realiza-se a checagem do tipo de elemento, que pode ser um elo, uma junta prismática ou uma junta de rotação. Se o elemento for um elo, o índex que percorre o vetor de posições das juntas é decrementado caso o CCD seja do efetuador à base, e incrementado caso o CCD vá da base ao efetuador.

No caso do elemento ser uma junta do tipo prismática, o Algoritmo 11 é executado. Nele, inicialmente o vetor unitário z_i é calculado, bem como os vetores $\vec{jt}$ e $\vec{je}$. Em seguida, a Equação 3.25 é calculada e passa pelas verificações de limites e colisões. Por fim, o índex é decrementado ou incrementado, sendo o CCD respectivamente do efetuador à base e da base ao efetuador.

Algorithm 11 Código do CCD para junta prismática no algoritmo POM.

1: **if** CCD do efetuador à base **then**
2: $z_i \leftarrow j_{index} - j_{index-1}$
3: **else**
4: $z_i \leftarrow j_{index+1} - j_{index}$
5: **end if**
6: $z_i \leftarrow z_i / ||z_i||$

7: $e \leftarrow \text{DH}(angulosAnteriores)$
8: $jt \leftarrow t - j$
9: $je \leftarrow e - j$
10: $\delta P \leftarrow jt - je$
11: $dis \leftarrow \delta P \cdot z_i$
12: **if** $dis > ub-$ tamanho do elo **then**
13: $dis \leftarrow ub-$ tamanho do elo
14: **end if**
15: $anguloCalculado \leftarrow angulosAnteriores + dis$
16: **if** $anguloCalculado > ub$ **then**
17: $anguloCalculado \leftarrow ub$
18: **else if** $anguloCalculado < lb$ **then**
19: $anguloCalculado \leftarrow lb$
20: **end if**
21: $angulos_i \leftarrow anguloCalculado$
22: **if** VerifyJoints($angulos$) **then**
23: $angulosAnteriores \leftarrow angulos$
24: **end if**
25: $e \leftarrow \text{DH}(angulos)$
26: **if** CCD do efetuador à base **then**
27: $index \leftarrow index - 1$
28: **else**
29: $index \leftarrow index + 1$
30: **end if**

Se o elemento não for um elo, nem uma junta prismática, ele é uma junta do tipo rotação. Neste caso, executa-se o Algoritmo 12 onde inicialmente são calculados j, e, $\vec{jt}$ e $\vec{je}$. A modificação realizada neste caso é o cálculo no plano, zerando o eixo onde não há alteração de acordo com cada tipo de junta, como realizado nas linhas 4 à 10 do código. O restante dele segue as equações definidas na teoria do CCD, seguido pela verificação dos limites e de colisões.

Algorithm 12 Código do CCD para junta rotacional no algoritmo POM.

1: **procedure** RotacionalCCD
2: $jt \leftarrow t - j$
3: $je \leftarrow e - j$
4: **if** Rotacao **then** ▷ Realiza-se o produto no plano YZ
5: $jt[0] \leftarrow 0$
6: $je[0] \leftarrow 0$
7: **else if** Torcao **then** ▷ Realiza-se o produto no plano XZ

8: $jt[1] \leftarrow 0$
9: $je[1] \leftarrow 0$
10: **end if**
11: Calcula θ pela Eq. 3.20
12: **if** $\theta > ub$ **then**
13: $\theta \leftarrow ub$
14: **end if**
15: Calcular produto vetorial da Equação 3.21
16: **if** Primeiro movimento & θ igual anterior **then**
17: $\theta \leftarrow \theta + 10$
18: **end if**
19: Calcular *dir* pela Equação 3.21
20: **if** dir > 0 **then** ▷ Relação da Equação 3.22
21: $\theta \leftarrow angulos_i + \theta$
22: **else**
23: $\theta \leftarrow angulos_i - \theta$
24: **end if**
25: **if** $\theta > ub$ **then**
26: $\theta \leftarrow ub$
27: **else if** $\theta < lb$ **then**
28: $\theta \leftarrow lb$
29: **end if**
30: $angulos_i \leftarrow \theta$
31: **if** VerifyAngle$(angulos, index)$ **then**
32: $angulosAnteriores \leftarrow angulos$
33: **else**
34: $angulosAnteriores \leftarrow$ Shake$(angulos, i, index)$
35: **end if**
36: $e \leftarrow$ DH$(angulos)$
37: **end procedure**

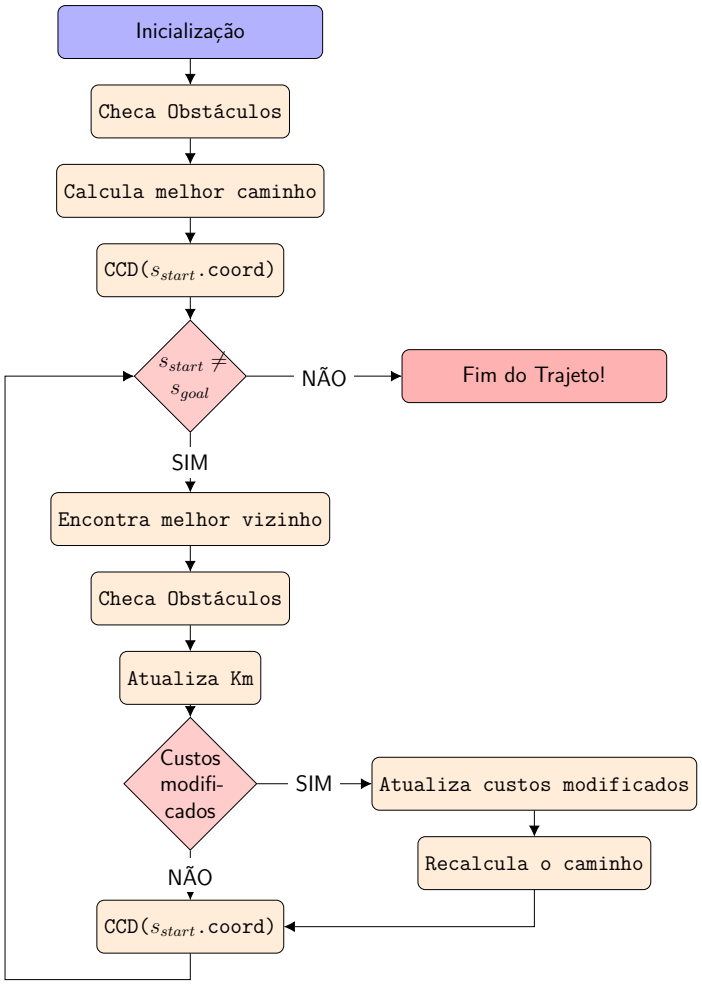
É importante ressaltar a utilização de duas funções utilizadas no Algoritmo 12: *VerifyAngle* e *Shake*. A primeira realiza a verificação de colisão com o corpo do robô apenas no elo afetado pelo movimento da junta que deseja-se alterar. Caso o movimento resulte em colisão, a última função é utilizada.

A função *Shake* gera um valor decimal contido entre os limites de junta estabelecidos até que o elo não esteja mais em colisão. Se após n_{max} iterações não foi encontrado um valor de junta que consiga evitar a colisão do elo, então os valores anteriores de junta são retornados. Esta função é importante em casos onde o algoritmo do CCD encontra-se

preso em uma configuração com a qual não é possível continuar o trajeto.

O algoritmo POM tem seu funcionamento descrito pelo fluxograma da Figura 25. Cada processo é realizado conforme descrito neste capítulo e no algoritmo 10. O primeiro processo realizado é a inicialização, no qual, além da inicialização das variáveis do algoritmo, também será realizado o mapeamento do ambiente, com a vizinhança e cálculo de custo dos nós do mapeamento. Este processo é melhor detalhado no fluxograma da Figura 26.

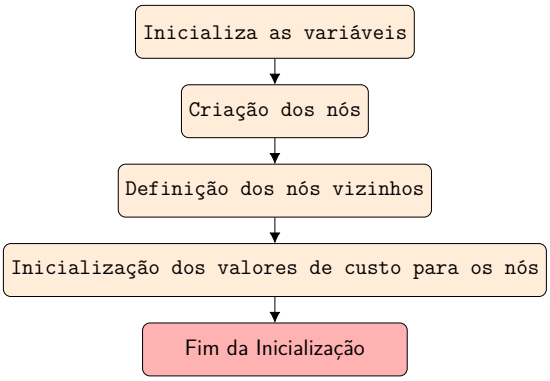
Figura 25 – Fluxograma geral de funcionamento do POM.



Fonte: Autoria própria

Após o processo de inicialização da Figura 26, é possível analisar se algum nó se encontra obstruído através do processo de checagem de obstáculos. A partir desta informação, calcula-se o melhor caminho e o próximo ponto de passagem desejado é enviado para o CCD, que deve retornar os valores de junta ideais para onde o manipulador deve mover. Enquanto não chegar ao fim do trajeto, isto é repetido com a atualização da variável modificadora K_m, além de atualizar os custos caso haja alterações no ambiente.

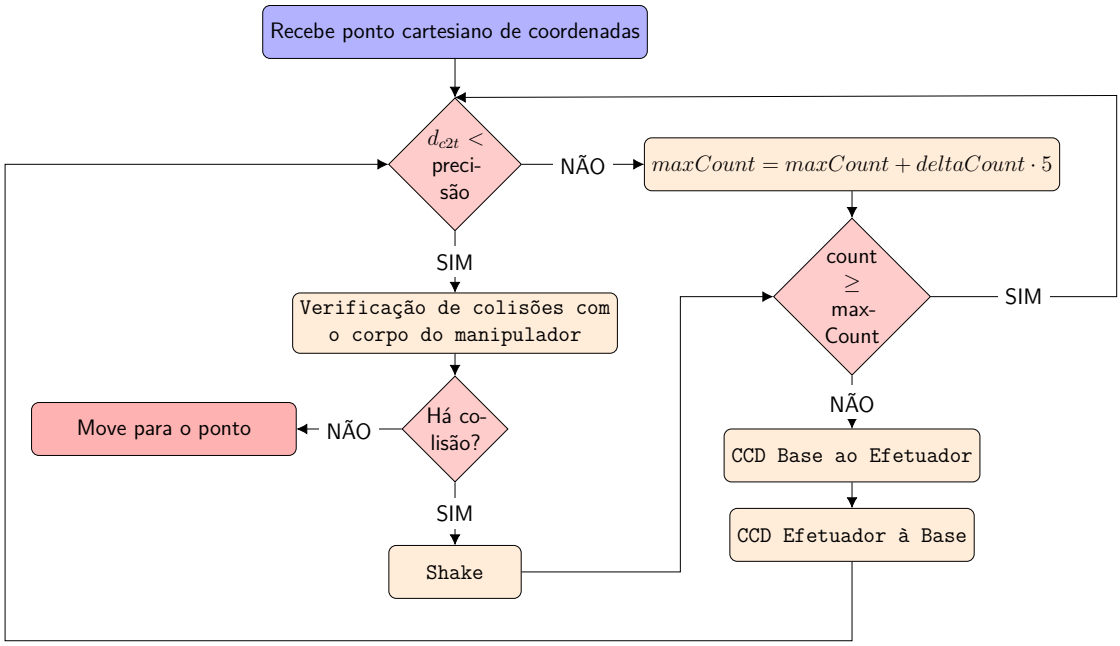
Figura 26 – Fluxograma do processo de inicialização do POM.



Fonte: Autoria própria

O processo do CCD utilizado no algoritmo POM aqui proposto segue o fluxograma da Figura 27. Os cálculos internos realizados são expostos nos algoritmos 11 e 12.

Figura 27 – Fluxograma detalhado do processo do CCD no POM.



Fonte: Autoria própria

Espera-se que o algoritmo POM aqui proposto, dado pela fusão do algoritmo D* Lite com o algoritmo CCD, garanta trajetos minimizados no espaço cartesiano, contornando as singularidades e garantindo uma movimentação mais suave no espaço das juntas. O D* Lite calcula um trajeto minimizado em termos espaciais, enquanto o CCD alcança os pontos de passagem com um deslocamento de junta ameno dentro dos requerimentos utilizados. Os testes realizados e resultados obtidos para a utilização do algoritmo POM são apresentados no Capítulo 5.

5 | Resultados e Discussões

Neste capítulo serão abordados os testes realizados a fim de analisar a eficiência do método proposto na minimização do caminho planejado em tempo real, bem como sua capacidade de desviar tanto de obstáculos estáticos como dinâmicos. Como este trabalho não considera parâmetros temporais, o conceito de dinamismo é relacionado à percepção do ambiente a cada iteração.

5.1 Ambientes com obstáculos estáticos

Nesta seção serão apresentados os testes realizados em ambientes livres de obstáculos ou na presença de obstáculos estáticos. O coeficiente de resolução utilizado em todos os testes seguintes é $rc = 10$.

5.1.1 Manipulador 3R em ambiente livre de obstáculos de Banga, Singh e Kumar (2007)

O primeiro dos testes realizados examina a capacidade de minimização de juntas do algoritmo enquanto minimiza posição, apesar deste não ser um objetivo do algoritmo aqui proposto. O artigo de Banga, Singh e Kumar (2007) utiliza a equação 5.1 como função de aptidão em um robô 3R. Os três links possuem tamanhos de 330 mm, 320 mm e 265 mm, respectivamente da base ao efetuador. Os limites de ângulos não são definidos pelo autor, portanto foi utilizado $-180° \leq \theta \leq 180°$.

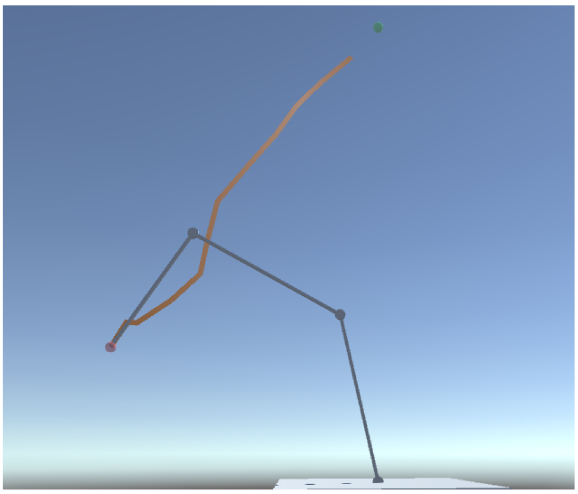
$$f(x) = \theta_1 * 2.99 + \theta_2 * 2.98 + \theta_3 * 2.99 \tag{5.1}$$

Banga, Singh e Kumar (2007) utilizam o AG combinado com o AHP e realizam a minimização entre o ponto de referência e o ponto $(0\text{ mm}, 250\text{ mm}, 500\text{ mm})$ em um ambiente 2D. São exibidos 32 valores de cromossomos com seus respectivos valores de aptidão, os quais possuem uma média de $761,85$ e o mínimo valor apresentado, de $375,68$.

A mesma anatomia de robô foi utilizada e o trajeto gerado entre os pontos utilizados por Banga, Singh e Kumar (2007) para gerar a minimização. Os valores obtidos utilizando o algoritmo POM, se encontram na Tabela 19 no Apêndice D. Calculando o valor de aptidão a partir dos ângulos encontrados para cada junta com base na Equação 5.1, encontra-se o valor $439,4405$ que é melhor que a média de Banga, Singh e Kumar (2007) e relativamente próximo ao mínimo encontrado pelos autores.

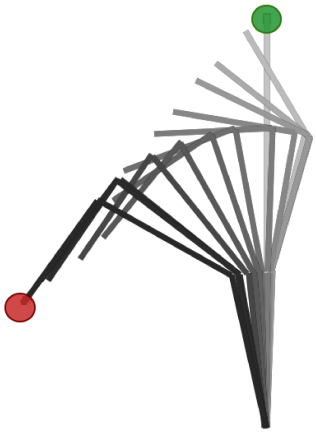
O trajeto completo calculado pelo algoritmo POM é retratado na Figura 28 e os movimentos realizados a cada ponto de controle são ilustrados na Figura 29. O tamanho total do trajeto tem $864,78$ mm e o deslocamento das juntas durante todo o trajeto é de $240,3960°$.

Figura 28 – Trajeto calculado pelo POM com o manipulador 3R sem obstáculos usado por Banga, Singh e Kumar (2007).



Fonte: Autoria própria

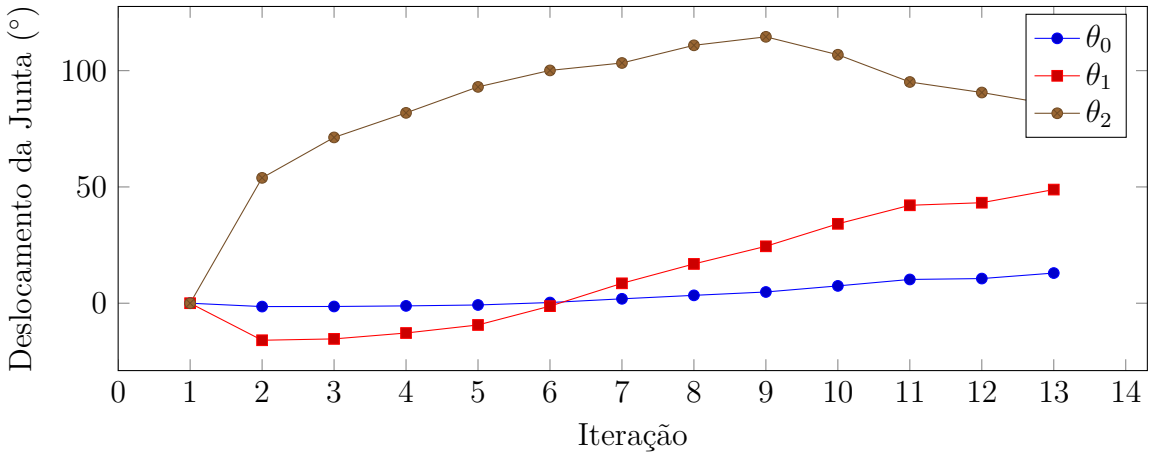
Figura 29 – Gráfico de movimento planejado pelo POM com o manipulador 3R sem obstáculos usado por Banga, Singh e Kumar (2007).



Fonte: Autoria própria

Ao analisar as Figuras 28 e 29, nota-se que o trajeto é otimizado enquanto mantém um caminho suave em relação às juntas, os quais são calculados de um ponto de controle a outro em tempo real. O deslocamento suave das juntas pode ser comprovado pela Figura 30 que ilustra os deslocamentos de cada junta em graus a cada ponto de controle calculado em cada iteração.

Figura 30 – Gráfico de deslocamento das juntas a partir do trajeto calculado pelo POM com o manipulador 3R sem obstáculos usado por Banga, Singh e Kumar (2007).



Fonte: Autoria própria

5.1.2 Manipulador 3R em duas etapas de Kazem, Mahdi e Oudah (2008)

Ainda na anatomia de 3 G.L., Kazem, Mahdi e Oudah (2008) propõem um manipulador semelhante ao de Banga, Singh e Kumar (2007), diferindo deste com relação aos tamanhos dos links. O manipulador de Kazem, Mahdi e Oudah (2008) possui $L_0 = 1,0$ m, $L_1 = 1,0$ m e $L_2 = 0,5$ m e seu primeiro experimento também é em ambiente livre de obstáculos, utilizando AG para a minimização de caminho e de deslocamento das juntas. O caminho deve ser planejado entre os pontos $(0\text{ m}, 2.3\text{ m}, 0\text{ m})$ e $(0\text{ m}, 0\text{ m}, -2\text{ m})$.

Savsani, Jhala e Savsani (2013) realizam o mesmo experimento utilizando os algoritmos ABC e TLBO. Os autores comparam ambos com os resultados de Kazem, Mahdi e Oudah (2008) em relação ao tamanho do trajeto e dos valores deslocados pelas juntas. A Tabela 4 compara os valores apresentados nos artigos de Kazem, Mahdi e Oudah (2008) e Savsani, Jhala e Savsani (2013) com os obtidos utilizando o algoritmo POM proposto neste trabalho.

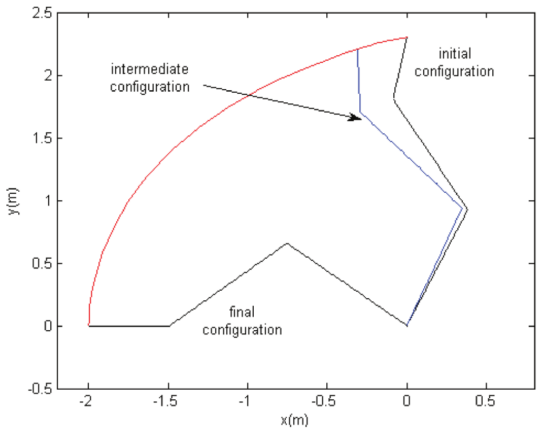
Tabela 4 – Tabela de resultados para o manipulador 3R em relação ao trajeto e às juntas.

	AG	**ABC**	**TLBO**	**POM**
f_{dist}	$3,28$ m	$3,3268$ m	$3,32$ m	$3,0892$ m
f_{joint}	$1,91$ rad	$1,8727$ rad	$1,884$ rad	$5,3018$ rad

A partir da Tabela 4 nota-se que o algoritmo POM cumpre bem seu objetivo de minimização do trajeto. Porém, por se tratar de um movimento calculado em tempo real, o manipulador suaviza os movimentos entre dois pontos de controle, mas na maioria dos casos não minimiza os deslocamentos de juntas na totalidade do caminho.

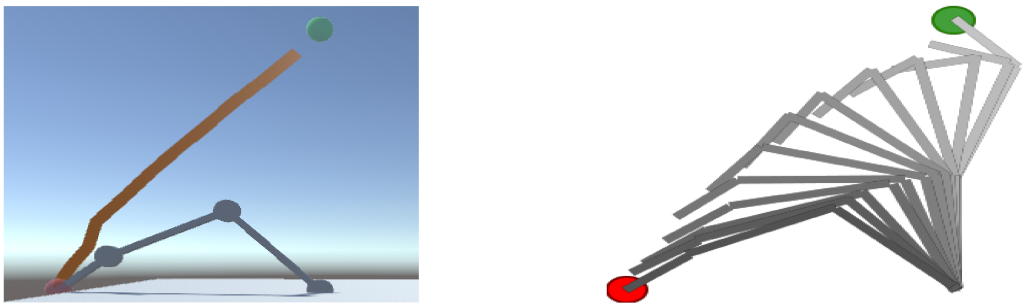
O trajeto final calculado por Kazem, Mahdi e Oudah (2008) é ilustrado no gráfico da Figura 31 e pode ser visualmente comparado ao calculado utilizando o POM. Este é representado à esquerda da Figura 32, a qual possui à direita o gráfico de cada movimento a cada ponto de controle calculado.

Figura 31 – Gráfico de movimento e trajeto apresentado por Kazem, Mahdi e Oudah (2008) em ambiente livre de obstáculos.



Fonte: Kazem, Mahdi e Oudah (2008).

Figura 32 – Representação do trajeto calculado pelo POM (a) e gráfico de movimento (b) do manipulador 3R em ambiente livre de obstáculos.

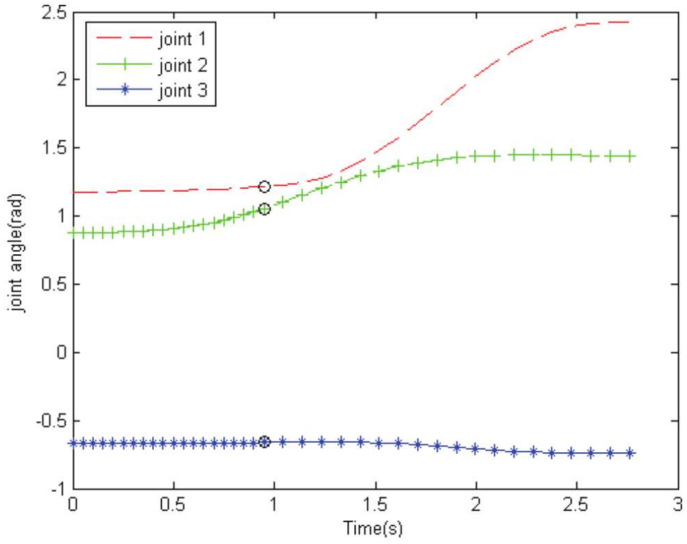


(a) (b)

Fonte: Autoria própria

O caminho em sua totalidade utilizando o algoritmo aqui proposto, possui um deslocamento de $303,7708°$ e fica mais explícito o deslocamento de cada junta através do gráfico. A Figura 33 mostra os deslocamentos ocorridos utilizando o AG no artigo de Kazem, Mahdi e Oudah (2008).

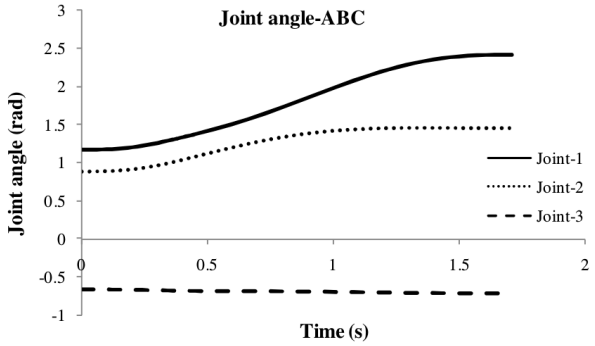
Figura 33 – Gráfico de deslocamento de juntas livre de obstáculos obtidos por Kazem, Mahdi e Oudah (2008) usando AG.



Fonte: Kazem, Mahdi e Oudah (2008)

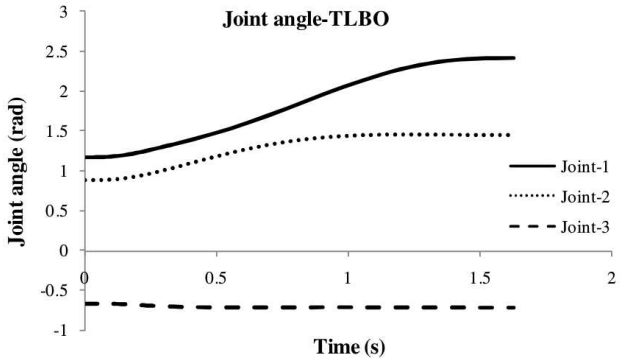
O gráfico de Kazem, Mahdi e Oudah (2008) ilustrado na Figura 33 é bem próximo aos gráficos de Savsani, Jhala e Savsani (2013), tanto para o ABC (Figura 34) quanto para o TLBO (Figura 35).

Figura 34 – Gráfico de deslocamento de juntas livre de obstáculos obtidos por Savsani, Jhala e Savsani (2013) usando ABC.



Fonte: Savsani, Jhala e Savsani (2013)

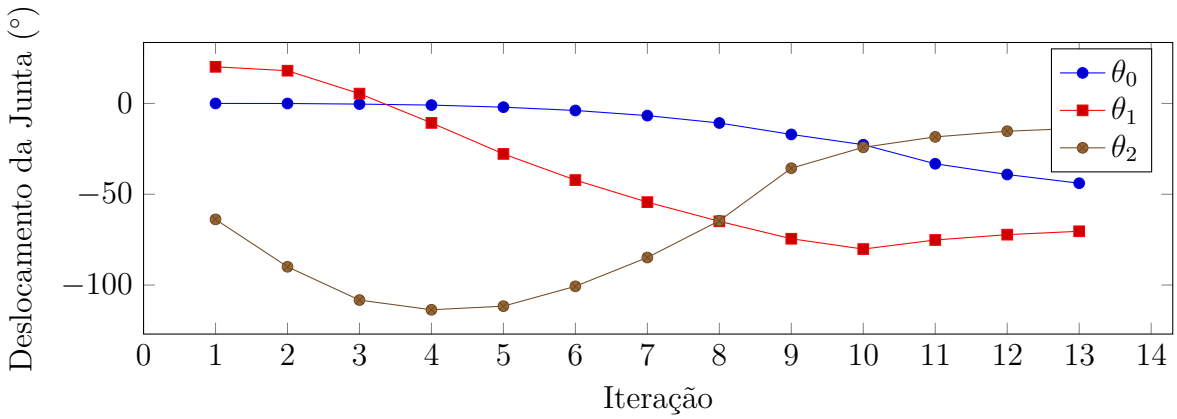
Figura 35 – Gráfico de deslocamento de juntas livre de obstáculos obtidos por Savsani, Jhala e Savsani (2013) usando TLBO.



Fonte: Savsani, Jhala e Savsani (2013)

Já o gráfico da Figura 36 difere bastante dos anteriores, os quais buscam minimização de juntas. Porém, os movimentos de cada junta são suaves, sem possuir picos ou cair em singularidades. Os valores dos ângulos a cada ponto de controle encontram-se na Tabela 20 do Apêndice D.

Figura 36 – Gráfico de deslocamento das juntas calculado pelo POM com o manipulador 3R em ambiente livre de obstáculos.



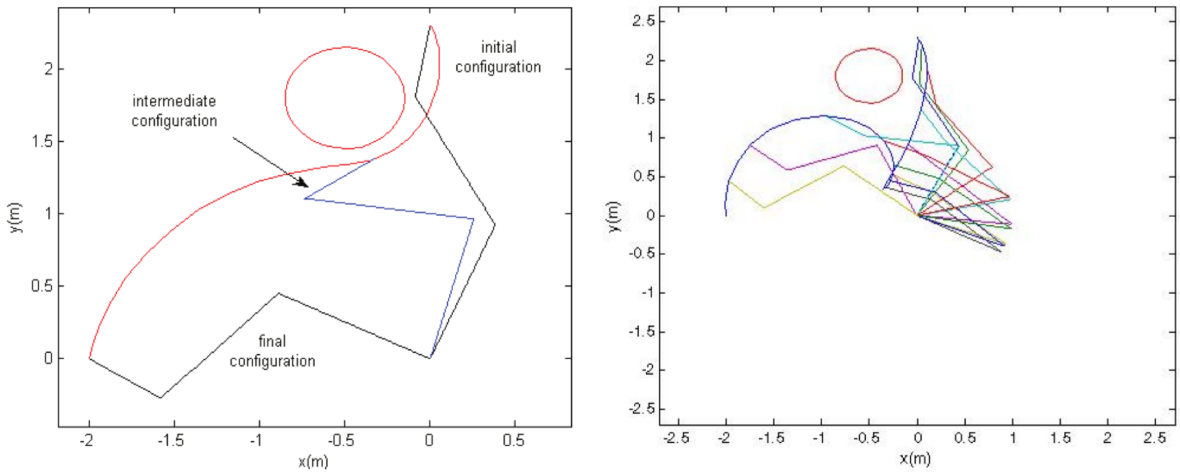
Fonte: Autoria própria

Assim, pode-se afirmar que o algoritmo POM funciona bem para minimização de trajetos em ambientes livres de obstáculos. O trajeto se mostrou eficaz, já que é $0,1908$ m menor que o do AG, $0,2376$ m menor que o do ABC e $0,2308$ m menor que o do TLBO. Além disto, os movimentos realizados pelo manipulador durante o trajeto são suaves.

O artigo de Kazem, Mahdi e Oudah (2008) também propõe um experimento com um obstáculo em formato de esfera com raio $r = 0,35\ m$ centrada no ponto $(0$ m, 1.8 m, -0.5 m$)$. Os pontos do trajeto são os mesmos do experimento livre de obstáculos, partindo do ponto $(0$ m, 2.3 m, 0 m$)$ até chegar no ponto $(0$ m, 0 m, -2 m$)$.

O mesmo experimento é realizado por Yeasmin, Shill e Paul (2017) e comparado ao de Kazem, Mahdi e Oudah (2008). Os trajetos encontrados por eles, bem como alguns movimentos realizados são ilustrados na Figura 37, onde à esquerda se encontra o trajeto de Kazem, Mahdi e Oudah (2008) e à direita, o trajeto encontrado por Yeasmin, Shill e Paul (2017) com o EPSO.

Figura 37 – Representação do trajeto calculado usando (a)AG e (b) EPSO com obstáculo.

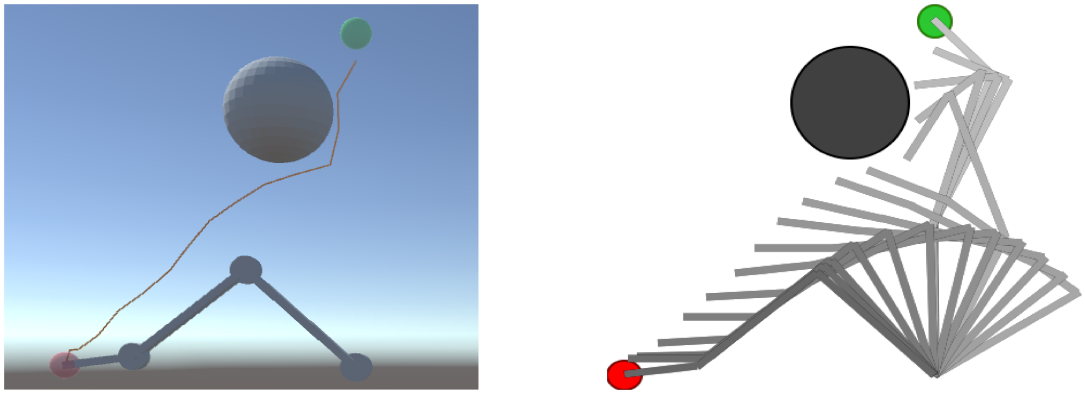


(a) (b)

Fonte: Kazem, Mahdi e Oudah (2008) Fonte: Yeasmin, Shill e Paul (2017)

Na Figura 38 encontra-se à esquerda o trajeto final percorrido pelo manipulador utilizando o algoritmo POM proposto neste trabalho. À direita é retratado o gráfico dos movimentos do manipulador a cada ponto de controle calculado pelo algoritmo.

Figura 38 – Representação do trajeto calculado (a) e gráfico de movimento (b) obtidos pelo uso do POM para o manipulador 3R de Kazem, Mahdi e Oudah (2008) com obstáculo.



(a) (b)

Fonte: Autoria própria

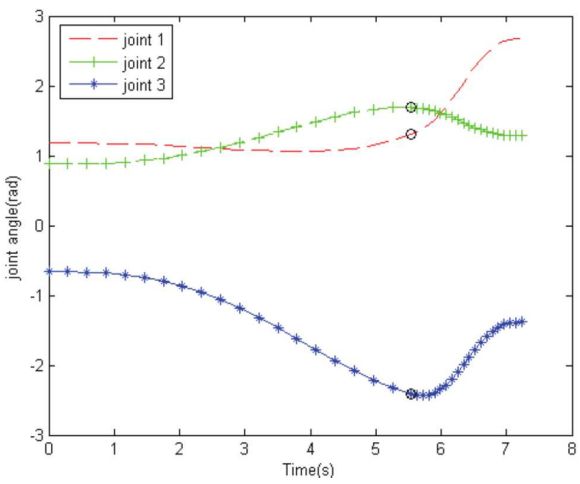
Ao analisar as Figuras 37 e 38, juntamente com os dados comparativos da Tabela 5, pode-se afirmar que novamente o algoritmo POM obteve êxito na minimização do trajeto na presença de um obstáculo estático. Ainda, pelo gráfico à direita da Figura 38, observa-se que mesmo na presença de obstáculos o manipulador consegue realizar movimentos suaves enquanto mantém o trajeto otimizado e livre de colisões.

Tabela 5 – Tabela de resultados para o manipulador 3R em relação ao trajeto e às juntas.

	AG	**EPSO**	**POM**
f_{dist}	$3,42$ m	$3,4702$ m	$3,3208$ m
f_{joint}	$5,78$ rad	$5,0070$ rad	$10,7478$ rad

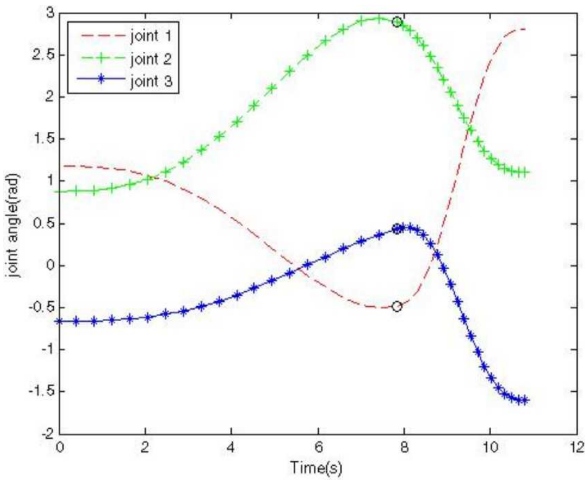
Os gráficos da Figura 39 representam os deslocamentos por junta obtidos por Kazem, Mahdi e Oudah (2008) e Yeasmin, Shill e Paul (2017) respectivamente à esquerda e à direita. Como é perceptível pelos gráficos e os dados da Tabela 5, Kazem, Mahdi e Oudah (2008) possuem maiores valores de deslocamento de junta, porém um trajeto menor que o de Yeasmin, Shill e Paul (2017).

Figura 39 – Gráfico de deslocamento das juntas obtidas usando (a)AG e (b)EPSO do manipulador 3R em ambiente com obstáculos.



(a)

Fonte: Kazem, Mahdi e Oudah (2008)

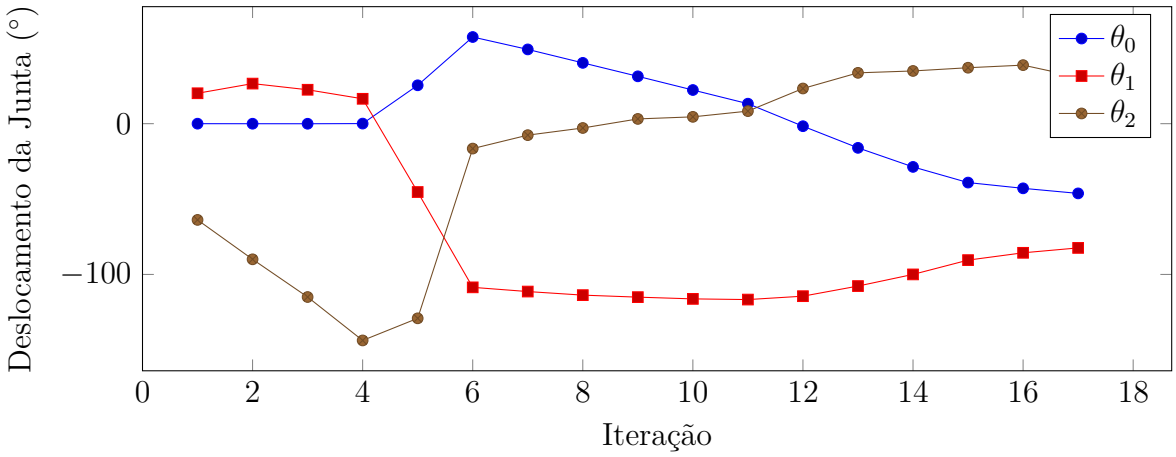


(b)

Fonte: Yeasmin, Shill e Paul (2017)

A Figura 40 retrata os deslocamentos de junta do manipulador para o trajeto com obstáculo. Os valores encontram-se na Tabela 21 do Apêndice D. Apesar de possuir um valor alto de deslocamento de junta em relação aos outros autores, percebe-se que mantém-se a suavidade dos movimentos mesmo com a presença de obstáculo e o cálculo realizado em tempo real.

Figura 40 – Gráfico de deslocamento das juntas do manipulador 3R usando o algoritmo POM em ambiente com obstáculos.



Fonte: Autoria própria

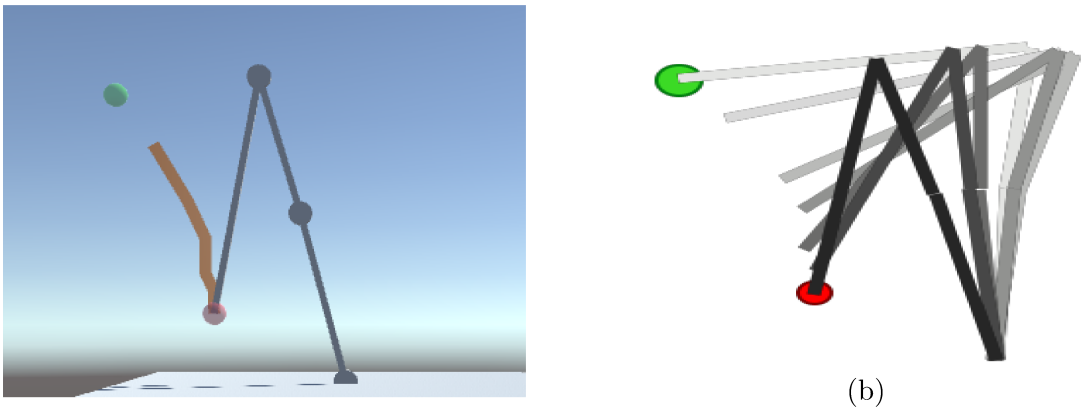
5.1.3 Manipulador 3R em ambiente livre de obstáculos de Sharma, Singh e Singh (2011)

Outro fator importante analisado por Sharma, Singh e Singh (2011) é o gasto de energia. Os autores também utilizam um manipulador com anatomia 3R, com elos de tamanhos $L_0 = 1,53$ m, $L_1 = 1,23$ m e $L_2 = 2,1$ m. O trajeto do manipulador deve ir do ponto $(0\text{ m}, 2.5\text{ m}, 1.9\text{ m})$ ao ponto $(0\text{ m}, 0.6\text{ m}, 1.1\text{ m})$ em um ambiente livre de obstáculos. Os autores utilizam como algoritmo de minimização o AG.

A função de aptidão é calculada pela Equação 5.2, e o trajeto calculado por Sharma, Singh e Singh (2011) possui $2,75$ m, enquanto o trajeto planejado pelo algoritmo POM possui $2,1178$ m. Este é ilustrado à esquerda da Figura 41, e à direita está representado o gráfico dos movimentos realizados pelo manipulador a cada iteração.

$$E = 0.08(3\theta_2 + 2\theta_1 + \theta_0) \tag{5.2}$$

Figura 41 – Representação do trajeto (a) e gráfico de movimento (b) usando POM para o manipulador 3R para otimização de energia usado por Sharma, Singh e Singh (2011) em ambiente livre de obstáculos.

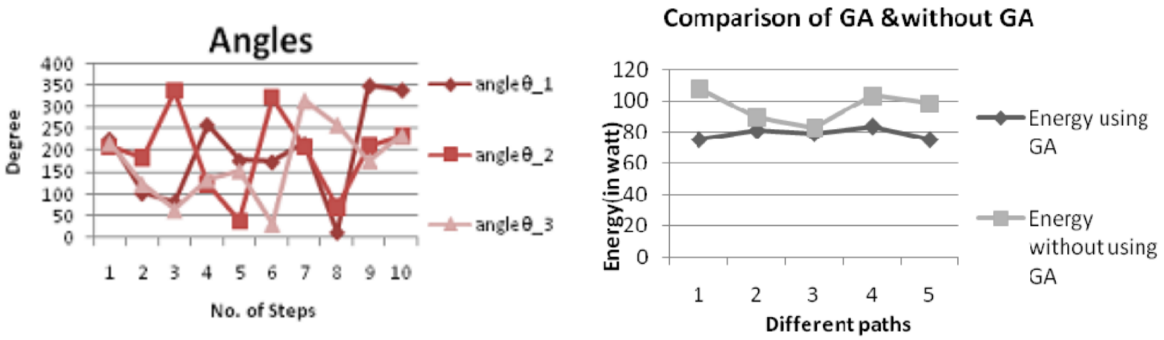


(a) (b)

Fonte: Autoria própria

A Figura 42 mostra à esquerda os ângulos obtidos por Sharma, Singh e Singh (2011) e à direita os valores obtidos de energia em diferentes trajetos. O algoritmo POM aqui proposto possui um deslocamento total de juntas de $90,7755°$ e um consumo de energia calculado com a Equação 5.2 de $15,9852$ W.

Figura 42 – Resultados gráficos apresentados por Sharma, Singh e Singh (2011) para o robô 3R em ambiente livre de obstáculos.

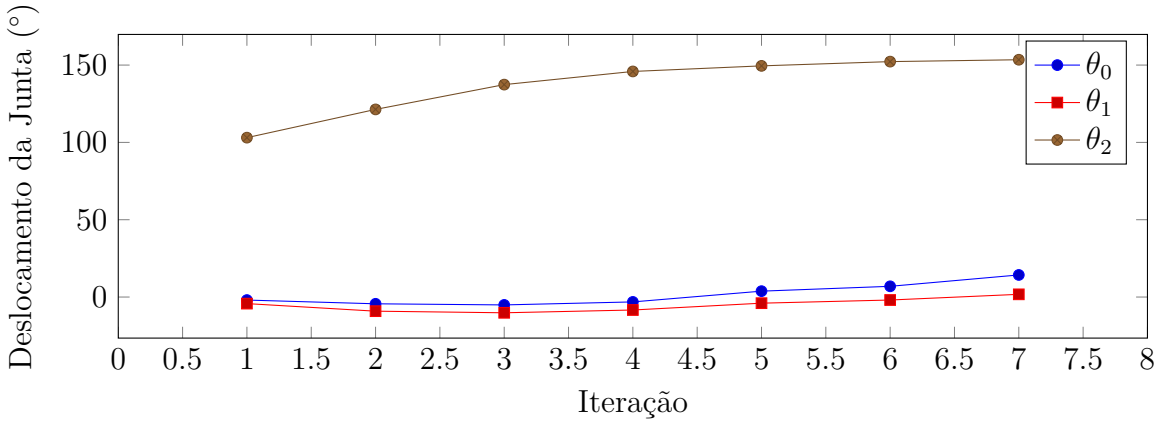


(a) (b)

Fonte: Sharma, Singh e Singh (2011).

Comparado às imagens anteriores, a Figura 43 comprova o movimento suave das juntas do manipulador durante o trajeto percorrido por ele. O gráfico de Sharma, Singh e Singh (2011) retratado à esquerda da Figura 42, possui alterações bruscas de valores de ângulos, que são muito maiores se comparados pelos gráficos.

Figura 43 – Gráfico de deslocamento das juntas do manipulador 3R para otimização de energia usado por Sharma, Singh e Singh (2011) em ambiente livre de obstáculos calculado usando o POM.



Fonte: Autoria própria

Os valores de ângulos deslocados em cada junta e os pontos de controle calculados pelo POM se encontram na Tabela 22 do Apêndice D. Através dos resultados obtidos neste teste, foi comprovada um baixo gasto de energia na utilização do algoritmo POM em relação ao proposto por Sharma, Singh e Singh (2011) com a utilização do AG, apesar de não ser um dos objetivos almejados pelo trabalho.

É importante ressaltar que esta afirmação é válida apenas para este teste, sendo que para os outros manipuladores dos outros testes realizados não se estimou o impacto energético com a utilização do POM, já que esta não é uma proposta deste algoritmo.

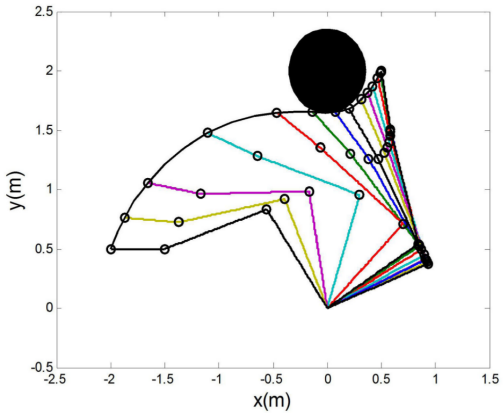
5.1.4 Manipulador 3R em três etapas com obstáculos em diferentes quantidades e posições de Jiang, Yao e Zhou (2018)

Este trabalho objetiva propor um algoritmo que seja minimizador de trajeto garantindo movimentos de junta suaves, capaz de evitar colisões com obstáculos para diferentes anatomias de manipuladores. Por isto, o artigo de Jiang, Yao e Zhou (2018) também será usado como base de comparação.

Jiang, Yao e Zhou (2018) realizam seu experimento em duas etapas. A primeira, é bem semelhante ao segundo experimento de Kazem, Mahdi e Oudah (2008). É um manipulador 3R, com limitações de ângulos das juntas em $-180° \leq \theta \leq 180°$. Seus elos possuem tamanhos $L_0 = 1,0$ m, $L_1 = 1,0$ m e $L_2 = 0,5$ m respectivamente a partir da base.

O ambiente possui um obstáculo estático em formato esférico de raio $r = 0,35$ m, centrado no ponto cartesiano (0 m, 2 m, 0 m). O trajeto deve partir do ponto (0 m, 2 m, 0.5 m) e finalizar no ponto (0 m, 0.5 m, −2 m). O trajeto minimizado apresentado por Jiang, Yao e Zhou (2018) usando AG é representado na Figura 44, bem como a representação dos movimentos realizados pelo manipulador ao longo do trajeto.

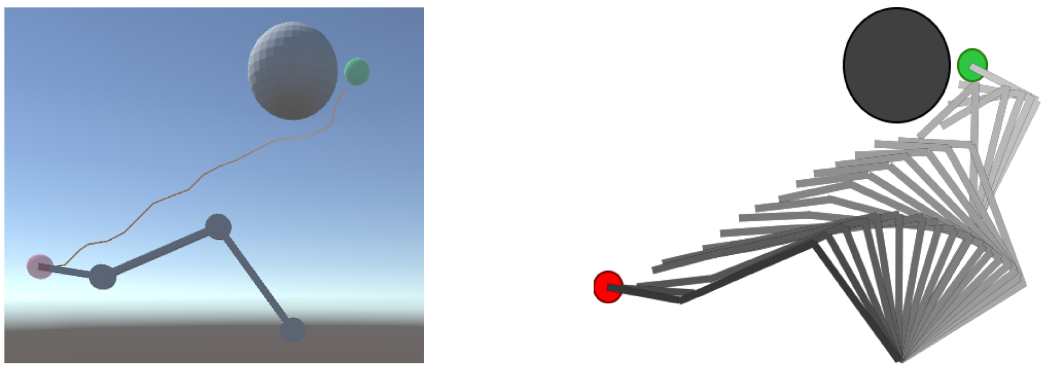
Figura 44 – Gráfico de trajeto e movimento do manipulador 3R com obstáculo de Jiang, Yao e Zhou (2018).



Fonte: Jiang, Yao e Zhou (2018).

Pode-se então comparar o trajeto original do autor com o gerado pelo POM. Este último encontra-se à esquerda da Figura 45 e nota-se sua melhor otimização em relação ao trajeto original de Jiang, Yao e Zhou (2018). À direita, encontra-se o gráfico de movimentos do robô manipulador ao longo do cominho. Os valores de ângulos deslocados a cada movimento a fim de atingir o ponto de controle calculado, encontram-se na Tabela 23, no Apêndice D.

Figura 45 – Representação do trajeto (a) e do gráfico de movimento (b) gerados pelo POM para o manipulador 3R com obstáculo usado por Jiang, Yao e Zhou (2018).



(a) (b)

Fonte: Autoria própria

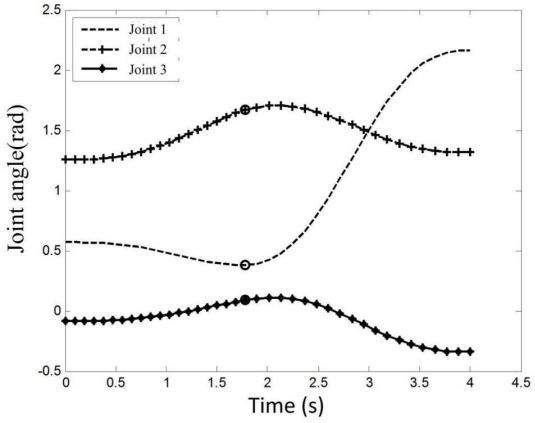
A Tabela 6 comprova o menor tamanho do trajeto calculado pelo POM em relação ao calculado por Jiang, Yao e Zhou (2018) usando o AG. Os deslocamentos de junta são maiores para o algoritmo aqui proposto, contudo a partir das Figuras 44 e 45 é possível notar que ambos trajetos são suaves em relação às juntas.

Tabela 6 – Tabela de resultados para o manipulador 3R de Jiang, Yao e Zhou (2018) com um obstáculo em relação ao trajeto e às juntas.

	AG	**POM**
f_{dist}	$3,1343$ m	$3,0861$ m
f_{joint}	$3,4612$ rad	$8,3173$ rad

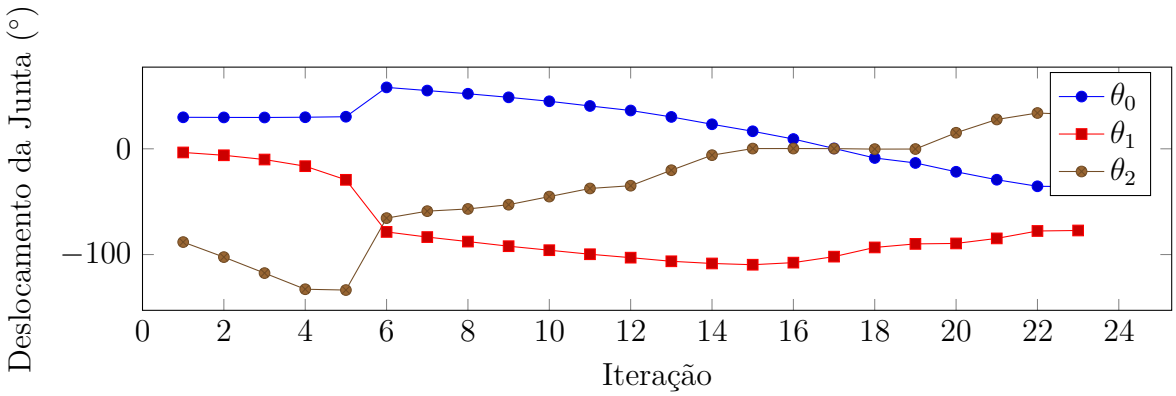
A Figura 46 ilustra o gráfico de deslocamento para cada uma das juntas calculadas por Jiang, Yao e Zhou (2018) e comprova a afirmação realizada anteriormente sobre o trajeto de juntas realizado sem movimentos bruscos. O mesmo pode ser observado para o trajeto planejado pelo POM, cujos movimentos de juntas são retratados no gráfico da Figura 47.

Figura 46 – Gráfico de deslocamento de juntas do manipulador 3R com um obstáculo de Jiang, Yao e Zhou (2018).



Fonte: Jiang, Yao e Zhou (2018).

Figura 47 – Gráfico de deslocamento das juntas calculado usando o POM com o manipulador 3R com obstáculo usado por Jiang, Yao e Zhou (2018).



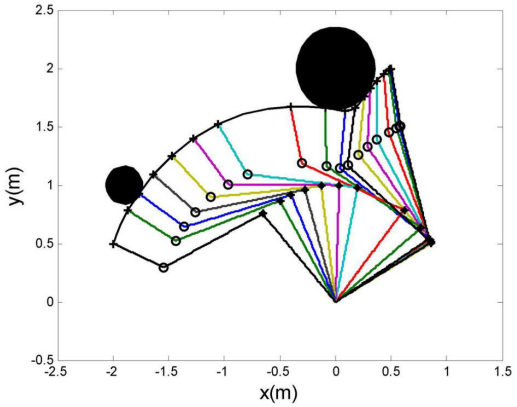
Fonte: Autoria própria

Ao analisar o gráfico da Figura 47 pode-se afirmar que o trajeto das juntas do manipulador para o ambiente com um obstáculo estático é bem moderado. É importante considerar na comparação deste com o original de Jiang, Yao e Zhou (2018) que enquanto o autor calcula todo o trajeto de juntas inicialmente, o POM o planeja em tempo real.

A segunda etapa do experimento de Jiang, Yao e Zhou (2018) consiste em adicionar ao ambiente atual, outro obstáculo. Este também possui formato esférico, com raio $r = 0,16$ m, cujo centro se posiciona no ponto $(0\text{ m}, 1\text{ m}, -1.9\text{ m})$. Assim como na primeira etapa, o manipulador deve partir do ponto $(0\text{ m}, 2\text{ m}, 0.5\text{ m})$ e findar no ponto $(0\text{ m}, 0.5\text{ m}, -2\text{ m})$.

O trajeto calculado por Jiang, Yao e Zhou (2018) como otimizado para este segundo cenário, é ilustrado pela Figura 48. Também é retratado nela, os movimentos realizados pelo manipulador ao longo do trajeto.

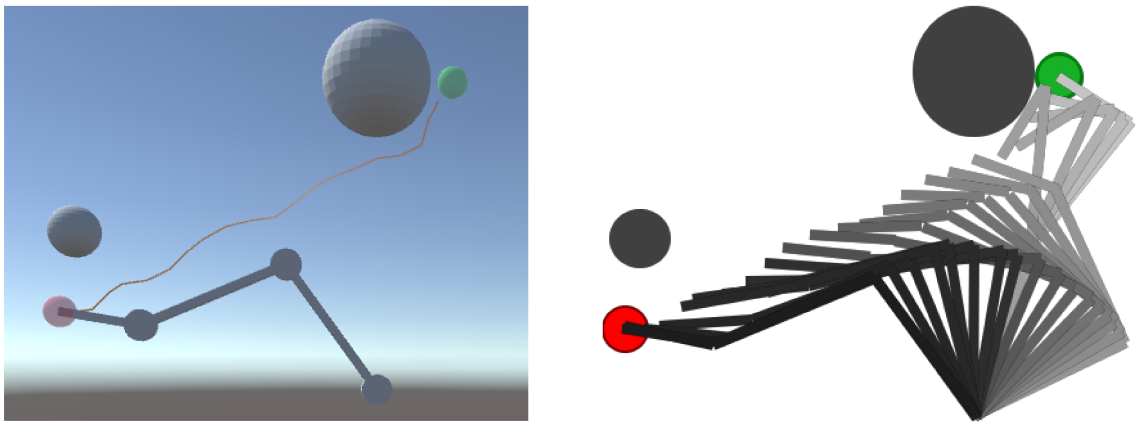
Figura 48 – Gráfico de trajeto e movimento do manipulador 3R com dois obstáculos de Jiang, Yao e Zhou (2018).



Fonte: Jiang, Yao e Zhou (2018).

Em comparação ao trajeto original, a Figura 49 contém à esquerda o trajeto final calculado pelo algoritmo POM. A representação dos movimentos ao longo do caminho, são ilustrados à direita da Figura 49.

Figura 49 – Representação do trajeto (a) e do gráfico de movimento (b) do manipulador 3R com dois obstáculos usado por Jiang, Yao e Zhou (2018) calculados usando o POM.



(a) (b)

Fonte: Autoria própria

Comparando os trajetos de Jiang, Yao e Zhou (2018) entre os dois experimentos, ilustrados nas Figuras 44 e 48, é possível verificar que o segundo obstáculo é inserido a fim de alterar levemente o trajeto calculado na primeira etapa. Confrontando com os trajetos obtidos pelo POM retratados nas Figuras 45 e 49, o obstáculo não altera o caminho calculado na primeira etapa, pois o trajeto não é próximo à localização do segundo obstáculo.

Assim, como nota-se pela análise da Tabela 7 e dos gráficos de movimento das Figuras 45 e 49, o trajeto cartesiano se mantém quase idêntico entre as duas etapas. O trajeto das juntas é visualmente semelhante, apesar de possuir um valor maior de deslocamento.

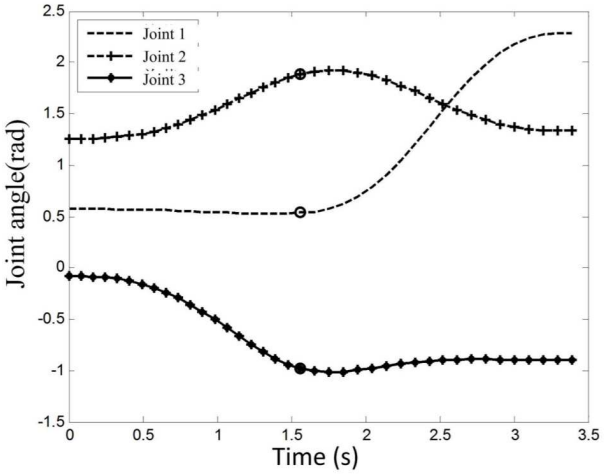
Tabela 7 – Tabela de resultados para o manipulador 3R de Jiang, Yao e Zhou (2018) com dois obstáculos em relação ao trajeto e às juntas.

	AG	**POM**
f_{dist}	$3,1535$ m	$3,0817$ m
f_{joint}	$4,1246$ rad	$10,0465$ rad

O valor inconstante das juntas é esperado, por se tratar de um trajeto em tempo real e que pode ser levemente alterado dependendo das distâncias e localizações dos pontos de controle a cada momento. Os gráficos de deslocamento de juntas comprovam que

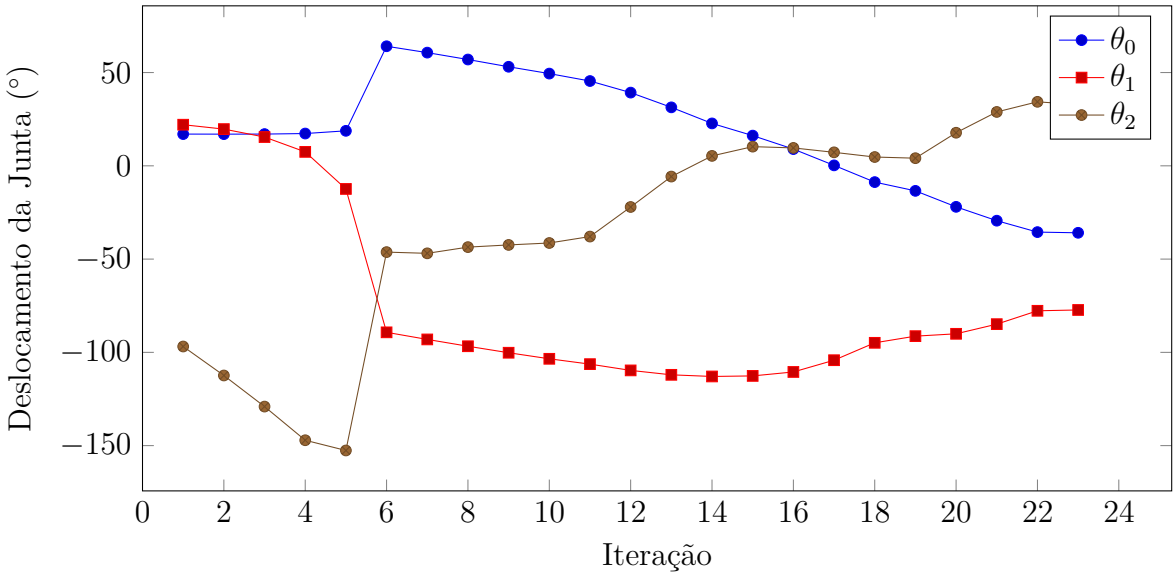
tanto no trabalho original retratado na Figura 50, quanto no trajeto calculado pelo POM ilustrado pela Figura 51, os movimentos de junta são suaves.

Figura 50 – Gráfico de deslocamento de juntas do manipulador 3R com dois obstáculos de Jiang, Yao e Zhou (2018).



Fonte: Jiang, Yao e Zhou (2018).

Figura 51 – Gráfico de deslocamento das juntas do manipulador 3R com dois obstáculos usado por Jiang, Yao e Zhou (2018) calculados usando o POM.



Fonte: Autoria própria

Dados os resultados dos experimentos, pode-se afirmar que o algoritmo POM apresentou bons resultados em ambos, tanto em trajeto cartesiano quanto de juntas. Portanto, os testes realizados a partir dos experimentos de Jiang, Yao e Zhou (2018) comprovam que o algoritmo aqui proposto é eficaz em traçar caminhos otimizados mesmo na presença de mais de um obstáculo.

Assim, comprova-se que para anatomias de manipuladores 3R com alterações quanto aos tamanhos de elos e limitações de junta, o algoritmo POM obteve resultados satisfatórios. Entretanto, é necessário alterar também a quantidade de graus de liberdade do manipulador a fim de analisar se o algoritmo é capaz de se adaptar ao novo modelo sem sofrer complicações. Com este intuito, o trabalho de Tian e Collins (2004) será utilizado a fim de comparar os resultados para análise de eficácia no propósito do algoritmo POM.

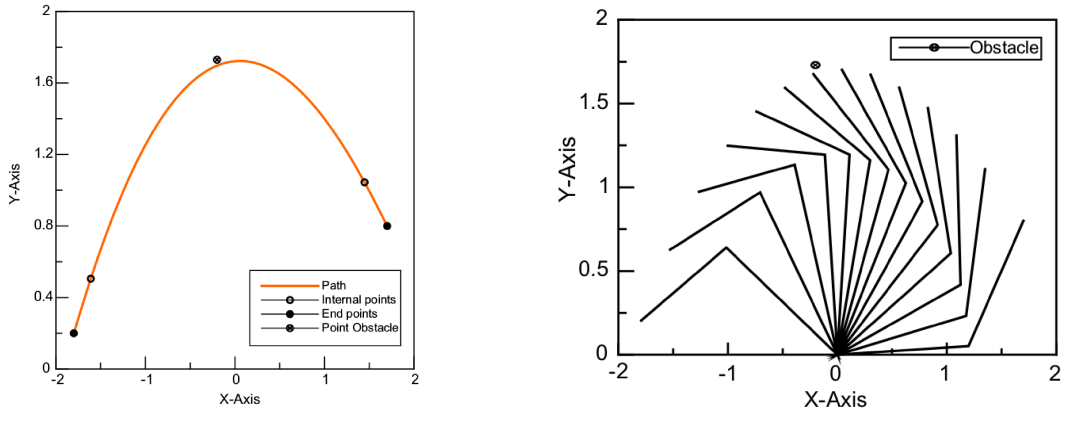
5.1.5 Manipulador 2R com obstáculos em diferentes posições e quantidades variadas de Tian e Collins (2004)

Tian e Collins (2004) dividem suas análises em três experimentos distintos com uma mesma anatomia de manipulador. Trata-se de um robô 2R, cujas juntas possuem limitações de ângulos iguais a $-180° \leq \theta \leq 180°$. Seus elos tem tamanhos iguais a $L_0 = 1,0$ m e $L_1 = 1,0$ m. O trajeto em todos os experimentos inicia no ponto $(0\text{ m}, 0.2\text{ m}, -1.8\text{ m})$ e finaliza no ponto $(0\text{ m}, 0.8\text{ m}, 1.7\text{ m})$.

O primeiro experimento possui um obstáculo no ponto cartesiano $(0\text{ m}, 1.5\text{ m}, -0.2\text{ m})$. O autor não cita a dimensão do obstáculo, que será utilizada neste trabalho com um raio igual a $r = 0,05$ m.

O trajeto obtido por Tian e Collins (2004) é exposto à esquerda na Figura 52. Os dois pontos de controle utilizados pelos autores são $PC1 = (0\text{ m}, 0.51\text{ m}, -1.61\text{ m})$ e $PC2 = (0\text{ m}, 1.04\text{ m}, 1.45\text{ m})$. À direita da Figura 52 é representado graficamente os movimentos do manipulador durante o trajeto resultante do AG.

Figura 52 – Trajeto obtido (a) e gráfico de movimento (b) de Tian e Collins (2004) em seu primeiro experimento com o manipulador 2R.

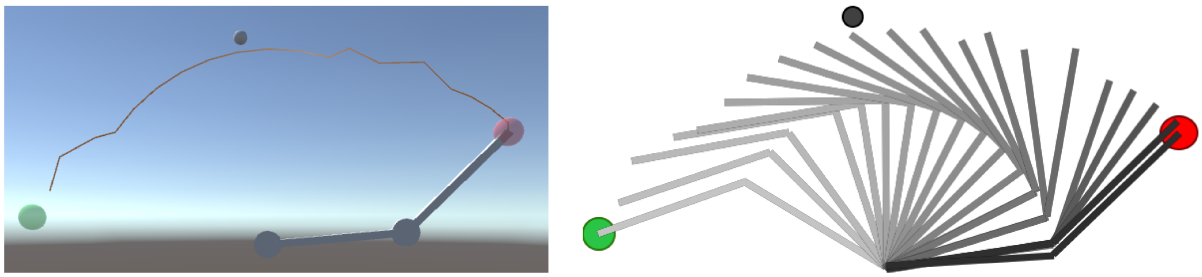


(a) (b)

Fonte: Tian e Collins (2004).

Da mesma maneira, a Figura 53 mostra à esquerda o trajeto calculado pelo POM, e à direita o gráfico de movimentos a cada ponto de controle calculado. Os pontos de controle com seus devidos valores de junta encontram-se descritos na Tabela 25 no Apêndice D. Os que mais se assemelham aos utilizados por Tian e Collins (2004) são o da terceira iteração $P_{i=3} = (0\text{ m}, 0.653\text{ m}, -1.575\text{ m})$ e o da 18ª iteração $P_{i=18} = (0\text{ m}, 1.047\text{ m}, 1.442\text{ m})$.

Figura 53 – Trajeto obtido (a) e gráfico de movimento (b) do primeiro teste com o manipulador 2R com um obstáculo calculados usando o POM.

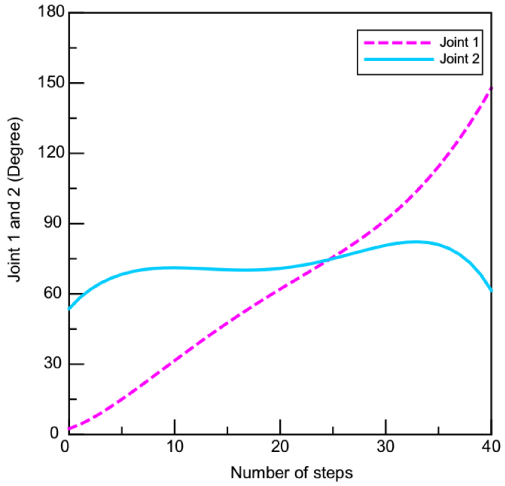


(a) (b)

Fonte: Autoria própria

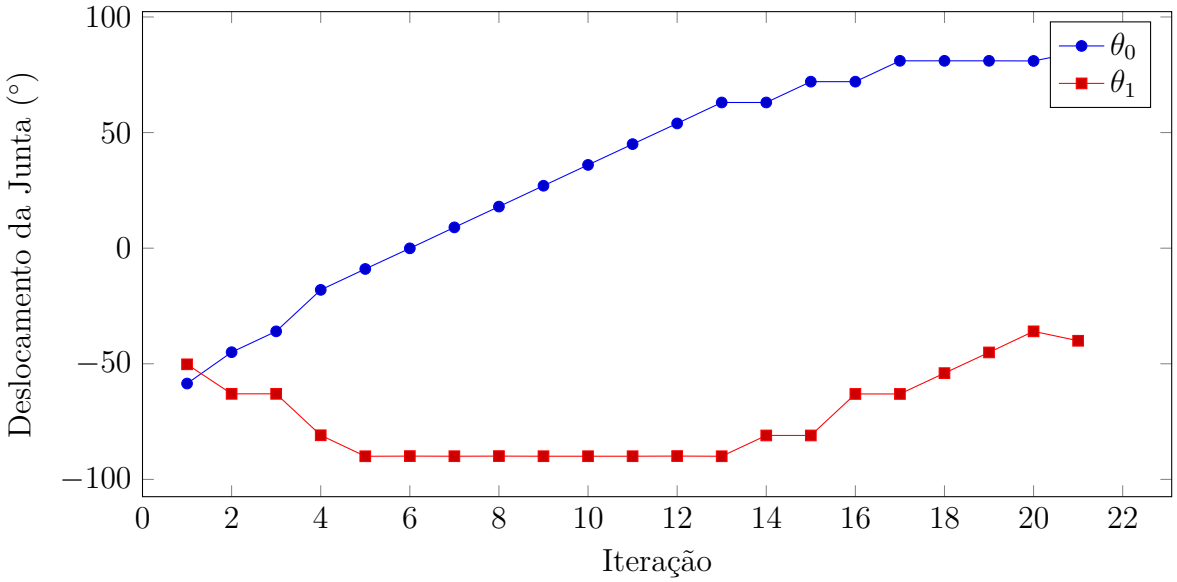
Os deslocamentos de juntas neste primeiro experimento de Tian e Collins (2004) são exibidos no gráfico da Figura 54 e comparados aos obtidos usando o algoritmo POM, os quais são expostos no gráfico da Figura 55. É importante ressaltar que no presente trabalho o zero da junta é contado como a posição em que o manipulador se encontra totalmente esticado para cima. De maneira distinta, Tian e Collins (2004) utiliza como zero de junta o equivalente a $-90°$ do aqui utilizado.

Figura 54 – Gráfico dos deslocamentos de juntas de Tian e Collins (2004) em seu primeiro experimento com o manipulador 2R.



Fonte: Tian e Collins (2004).

Figura 55 – Gráfico de deslocamento das juntas do primeiro teste com o manipulador 2R com um obstáculo obtidos com o POM.



Fonte: Autoria própria

Os autores Tian e Collins (2004) não expõem os valores obtidos do tamanho do caminho finalizado, nem o total deslocamento das juntas. Porém, pela análise dos gráficos pode-se afirmar que os valores das juntas são aproximados. Os valores obtidos pelo algoritmo aqui proposto são exibidos na Tabela 8.

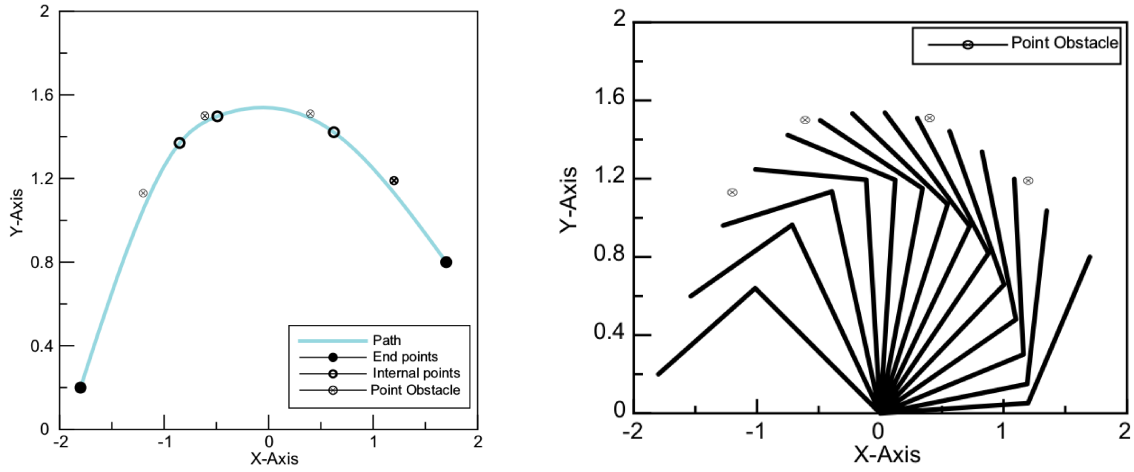
Tabela 8 – Tabela de resultados para o manipulador 2R de Tian e Collins (2004) em seu primeiro experimento.

	POM
f_{dist}	$4,2672$ m
f_{joint}	$241,6989°$

No segundo experimento, novamente o manipulador deve ir do ponto $(0\text{ m}, 0.2\text{ m}, -1.8\text{ m})$ ao ponto $(0\text{ m}, 0.8\text{ m}, 1.7\text{ m})$. No novo ambiente, existem quatro obstáculos de formato esférico, os quais serão utilizados novamente com raio $r = 0,05$ m. Os obstáculos se encontram centrados nos pontos $O_0 = (0\text{ m}, 1.1\text{ m}, -1.2\text{ m})$, $O_1 = (0\text{ m}, 1.5\text{ m}, -0.6\text{ m})$, $O_2 = (0\text{ m}, 1.5\text{ m}, 0.4\text{ m})$ e $O_3 = (0\text{ m}, 1.2\text{ m}, 1.2\text{ m})$.

O trajeto calculado por Tian e Collins (2004) é exibido à esquerda da Figura 56. Os três pontos demarcados como pontos de controle utilizados pelos autores são $PC1 = (0\text{ m}, 1.37\text{ m}, -0.85\text{ m})$, $PC2 = (0\text{ m}, 1.497\text{ m}, -0.49\text{ m})$ e $PC3 = (0\text{ m}, 1.42\text{ m}, 0.62\text{ m})$. À direita da Figura 56 é ilustrado o gráfico de movimentos do manipulador durante o caminho.

Figura 56 – Trajeto obtido (a) e gráfico de movimento (b) de Tian e Collins (2004) em seu segundo experimento com o manipulador 2R.

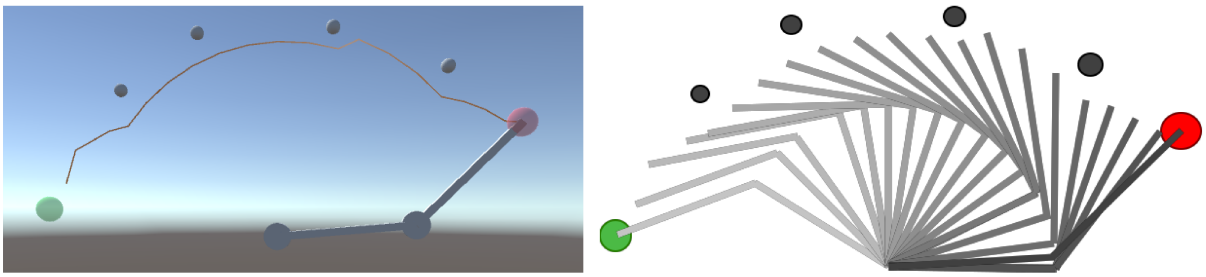


(a) (b)

Fonte: Tian e Collins (2004).

O caminho planejado pelo POM é exibido do lado esquerdo da Figura 57 e à direita, o gráfico de movimentos. Os pontos de controle calculados e os valores de junta para alcançar cada um deles são descritos na Tabela 26 do Apêndice D.

Figura 57 – Trajeto obtido (a) e gráfico de movimento (b) do segundo teste com o manipulador 2R com quatro obstáculos calculados pelo POM.



(a) (b)

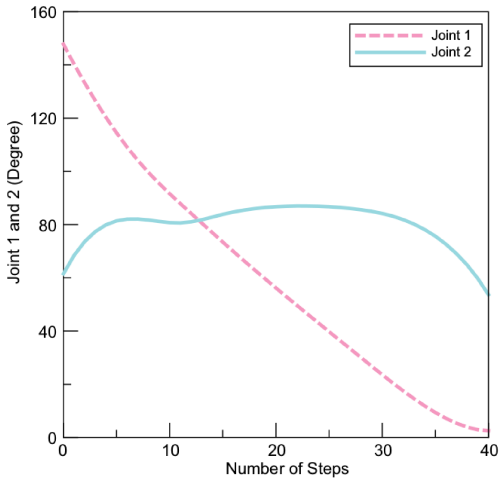
Fonte: Autoria própria

Nos pontos de controle do trajeto calculado pelo algoritmo POM não existem pontos próximos aos utilizados por Tian e Collins (2004). Isto ocorre pois, como pode ser visualmente notado ao comparar as Figuras 56 e 57, o trajeto dos autores originais passa muito próximo aos obstáculos. Isto não ocorre no trajeto do algoritmo proposto neste trabalho, evitando possíveis colisões e diminuindo o tamanho do trajeto.

Em relação aos deslocamentos das juntas, em ambos trabalhos eles são suaves. O gráfico que confirma esta afirmação para o experimento de Tian e Collins (2004) é exibido

na Figura 58. É importante observar que em comparação ao gráfico do experimento anterior (Figura 54), existe uma diferença notória.

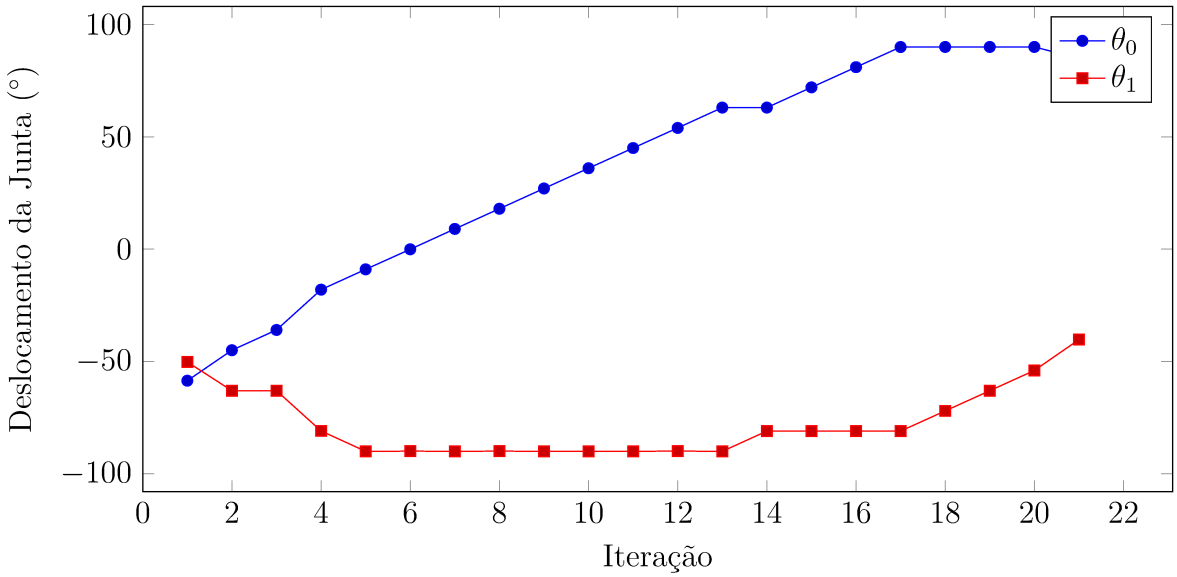
Figura 58 – Gráfico dos deslocamentos de juntas de Tian e Collins (2004) em seu segundo experimento com o manipulador 2R.



Fonte: Tian e Collins (2004).

No caso do trajeto calculado pelo POM, os deslocamentos das juntas durante o caminho são retratados no gráfico da Figura 59. Observa-se que em comparação com o gráfico do experimento anterior (Figura 55), os dois são quase idênticos.

Figura 59 – Gráfico de deslocamento das juntas do segundo experimento com o manipulador 2R com quatro obstáculos calculados usando o POM.



Fonte: Autoria própria

Conforme supramencionado, Tian e Collins (2004) não apresentam os valores do

tamanho do trajeto e dos ângulos deslocados da junta. Estes valores para o algoritmo aqui proposto encontram-se na Tabela 59.

Tabela 9 – Tabela de resultados para o manipulador 2R de Tian e Collins (2004) em seu segundo experimento.

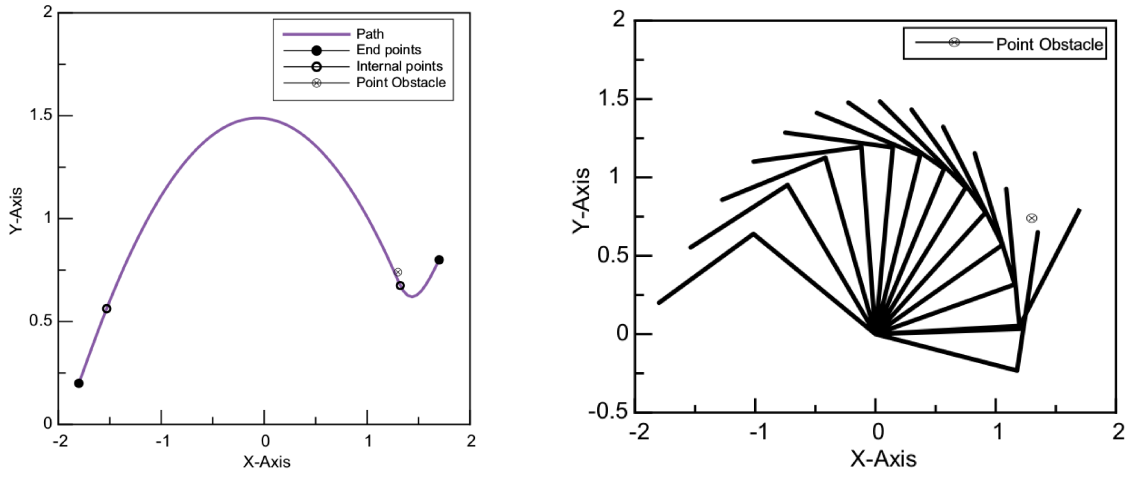
	POM
f_{dist}	$4,2127$ m
f_{joint}	$243,4745°$

A partir da análise comparativa realizada visualmente entre as Figuras 56 e 57, e também entre as Figuras 58 e 59, pode-se afirmar que o algoritmo POM realizou uma melhor minimização do trajeto. É possível ainda reiterar que o trajeto realizado com o algoritmo aqui proposto tem menores chances de colisão com os obstáculos no ambiente.

Por fim, o terceiro experimento realizado por Tian e Collins (2004) com o objetivo de partir do ponto $(0 \text{ m}, 0.2 \text{ m}, -1.8 \text{ m})$ e chegar ao ponto $(0 \text{ m}, 0.8 \text{ m}, 1.7 \text{ m})$, possui um ambiente com apenas um obstáculo localizado próximo ao ponto de destino. Decidiu-se mais uma vez por utilizar um raio de valor $r = 0,05$ m em formato esférico. Este se encontra centrado na coordenada cartesiana $O = (0 \text{ m}, 0.7 \text{ m}, 1.3 \text{ m})$.

Ao realizar o planejamento do trajeto, Tian e Collins (2004) encontram o caminho ilustrado à esquerda da Figura 60. Para isto são utilizados dois pontos de controle, conforme representado graficamente, porém os autores não mencionam os pontos cartesianos referentes a eles.

Figura 60 – Trajeto obtido (a) e gráfico de movimento (b) de Tian e Collins (2004) em seu terceiro experimento com o manipulador 2R.



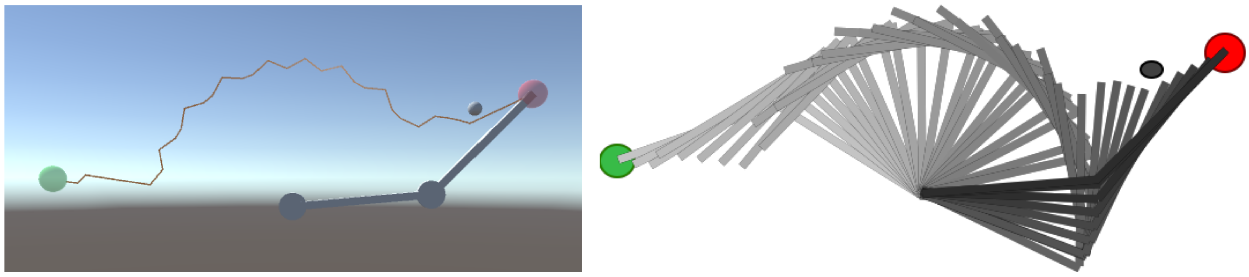
(a) (b)

Fonte: Tian e Collins (2004).

À direita da Figura 60 é apresentado o gráfico dos movimentos realizados ao longo

do caminho. O trajeto e o gráfico de movimentos obtidos através da utilização do POM, encontram-se respectivamente à esquerda e à direita da Figura 61.

Figura 61 – Trajeto obtido (a) e gráfico de movimento (b) do terceiro teste com o manipulador 2R com um obstáculo próximo ao ponto de destino calculados pelo POM.

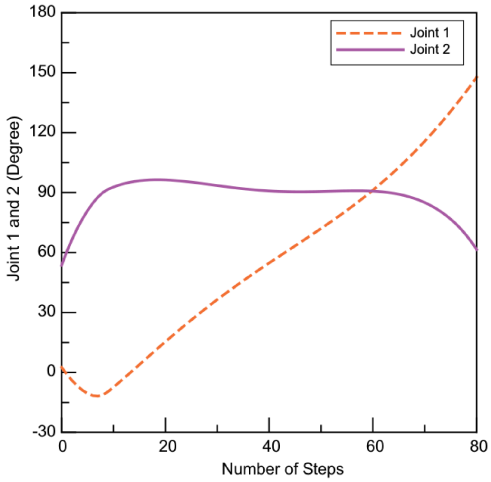


(a) (b)

Fonte: Autoria própria

Com base na comparação entre o trajeto original da Figura 60 e o aqui proposto (Figura 61), nota-se que o trajeto calculado pelo POM é menor. O gráfico relativo aos deslocamentos das juntas utilizadas por Tian e Collins (2004) é exibido na Figura 62.

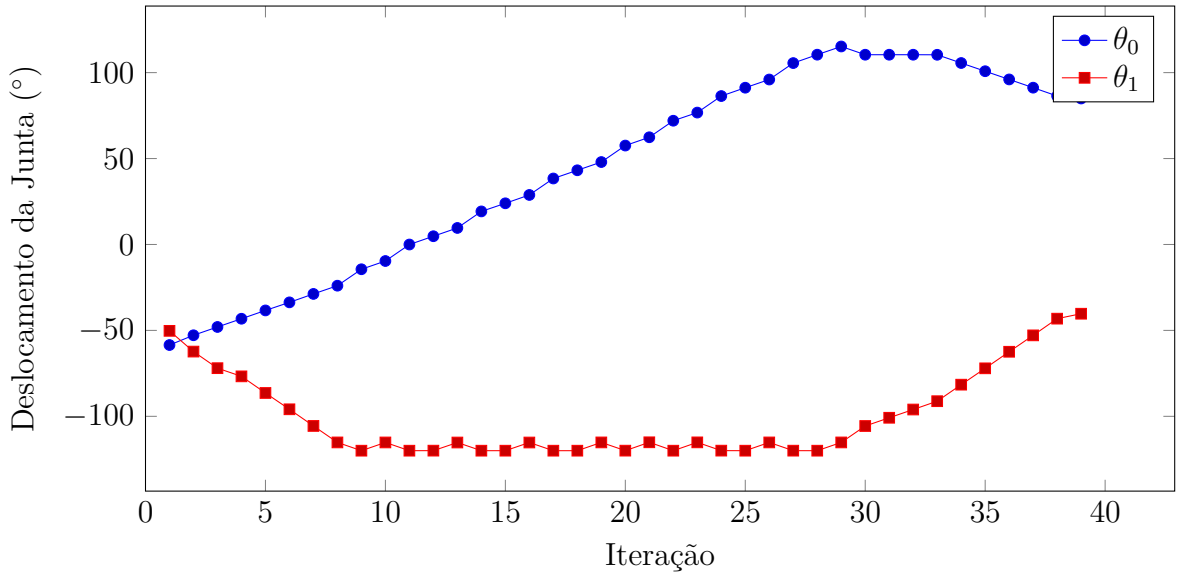
Figura 62 – Gráfico dos deslocamentos de juntas de Tian e Collins (2004) em seu terceiro experimento com o manipulador 2R.



Fonte: Tian e Collins (2004).

O gráfico do deslocamento das juntas advindo do trajeto calculado pelo algoritmo POM é exposto na Figura 63. Observa-se que este possui valores de junta um pouco maiores que os experimentos anteriores, porém mantém a suavidade no trajeto das juntas.

Figura 63 – Gráfico de deslocamento das juntas do terceiro experimento com o manipulador 2R com obstáculo próximo ao ponto de chegada calculadas pelo POM.



Fonte: Autoria própria

Os valores do tamanho do trajeto e dos deslocamentos totais das juntas obtidos com o algoritmo aqui proposto são exibidos na Tabela 10. Os valores dos pontos de controle e das juntas para alcançá-los a cada iteração, encontram-se na Tabela 27 do Apêndice D.

Tabela 10 – Tabela de resultados para o manipulador 2R de Tian e Collins (2004) em seu terceiro experimento.

	POM
f_{dist}	$4,4358$ m
f_{joint}	$420,7519°$

Portanto, pode-se afirmar que o algoritmo POM proposto neste trabalho encontra trajetos otimizados para diferentes anatomias de manipuladores. Também foi comprovado que ele se comporta bem em ambientes com a presença de obstáculos estáticos.

5.2 Ambientes com obstáculos dinâmicos

Outro objetivo proposto neste trabalho é o cálculo do caminho minimizado em ambientes com obstáculos dinâmicos, ou seja, que podem ser inseridos ou retirados a qualquer momento entre as iterações do caminho. Foram realizados dois experimentos a fim de constatar se o algoritmo é eficaz no replanejamento do caminho evitando colidir com o obstáculo.

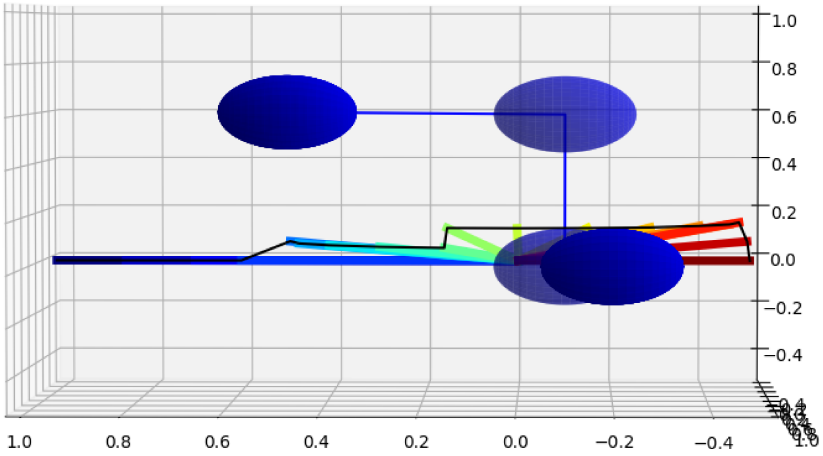
O primeiro experimento utiliza um manipulador com anatomia Torção-Rotação-Deslocamento, cujos limites estabelecidos são exibidos na Tabela 11. O caminho deve partir do ponto $(0\text{ m}, 0\text{ m}, 1\text{ m})$ e chegar até o ponto $(0\text{ m}, 0\text{ m}, -0.5\text{ m})$.

Tabela 11 – Tabela dos limites utilizados para a anatomia TRD em ambiente dinâmico.

Tipo de Junta	**Limites desejados**
T_0	$0° \leq \alpha_0 \leq 180°$
R_0	$0° \leq \theta_0 \leq 90°$
d_0	$0,5\text{ m} \leq d_0 \leq 1,0\text{ m}$

O ambiente possui um obstáculo em formato esférico de raio $r = 0,15$ m, o qual desloca-se pelo ambiente durante o trajeto do manipulador. A Figura 64 ilustra graficamente o trajeto tanto do obstáculo quanto do manipulador. Nela, o obstáculo é representado pelas esferas azuis e o caminho realizado pelo manipulador representado pela linha preta.

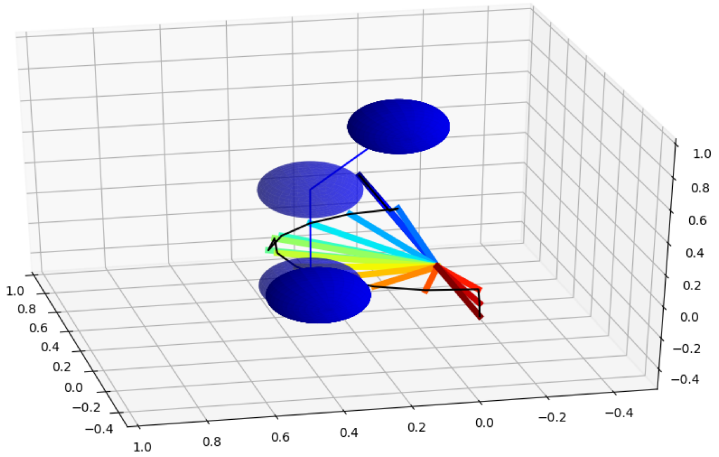
Figura 64 – Gráfico do caminho planejado pelo POM evidenciando o desvio do obstáculo pelo manipulador TRD.



Fonte: Autoria própria

A Figura 64 evidencia o desvio realizado pelo manipulador ao detectar o obstáculo. Os movimentos realizados pelo manipulador são representados pelas linhas grossas coloridas, onde o azul escuro é o ponto de partida, seguindo a escala cromática até chegar ao destino na cor vermelho escuro. Estes movimentos são melhor representados no gráfico da Figura 65.

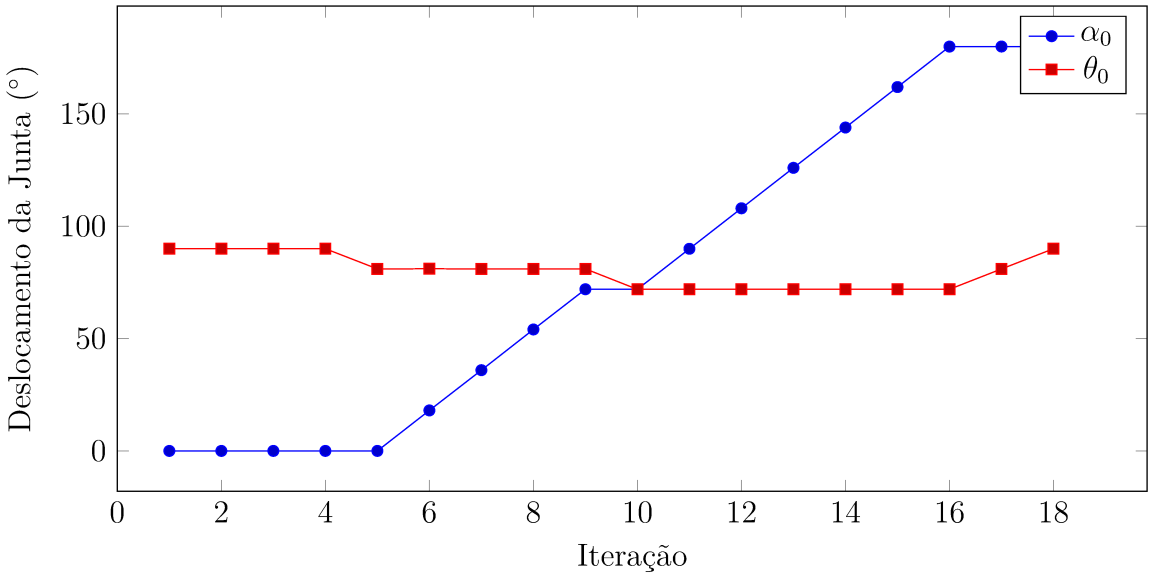
Figura 65 – Gráfico do caminho planejado pelo POM com os movimentos realizados pelo manipulador TRD.



Fonte: Autoria própria

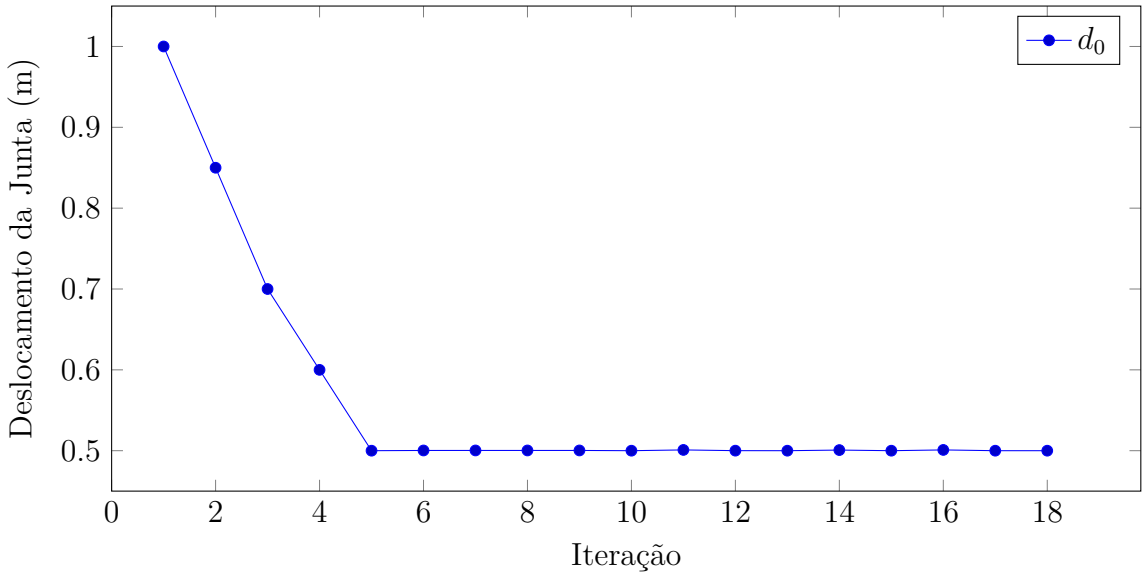
Pode-se afirmar pelas Figuras 64 e 65 que mesmo em ambiente dinâmico, o algoritmo aqui proposto é capaz de manter a suavidade dos movimentos tanto no espaço cartesiano quanto no espaço das juntas. Isto é confirmado pelos gráficos de deslocamento das juntas das Figuras 66 e 67, onde a Figura 66 ilustra os deslocamentos em graus das juntas de torção e rotação do manipulador. A Figura 67 retrata o deslocamento da junta prismática ao longo do caminho.

Figura 66 – Gráfico de deslocamento das juntas do experimento com o manipulador TRD em ambiente dinâmico calculados pelo POM.



Fonte: Autoria própria

Figura 67 – Gráfico de deslocamento da junta prismática no experimento com o manipulador TRD em ambiente dinâmico calculados pelo POM.



Fonte: Autoria própria

Os valores de deslocamentos das juntas utilizados para a reprodução dos gráficos são exibidos na Tabela 28 no Apêndice D. Nela, também encontram-se os pontos de controle calculados pelo algoritmo POM a cada iteração. A partir deles, obtém-se o tamanho do trajeto e o valor total deslocado pelas juntas, os quais são exibidos na Tabela 12.

Tabela 12 – Tabela de resultados para o manipulador TRD em ambiente dinâmico.

	POM
f_{dist}	$2,2795$ m
f_{joint}	$216,9424°$

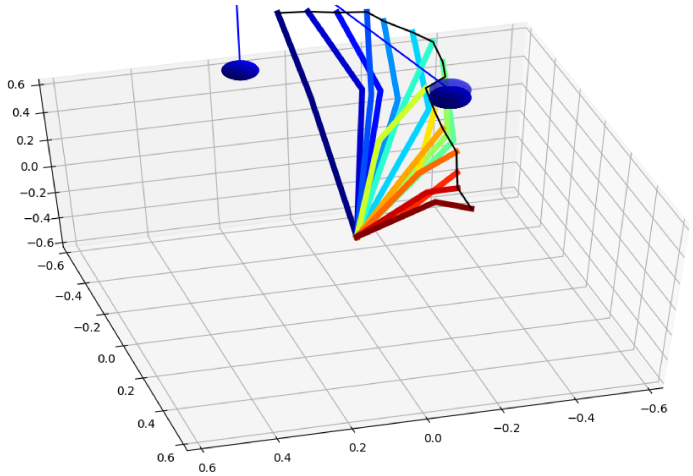
A fim de certificar a eficácia do algoritmo POM em todos os objetivos desejados, será realizado um último experimento. Neste experimento, o manipulador também encontra-se em ambiente com obstáculos dinâmicos e possui uma anatomia mais complexa que as dos manipuladores utilizados anteriormente, a qual tem seus dados descritos na Tabela 13.

Tabela 13 – Valores utilizados na anatomia TRLTRL do manipulador para o segundo experimento em ambiente dinâmico.

Tipo de Junta	**Valores utilizados**
T_0	$-90° \leq \alpha_0 \leq 90°$
R_0	$-45° \leq \theta_0 \leq 45°$
L_1	$0,7$ m
T_1	$-60° \leq \alpha_0 \leq 60°$
R_1	$-30° \leq \theta_0 \leq 30°$
L_2	$0,4$ m

O obstáculo que irá se deslocar pelo ambiente durante o caminho do manipulador possui formato esférico e raio de tamanho $r = 0,05$ m. O trajeto planejado pelo POM para ir do ponto $(-0.8\text{ m}, 0.5\text{ m}, 0\text{ m})$ ao ponto $(0.6\text{ m}, 0.9\text{ m}, -0.1\text{ m})$ é representado na Figura 68 pela linha preta. Esta figura evidencia os pontos replanejados pelo algoritmo para contornar o obstáculo assim que o mesmo foi detectado.

Figura 68 – Gráfico do caminho planejado pelo POM evidenciando o desvio do obstáculo pelo manipulador TRLTRL.

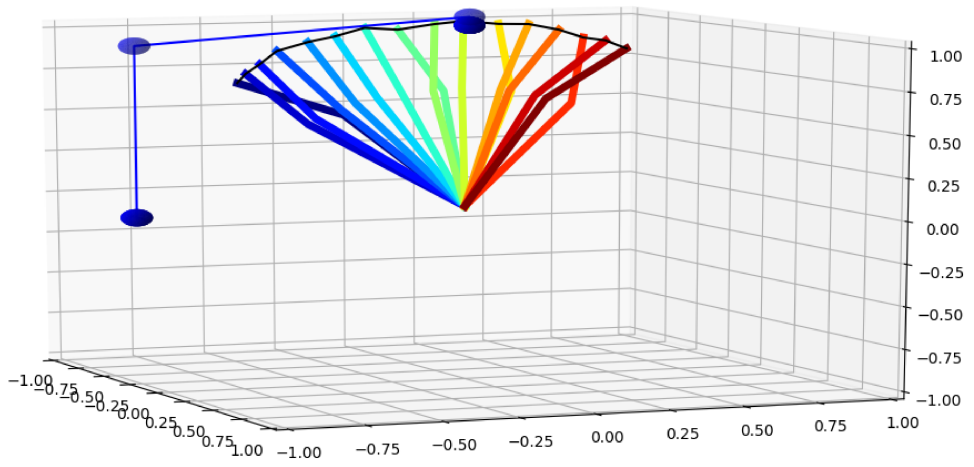


Fonte: Autoria própria

Assim como na representação do experimento anterior, as esferas azuis do gráfico simbolizam os obstáculos e as linhas grossas coloridas correspondem aos movimentos realizados pelo manipulador ao longo do trajeto. Os movimentos seguem a escala cromática, sendo o azul escuro o movimento para alcançar o ponto inicial e o vermelho escuro o movimento realizado para atingir o ponto de destino.

Estes movimentos são melhor identificáveis no gráfico da Figura 69. A partir dele podemos afirmar que mesmo em uma anatomia mais complexa e com obstáculos dinâmicos, o manipulador consegue ser eficaz no planejamento do trajeto sem movimentos bruscos tanto no espaço cartesiano quanto de juntas, evitando possíveis colisões.

Figura 69 – Gráfico do caminho planejado pelo POM com os movimentos realizados pelo manipulador TRLTRL.

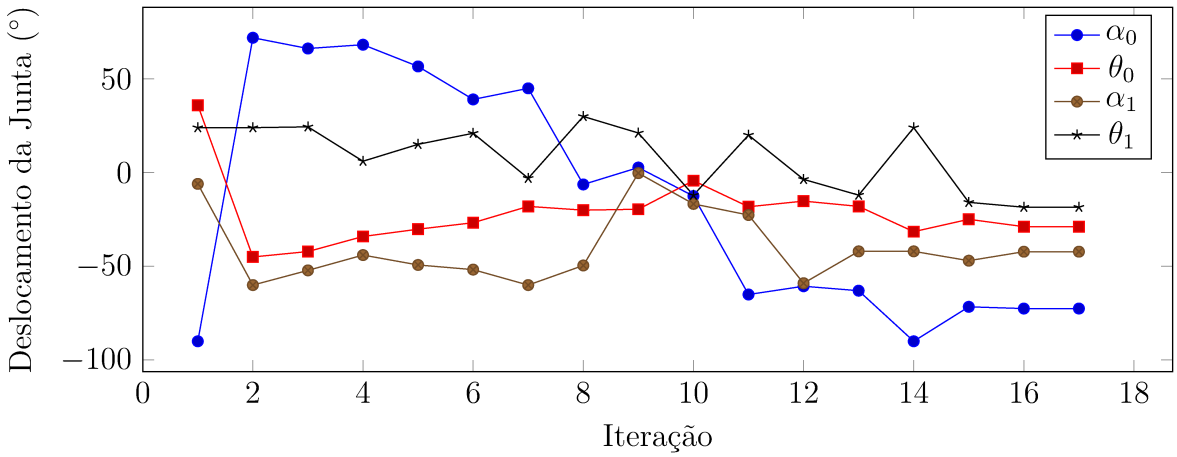


Fonte: Autoria própria

É importante ressaltar que a junta de torção pode acarretar singularidades advindas do fato de que existem múltiplas soluções para um mesmo ponto. Isto pode levar o robô a ficar preso em um mesmo movimento ou realizar movimentos bruscos e desnecessários. A anatomia utilizada neste experimento possui duas juntas de torção, aumentando a possibilidade de ocorrer este tipo de singularidade.

Entretanto, com base nos movimentos retratados na Figura 69 e no gráfico da Figura 70 que representa os deslocamentos realizados para cada junta a cada movimento, nota-se que o manipulador consegue contornar bem este problema. Assim, pode-se afirmar que mesmo para anatomias mais complexas, o algoritmo possui resultados satisfatórios.

Figura 70 – Gráfico de deslocamento das juntas do experimento com o manipulador TRLTRL em ambiente dinâmico usando o algoritmo POM.



Fonte: Autoria própria

Pelo gráfico da Figura 70 observa-se que mesmo não existindo movimentos bruscos das juntas entre os pontos de controle, ele ainda possui um alto deslocamento total. Isto pode ser atestado pelo valor exibido para o deslocamento de juntas na Tabela 14. Apesar disto, é evidente a eficácia do algoritmo POM na minimização do trajeto, pelo tamanho total do caminho dado na Tabela 14 e pela análise do mesmo na Figura 69.

Tabela 14 – Tabela de resultados para o manipulador TRLTRL em ambiente dinâmico.

	POM
f_{dist}	$1,8588$ m
f_{joint}	$1062,3103°$

Após a realização dos experimentos abordados neste capítulo, é importante recordar que o presente trabalho propõe um algoritmo para planejamento de caminho para manipuladores robóticos com diferentes anatomias. Além de calcular um caminho minimizado em ambientes 3D, o movimento realizado pelas juntas deve ser suave. É objetivado ainda que o mesmo consiga planejar o trajeto tanto para ambientes estáticos, quanto replanejá-lo em ambientes dinâmicos.

Fundamentado nos resultados obtidos com a utilização deste algoritmo para os experimentos supradescritos, é notória a capacidade do algoritmo POM em minimizar os caminhos mesmo na presença de obstáculos, sejam eles estáticos ou dinâmicos. É importante ressaltar que esta minimização ocorre em detrimento dos valores totais de deslocamento das juntas.

Ainda que os movimentos no espaço de junta sejam suaves entre dois pontos, não é possível garantir sua minimização no trajeto total, já que o algoritmo gera as coordenadas tanto cartesianas quanto de junta em tempo real. Portanto, é priorizado o trajeto, apesar de haver um esforço do algoritmo em suavizar o movimento entre dois pontos de controle.

6 | Conclusão

A inserção da robótica em diversos setores, principalmente no setor industrial, já é uma necessidade conhecida. Porém, nos últimos anos o investimento em aprimorar os robôs tem aumentado e o cenário apresentado em 2019 na conferência da Federação Internacional de Robótica é que se faz necessário inserir neles alguma forma de inteligência, principalmente relacionada ao conhecimento do ambiente. Isto aumentaria a segurança no processo juntamente com a diminuição do tempo dispendido na realização de tarefas padronizadas.

Por este motivo, o presente trabalho propôs a criação de um algoritmo que deve ser capaz de, dada uma anatomia de manipulador robótico, suas limitações e os pontos de início e destino do trajeto desejado, planejar um caminho de tamanho mínimo, sem movimentos bruscos das juntas e desviando de possíveis obstáculos no ambiente, sejam eles estáticos ou dinâmicos. Para tal, inicialmente decidiu-se analisar a eficácia em minimizar o erro de posição do efetuador.

Com base na revisão realizada na literatura, optou-se pela utilização do algoritmo genético. Porém, observou-se que conforme aumentava-se a quantidade de juntas, o algoritmo era finalizado com erros maiores que 10^{-1}. Devido à ineficácia deste, optou-se por utilizar o algoritmo multiobjetivo NSGA-II com o intuito de minimizar o trajeto com deslocamentos de junta menores. Contudo, o erro de posição novamente encontrou-se fora dos padrões desejados para algumas anatomias.

A partir destas tentativas, percebeu-se que os parâmetros de otimização próprios dos algoritmos variavam conforme a anatomia do manipulador. Isto poderia ser corrigido pela implementação de otimização automática dos parâmetros do algoritmo utilizado. Entretanto, esta opção poderia causar um efeito sistemático no erro e por este motivo não foi realizada.

Por conseguinte, optou-se pela utilização de um algoritmo que não fosse dependente de parâmetros voláteis do otimizador. Neste sentido, baseado na eficácia do algoritmo D* Lite em trajetos calculados em jogos e outras aplicações abordadas no estado da arte, optou-se por utilizá-lo na otimização do trajeto para os manipuladores robóticos.

Foi necessário realizar algumas modificações no algoritmo original do D* Lite a fim

de melhorar sua percepção dos obstáculos, evitar colisões da estrutura do manipulador com os obstáculos, bem como realizar a manipulação dos nós para ambientes dinâmicos. O algoritmo modificado foi eficaz na minimização do trajeto, porém não era possível garantir um movimento suave por parte das juntas.

Assim, optou-se pela inserção do método CCD para encontrar um deslocamento mais suave das juntas entre dois pontos de controle dados pelo D* Lite modificado. Foi necessário combinar dois tipos de algoritmo do CCD, sendo que um trata as juntas progressivamente e o outro em sentido contrário.

Portanto, com base nos resultados apresentados, pode-se afirmar que o algoritmo POM aqui proposto minimiza com eficácia os trajetos, mantendo um erro de posição menor que 10^{-3} e possuindo deslocamentos de junta suaves ao seu decorrer. Os resultados comprovam ainda sua habilidade de evitar obstáculos tanto em ambientes estáticos quanto dinâmicos em diferentes modelos de anatomia dos manipuladores, cumprindo os objetivos almejados.

Apesar disto, o algoritmo POM possui algumas limitações, as quais podem ser abordadas em trabalhos futuros. Uma delas é a maneira de definição dos vizinhos de cada nó, a qual torna o mapeamento mais lento principalmente em anatomias com maiores quantidades de juntas. Outra limitação do algoritmo é o deslocamento total das juntas que pode ser maior que o desejado, já que este algoritmo não analisa o trajeto total para definir este deslocamento, mas apenas o caminho entre o ponto de controle no qual o efetuador se encontra e o próximo ponto.

O alto valor deslocamento das juntas pode acarretar desgaste das juntas e um maior gasto energético. Estes fatores podem ser futuramente avaliados no custo de movimentação a fim de garantir uma maior robustez no algoritmo. Além disto, pode-se ainda inserir alguns fatores como velocidade e aceleração, realizando o planejamento de uma trajetória ao invés de um caminho.

Referências

AL-MUTIB, K. et al. D* lite based real-time multi-agent path planning in dynamic environments. In: IEEE. *2011 Third International Conference on Computational Intelligence, Modelling & Simulation*. 2011. p. 170–174. Disponível em: <https://doi.org/10.1109/CIMSim.2011.38>.

ALGFOOR, Z. A.; SUNAR, M. S.; KOLIVAND, H. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, Hindawi, v. 2015, 2015. Disponível em: <https://doi.org/10.1155/2015/736138>.

ALMEIDA, D. V. de. *Projeto, controle e análise de um manipulador robótico modular*. 237 p. — Universidade de Mogi das Cruzes, Mogi das Cruzes, 2016.

ARISTIDOU, A.; LASENBY, J. Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver. University of Cambridge, Department of Engineering, 2009.

ASGARI, A. et al. Fuzzy optimization of queue length and makespan in job shop scheduling problem by nsga-ii algorithm. 2013.

BANGA, V.; SINGH, Y.; KUMAR, R. Simulation of robotic arm using genetic algorithm & ahp. *World Academy of Science, Engineering and Technology*, v. 25, n. 1, p. 95–101, 2007.

BARAC, D. B. Design specification ab mail robot. 2010.

BARAKAT, A. N.; GOUDA, K. A.; BOZED, K. A. Kinematics analysis and simulation of a robotic arm using matlab. In: IEEE. *2016 4th International Conference on Control Engineering & Information Technology (CEIT)*. 2016. p. 1–5. Disponível em: <https://doi.org/10.1109/CEIT.2016.7929032>.

BATISTA, L. S. et al. A comparison of dominance criteria in many-objective optimization problems. In: IEEE. *2011 IEEE Congress of Evolutionary Computation (CEC)*. 2011. p. 2359–2366. Disponível em: <https://doi.org/10.1109/CEC.2011.5949909>.

BEKELE, E. G.; NICKLOW, J. W. Multi-objective automatic calibration of swat using nsga-ii. *Journal of Hydrology*, Elsevier, v. 341, n. 3-4, p. 165–176, 2007. Disponível em: <https://doi.org/10.1016/j.jhydrol.2007.05.014>.

BESSONNET, G.; LALLEMAND, J.-P. Optimal trajectories of robot arms minimizing constrained actuators and travelling time. In: IEEE. *Proceedings., IEEE International Conference on Robotics and Automation*. [S.l.], 1990. p. 112–117.

BIANCO, C. G. L.; PIAZZI, A. A genetic/interval approach to optimal trajectory planning of industrial robots under torque constraints. In: IEEE. *1999 European Control Conference (ECC)*. 1999. p. 942–947. Disponível em: <https://doi.org/10.23919/ECC.1999.7099428>.

BISWAS, S. S.; ALAM, B.; DOJA, M. N. Generalization of dijkstra's algorithm for extraction of shortest paths in directed multigraphs. *JCS*, v. 9, n. 3, p. 377–382, 2013. Disponível em: <https://doi.org/10.3844/jcssp.2013.377.382>.

BOOMSMA, W.; HAMELRYCK, T. Full cyclic coordinate descent: Solving the protein loop closure problem in cα space. *BMC bioinformatics*, Springer, v. 6, n. 1, p. 159, 2005. Disponível em: <https://doi.org/10.1186/1471-2105-6-159>.

CHEN, G. Analytic closed-form solutions for suboptimal trajectory planning of single-link flexible-joint robot arms. *IEEE transactions on robotics and automation*, IEEE, v. 8, n. 5, p. 658–662, 1992. Disponível em: <https://doi.org/10.1109/70.163790>.

CHRIS, W. *Inverse kinematics and geometric constraints for articulated figure manipulation*. Tese (Doutorado) — M. Sc. thesis (Computing Science), Simon Fraser University, 1993.

CHU, X.; HU, Q.; ZHANG, J. Path planning and collision avoidance for a multi-arm space maneuverable robot. *IEEE Transactions on Aerospace and Electronic Systems*, IEEE, v. 54, n. 1, p. 217–232, 2017. Disponível em: <https://doi.org/10.1109/TAES.2017.2747938>.

COLIN, E. C. *Pesquisa Operacional: 170 aplicações em estratégia, finanças, logística, produção, marketing e vendas*. [S.l.]: Livros Técnicos e Científicos, 2007.

CRAIG, J. J. *Introduction to robotics: mechanics and control, 3/E*. [S.l.]: Pearson Education India, 2009.

CRANE, C. D.; DUFFY, J. *Kinematic analysis of robot manipulators*. [S.l.]: Cambridge University Press, 2008.

DEB, K. *Multi-objective optimization using evolutionary algorithms*. [S.l.]: John Wiley & Sons, 2001. v. 16.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002. Disponível em: <https://doi.org/10.1109/4235.996017>.

DENAVIT, J.; HARTENBERG, R. S. A kinematic notation for lower pair mechanisms based on matrices. 1955.

DIJKSTRA, E. Dijkstra's algorithm. *Dutch scientist Dr. Edsger Dijkstra network algorithm*, 1959.

DUKA, A.-V. Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. *Procedia Technology*, Elsevier, v. 12, n. 1, p. 20–27, 2014. Disponível em: <https://doi.org/10.1016/j.protcy.2013.12.451>.

EDAN, Y. et al. Near-minimum-time task planning for fruit-picking robots. *IEEE transactions on robotics and automation*, IEEE, v. 7, n. 1, p. 48–56, 1991. Disponível em: <https://doi.org/10.1109/70.68069>.

EMMERICH, M. T.; DEUTZ, A. H. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*, Springer, v. 17, n. 3, p. 585–609, 2018. Disponível em: <https://doi.org/10.1007/s11047-018-9685-y>.

FÊDOR, M. Application of inverse kinematics for skeleton manipulation in real-time. In: *Proceedings of the 19th spring conference on Computer graphics*. [s.n.], 2003. p. 203–212. Disponível em: <https://doi.org/10.1145/984952.984986>.

FERGUSON, D.; LIKHACHEV, M.; STENTZ, A. A guide to heuristic-based path planning. In: *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*. [S.l.: s.n.], 2005. p. 9–18.

FILHO, H. L. J.; FERREIRA, T. N.; VERGILIO, S. R. Incorporating user preferences in a software product line testing hyper-heuristic approach. In: IEEE. *2018 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.], 2018. p. 1–8.

FOGEL, D. B.; ATMAR, J. W. Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, Springer, v. 63, n. 2, p. 111–114, 1990. Disponível em: <https://doi.org/10.1007/BF00203032>.

FORTIN, F.-A.; PARIZEAU, M. Revisiting the nsga-ii crowding-distance computation. In: ACM. *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 2013. p. 623–630. Disponível em: <https://doi.org/10.1145/2463372.2463456>.

FRACCAROLI, E. S. *Análise de desempenho de algoritmos evolutivos no domínio do futebol de robôs*. Tese (Doutorado) — Universidade de São Paulo, 2010.

GANAPATHY, V.; YUN, S. C.; CHIEN, T. W. Enhanced d* lite algorithm for autonomous mobile robot. *International Journal of Applied*, v. 1, n. 1, p. 58–73, 2011.

GARG, D. P.; KUMAR, M. Optimization techniques applied to multiple manipulators for path planning and torque minimization. *Engineering applications of artificial intelligence*, Elsevier, v. 15, n. 3-4, p. 241–252, 2002. Disponível em: <https://doi.org/10.1016/S0952-1976(02)00067-2>.

GARRIZ, C.; DOMINGO, R. Development of trajectories through the kalman algorithm and application to an industrial robot in the automotive industry. *IEEE Access*, IEEE, v. 7, p. 23570–23578, 2019. Disponível em: <https://doi.org/10.1109/ACCESS.2019.2899370>.

GAVILANES, J. J. et al. Modeling and simulation of an algorithm for the control of a robotic arm. *KnE Engineering*, p. 153–164, 2018. Disponível em: <https://doi.org/10.18502/keg.v1i2.1492>.

GOLDBERG, A. V. A simple shortest path algorithm with linear average time. In: SPRINGER. *European Symposium on Algorithms*. 2001. p. 230–241. Disponível em: <https://doi.org/10.1007/3-540-44676-1_19>.

GOLDBERG, D. E. Genetic algorithms in search. *Optimization, and MachineLearning*, Addison Wesley Publishing Co. Inc., 1989.

GOLDENBERG, A.; LAWRENCE, D. A generalized solution to the inverse kinematics of robotic manipulators. 1985. Disponível em: <https://doi.org/10.1115/1.3140699>.

GUPTA, A.; UMRAO, S.; KUMAR, S. Optimal path planning in a dynamic environment. 2016.

GURBUZBALABAN, M. et al. When cyclic coordinate descent outperforms randomized coordinate descent. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2017. p. 6999–7007.

HEBERT, M.; MACLACHLAN, R.; CHANG, P. Experiments with driving modes for urban robots. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Mobile Robots XIV*. 1999. v. 3838, p. 140–149. Disponível em: <https://doi.org/10.1117/12.369249>.

HEBERT, P. et al. Supervised remote robot with guided autonomy and teleoperation (surrogate): a framework for whole-body manipulation. In: IEEE. *2015 IEEE international conference on robotics and automation (ICRA)*. 2015. p. 5509–5516. Disponível em: <https://doi.org/10.1109/ICRA.2015.7139969>.

IMAI, H.; ASANO, T. Efficient algorithms for geometric graph search problems. *SIAM Journal on Computing*, SIAM, v. 15, n. 2, p. 478–494, 1986. Disponível em: <https://doi.org/10.1137/0215033>.

International Federation of Robotics. *IFR World Robotics Presentation*. Shanghai: [s.n.], 2019. Disponível em: <https://ifr.org/downloads/press2018/IFRWorldRoboticsPresentation-18Sept2019.pdf>.

JENSEN, M. T. Reducing the run-time complexity of multiobjective eas: The nsga-ii and other algorithms. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 7, n. 5, p. 503–515, 2003. Disponível em: <https://doi.org/10.1109/TEVC.2003.817234>.

JIANG, A.; YAO, X.; ZHOU, J. Research on path planning of real-time obstacle avoidance of mechanical arm based on genetic algorithm. *The Journal of Engineering*, IET, v. 2018, n. 16, p. 1579–1586, 2018. Disponível em: <https://doi.org/10.1049/joe.2018.8266>.

JONG, K.; BRANDON, A. D. H. An analysis of the behavior of a class of genetic adaptive systems [ph. d. thesis]. 1975.

JR, J. D. K.; DAVIS, L. A hybrid genetic algorithm for classification. In: *IJCAI*. [S.l.: s.n.], 1991. v. 91, p. 645–650.

KANNAN, S. et al. Application of nsga-ii algorithm to generation expansion planning. *IEEE Transactions on Power systems*, IEEE, v. 24, n. 1, p. 454–461, 2008. Disponível em: <https://doi.org/10.1109/TPWRS.2008.2004737>.

KAZEM, B. I.; MAHDI, A. I.; OUDAH, A. T. Motion planning for a robot arm by using genetic algorithm. *Jjmie*, v. 2, n. 3, p. 131–136, 2008.

KENWRIGHT, B. Inverse kinematics–cyclic coordinate descent (ccd). *Journal of Graphics Tools*, Taylor & Francis, v. 16, n. 4, p. 177–217, 2012. Disponível em: <https://doi.org/10.1080/2165347X.2013.823362>.

KIM, B. K.; SHIN, K. G. Minimum-time path planning for robot arms and their dynamics. *IEEE transactions on systems, man, and cybernetics*, IEEE, n. 2, p. 213–223, 1985.

KIM, H. et al. A study on 3d optimal path planning for quadcopter uav based on d* lite. In: IEEE. *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2019. p. 787–793. Disponível em: <https://doi.org/10.1109/ICUAS.2019.8797815>.

KOENIG, S.; LIKHACHEV, M. Dˆ* lite. *Aaai/iaai*, v. 15, 2002.

KOENIG, S.; LIKHACHEV, M.; FURCY, D. Lifelong planning a. *Artificial Intelligence*, Elsevier, v. 155, n. 1-2, p. 93–146, 2004. Disponível em: <https://doi.org/10.1016/j.artint.2003.12.001>.

KÖKER, R. A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization. *Information Sciences*, Elsevier, v. 222, p. 528–543, 2013. Disponível em: <https://doi.org/10.1016/j.ins.2012.07.051>.

LACERDA, E. G. de; CARVALHO, A. D. Introdução aos algoritmos genéticos. *Sistemas inteligentes: aplicaçoes a recursos hıdricos e ciências ambientais*, v. 1, p. 99–148, 1999.

LANDER, J. Making kine more flexible. *Game Developer Magazine*, v. 1, n. 15-22, p. 2, 1998.

LANDER, J. Oh my god, i inverted kine. *Game Developer Magazine*, v. 9, p. 9–14, 1998.

LEMBONO, T. S. et al. Memory of motion for warm-starting trajectory optimization. *IEEE Robotics and Automation Letters*, IEEE, v. 5, n. 2, p. 2594–2601, 2020. Disponível em: <https://doi.org/10.1109/LRA.2020.2972893>.

LI, H.; ZHANG, Q. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 13, n. 2, p. 284–302, 2008. Disponível em: <https://doi.org/10.1109/TEVC.2008.925798>.

LIKHACHEV, M. *Search techniques for planning in large dynamic deterministic and stochastic environments*. Tese (Doutorado) — Thesis proposal. School of Computer Science, Carnegie Mellon University, 2003.

LINDEN, R. *Algoritmos genéticos (2a ediçao)*. [S.l.]: Brasport, 2008.

LIU, T.-K.; CHEN, C.-H.; TSAI, S.-E. Optimization on robot arm machining by using genetic algorithms. In: . [s.n.], 2007. Disponível em: <https://doi.org/10.1117/12.784513>.

LIU, X. et al. Costar: A d-star lite-based dynamic search algorithm for codon optimization. *Journal of theoretical biology*, Elsevier, v. 344, p. 19–30, 2014. Disponível em: <https://doi.org/10.1016/j.jtbi.2013.11.022>.

LU, Y. et al. Incremental multi-scale search algorithm for dynamic path planning with low worst-case complexity. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 41, n. 6, p. 1556–1570, 2011. Disponível em: <https://doi.org/10.1109/TSMCB.2011.2157493>.

MAHDAVIAN, M. et al. Optimal trajectory generation for energy consumption minimization and moving obstacle avoidance of a 4dof robot arm. In: IEEE. *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*. 2015. p. 353–358. Disponível em: <https://doi.org/10.1109/ICRoM.2015.7367810>.

MANI, I.; BLOEDORN, E. Multi-document summarization by graph search and matching. *arXiv preprint cmp-lg/9712004*, 1997.

MARQUES, F. A. C. *A flexible navigation system for autonomous robots operating in heterogeneous environments*. Tese (Doutorado) — Faculdade de Ciências e Tecnologia, 2012.

MARTIN, A.; BARRIENTOS, A.; CERRO, J. del. The natural-ccd algorithm, a novel method to solve the inverse kinematics of hyper-redundant and soft robots. *Soft robotics*, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 5, n. 3, p. 242–257, 2018. Disponível em: <https://doi.org/10.1089/soro.2017.0009>.

MATTHIES, L. et al. A portable, autonomous, urban reconnaissance robot. *Robotics and Autonomous Systems*, Elsevier, v. 40, n. 2-3, p. 163–172, 2002. Disponível em: <https://doi.org/10.1016/S0921-8890(02)00241-5>.

MCCLOY, D. *Robotics: an introduction*. [S.l.]: Springer Science & Business Media, 2013.

MENGANA, B. *Path Planning for Autonomous Heavy DutyVehicles in Unknown Environments using GraphSearch*. 2013.

MIRANDA, M. N. de. *Algoritmos Genéticos: Fundamentos e Aplicações*. 2007.

MOHAMMED, A. A.; SUNAR, M. Kinematics modeling of a 4-DOF robotic arm. In: *Proceedings - 2015 International Conference on Control, Automation and Robotics, ICCAR 2015*. [S.l.: s.n.], 2015. ISBN 9781467375238.

NARAYAN, S. et al. A priority based exploration algorithm for path planning of an unmanned ground vehicle. In: IEEE. *2014 International Conference on Embedded Systems (ICES)*. 2014. p. 275–280. Disponível em: <https://doi.org/10.1109/EmbeddedSys.2014.6953174>.

NEARCHOU, A. C. Solving the inverse kinematics problem of redundant robots operating in complex environments via a modified genetic algorithm. *Mechanism and machine theory*, Elsevier, v. 33, n. 3, p. 273–292, 1998. Disponível em: <https://doi.org/10.1016/S0094-114X(97)00034-7>.

NOF, S. Y. *Handbook of industrial robotics*. John Wiley & Sons, 1999. Disponível em: <https://doi.org/10.1002/9780470172506>.

OLSEN, A. L.; PETERSEN, H. G. Inverse kinematics by numerical and analytical cyclic coordinate descent. *Robotica*, Cambridge University Press, v. 29, n. 4, p. 619–626, 2011. Disponível em: <https://doi.org/10.1017/S026357471000038X>.

ORAL, T. *Multi-objective path planning for virtual environments*. Dissertação (Mestrado), 2012.

ORAL, T.; POLAT, F. A multi-objective incremental path planning algorithm for mobile agents. In: IEEE. *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. 2012. v. 2, p. 401–408. Disponível em: <https://doi.org/10.1109/WI-IAT.2012.143>.

PACHECO, M. A. C. et al. Algoritmos genéticos: princípios e aplicações. *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, p. 28, 1999.

PARK, C.; PARK, J. S.; MANOCHA, D. Fast and bounded probabilistic collision detection for high-dof trajectory planning in dynamic environments. *IEEE Transactions on Automation Science and Engineering*, IEEE, v. 15, n. 3, p. 980–991, 2018. Disponível em: <https://doi.org/10.1109/TASE.2018.2801279>.

PAZOS, F. *Automação de sistema e robótica*. [S.l.]: Axel Books, 2002.

PIMENTA, A. H. d. M. et al. Geração genética multiobjetivo de bases de conhecimento fuzzy com enfoque na distribuição das soluções não dominadas. Universidade Federal de São Carlos, 2014.

POBIL, A. del; SERNA, M. Robot motion planning. *WIT Transactions on Information and Communication Technologies*, WIT Press, v. 1, 1993.

PRADEEP, V.; MEDIONI, G.; WEILAND, J. A wearable system for the visually impaired. In: IEEE. *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*. 2010. p. 6233–6236. Disponível em: <https://doi.org/10.1109/IEMBS.2010.5627715>.

PRZYBYLSKI, M.; PUTZ, B. D* extra lite: A dynamic a* with search–tree cutting and frontier–gap repairing. *International Journal of Applied Mathematics and Computer Science*, Sciendo, v. 27, n. 2, p. 273–290, 2017. Disponível em: <https://doi.org/10.1515/amcs-2017-0020>.

QIN, C.; HENRICH, D. „path planning for industrial robot arms-a parallel randomized approach ". In: CITESEER. *Proc. of the Int. Symp. on Intelligent Robotic Systems (SIRS 96), Lissabon, Portugal*. [S.l.], 1996. p. 65–72.

RADAVELLI, L. et al. A comparative study of the kinematics of robots manipulators by denavit-hartenberg and dual quaternion. *Mecánica Computacional, Multi-Body Systems*, v. 31, n. 15, p. 2833–2848, 2012.

RADCLIFFE, N. J. Non-linear genetic representations. In: *PPSN*. [S.l.: s.n.], 1992. p. 261–270.

RAJASEKARAN, K. et al. Imaging-guided collision-free transport of multiple optically trapped beads. In: IEEE. *2017 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS)*. 2017. p. 1–6. Disponível em: <https://doi.org/10.1109/MARSS.2017.8001940>.

RANKY, P. G.; HO, C. Y. *Robot modelling: control and applications with software*. [S.l.]: IFS (Publications), 1985.

RIESEN, K.; FANKHAUSER, S.; BUNKE, H. Speeding up graph edit distance computation with a bipartite heuristic. In: *MLG*. [S.l.: s.n.], 2007. p. 21–24.

ROSÁRIO, J. M. *Princípios de mecatrônica*. [S.l.]: Pearson Educación, 2005.

ROSHANINESHAT, A. *Evolutionary Optimization for Safe Navigation of an Autonomous Robot in Cluttered Dynamic Unknown Environments*. Tese (Doutorado) — Cleveland State University, 2018.

SAHA, S.; JULIUS, A. A. Task and motion planning for manipulator arms with metric temporal logic specifications. *IEEE robotics and automation letters*, IEEE, v. 3, n. 1, p. 379–386, 2017. Disponível em: <https://doi.org/10.1109/LRA.2017.2755078>.

SAVSANI, P.; JHALA, R.; SAVSANI, V. J. Optimized trajectory planning of a robotic arm using teaching learning based optimization (tlbo) and artificial bee colony (abc) optimization techniques. In: IEEE. *2013 IEEE International Systems Conference (SysCon)*. 2013. p. 381–386. Disponível em: <https://doi.org/10.1109/SysCon.2013.6549910>.

SAVSANI, P.; JHALA, R.; SAVSANI, V. J. Comparative study of different metaheuristics for the trajectory planning of a robotic arm. *IEEE Systems Journal*, IEEE, v. 10, n. 2, p. 697–708, 2014. Disponível em: <https://doi.org/10.1109/JSYST.2014.2342292>.

SCHILLING, R. J. *Fundamentals of robotics: analysis and control*. [S.l.]: Simon & Schuster Trade, 1996.

SEEREERAM, S.; WEN, J. T. A global approach to path planning for redundant manipulators. *IEEE Transactions on Robotics and Automation*, IEEE, v. 11, n. 1, p. 152–160, 1995. Disponível em: <https://doi.org/10.1109/70.345948>.

SHARMA, G. S.; SINGH, M.; SINGH, T. Optimization of energy in robotic arm using genetic algorithm. *International Journal of Computer Science and Technology*, Citeseer, v. 2, n. 2, p. 315–317, 2011.

SHAW, L. A.; CHIZARI, S.; HOPKINS, J. B. Improving the throughput of automated holographic optical tweezers. *Applied optics*, Optical Society of America, v. 57, n. 22, p. 6396–6402, 2018. Disponível em: <https://doi.org/10.1364/AO.57.006396>.

SICILIANO, B. et al. *Robotics: modelling, planning and control*. [S.l.]: Springer Science & Business Media, 2010.

SON, Y.; LEE, Y. Reconstructible s/w model for evolutionary robots in multi-robot cooperation environments. *International Information Institute (Tokyo). Information*, International Information Institute, v. 15, n. 3, p. 1199, 2012.

SOUZA, A. C. de; DIAS, B. C. D.; OLIVEIRA, B. A. S. Algoritmo evolucionário híbrido aplicado a problemas com muitos objetivos. In: *VIII SEMINÁRIO DE INICIAÇÃO CIENTÍFICA IFMG-RIBEIRÃO DAS NEVES*. [S.l.: s.n.], 2019.

STENTZ, A.; HEBERT, M. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, Springer, v. 2, n. 2, p. 127–145, 1995. Disponível em: <https://doi.org/10.1007/BF00735431>.

ŠTEVO, S.; SEKAJ, I.; DEKAN, M. Optimization of robotic arm trajectory using genetic algorithm. In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*. [s.n.], 2014. ISBN 9783902823625. ISSN 14746670. Disponível em: <https://doi.org/10.3182/20140824-6-ZA-1003.01073>.

STOLLE, M.; ATKESON, C. G. Policies based on trajectory libraries. In: IEEE. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* [S.l.], 2006. p. 3344–3349.

SUDHAKARA, P. et al. Optimal trajectory planning based on improved ant colony optimization with ferguson's spline technique for mobile robot. *International Journal of Control Theory and Applications*, v. 9, n. 40, p. 623–634, 2016.

SUN, X.; YEOH, W.; KOENIG, S. Moving target d* lite. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1.* [S.l.: s.n.], 2010. p. 67–74.

TAN, H. H.; POTTS, R. B. Minimum time trajectory planner for the discrete dynamic robot model with dynamic constraints. *IEEE Journal on Robotics and Automation*, IEEE, v. 4, n. 2, p. 174–185, 1988. Disponível em: <https://doi.org/10.1109/56.2081>.

THAYER, S. M. et al. Distributed robotic mapping of extreme environments. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Mobile Robots XV and Telemanipulator and Telepresence Technologies VII.* [S.l.], 2001. v. 4195, p. 84–95.

TIAN, L.; COLLINS, C. An effective robot trajectory planning method using a genetic algorithm. *Mechatronics*, Elsevier, v. 14, n. 5, p. 455–470, 2004. Disponível em: <https://doi.org/10.1016/j.mechatronics.2003.10.001>.

TOOGOOD, R.; HAO, H.; WONG, C. Robot path planning using genetic algorithms. In: IEEE. *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century.* [S.l.], 1995. v. 1, p. 489–494.

TSAI, L.-W. *Robot analysis: the mechanics of serial and parallel manipulators.* [S.l.]: John Wiley & Sons, 1999.

VAREDI-KOULAEI, S. M.; MOKHTARI, M. Trajectory tracking solution of a robotic arm based on optimized ann. In: IEEE. *2018 6th RSI International Conference on Robotics and Mechatronics (IcRoM).* 2018. p. 76–81. Disponível em: <https://doi.org/10.1109/ICRoM.2018.8657567>.

VARGAS, D. E. Um estudo dos parametros do algoritmo nsga-ii com o operador sbx em problemas de otimizacao estrutural multiobjetivo. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, v. 6, n. 2, 2018. Disponível em: <https://doi.org/10.5540/03.2018.006.02.0333>.

VOLIOTIS, S.; PANOPOULOS, G.; CHRISTODOULOU, M. Co-ordinated path tracking by two robot arms with a noninverting algorithm based on the simplex method. In: IET. *IEE Proceedings D-Control Theory and Applications.* 1990. v. 137, n. 6, p. 390–396. Disponível em: <https://doi.org/10.1049/ip-d.1990.0053>.

VOORNEVELD, M. Characterization of pareto dominance. *Operations Research Letters*, Elsevier, v. 31, n. 1, p. 7–11, 2003. Disponível em: <https://doi.org/10.1016/S0167-6377(02)00189-X>.

VUKOBRATOVIC, M. *Introduction to robotics.* [S.l.]: Springer Science & Business Media, 2012.

WANG, L.-C.; CHEN, C.-C. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, IEEE, v. 7, n. 4, p. 489–499, 1991. Disponível em: <https://doi.org/10.1109/70.86079>.

WHITLEY, D. A genetic algorithm tutorial. *Statistics and computing*, Springer, v. 4, n. 2, p. 65–85, 1994. Disponível em: <https://doi.org/10.1007/BF00175354>.

WHITMAN, J.; CHOSET, H. Task-specific manipulator design and trajectory synthesis. *IEEE Robotics and Automation Letters*, IEEE, v. 4, n. 2, p. 301–308, 2018. Disponível em: <https://doi.org/10.1109/LRA.2018.2890206>.

WRIGHT, A. H. Genetic algorithms for real parameter optimization. In: *Foundations of genetic algorithms*. Elsevier, 1991. v. 1, p. 205–218. Disponível em: <https://doi.org/10.1016/B978-0-08-050684-5.50016-1>.

YANG, L. et al. Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, Hindawi, v. 2016, 2016. Disponível em: <https://doi.org/10.1155/2016/7426913>.

YANO, F.; TOYODA, Y. Preferable movement of a multijoint robot arm using a genetic algorithm. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Intelligent Robots and Computer Vision XVIII: Algorithms, Techniques, and Active Vision*. [S.l.], 1999. v. 3837, p. 80–88.

YEASMIN, S.; SHILL, P. C.; PAUL, A. K. A new method for smooth trajectory planning for 3r robot arm using enhanced particle swarm optimization algorithm. In: IEEE. *2017 3rd International Conference on Electrical Information and Communication Technology (EICT)*. 2017. p. 1–6. Disponível em: <https://doi.org/10.1109/EICT.2017.8275209>.

YETIM, C. Kinematic analysis for robot arm. *Yildiz Technical University, Electrical and Electronics faculty, Department of Computer Engineering. Istanbul*, 2009.

YUE, S. et al. Point-to-point trajectory planning of flexible redundant robot manipulators using genetic algorithms. *Robotica*, Cambridge University Press, v. 20, n. 3, p. 269–280, 2002. Disponível em: <https://doi.org/10.1017/S0263574701003861>.

ZENG, J. et al. A cyclic coordinate descent algorithm for lq regularization. *arXiv preprint arXiv:1408.0578*, 2014.

ZHANG, Z. et al. Tricriteria optimization-coordination motion of dual-redundant-robot manipulators for complex path planning. *IEEE Transactions on Control Systems Technology*, IEEE, v. 26, n. 4, p. 1345–1357, 2017. Disponível em: <https://doi.org/10.1109/TCST.2017.2709276>.

ZHU, D.; LATOMBE, J.-C. *New heuristic algorithms for efficient hierarchical path planning*. [S.l.], 1991. Disponível em: <https://doi.org/10.1109/70.68066>.

ZLOT, R. et al. Multi-robot exploration controlled by a market economy. In: IEEE. *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. [S.l.], 2002. v. 3, p. 3016–3023.

Apêndices

A | Dados relacionados aos trabalhos abordados na revisão literária

Tabela 15 – Autores em relação ao tipo de método e ambiente utilizado em seu trabalho.

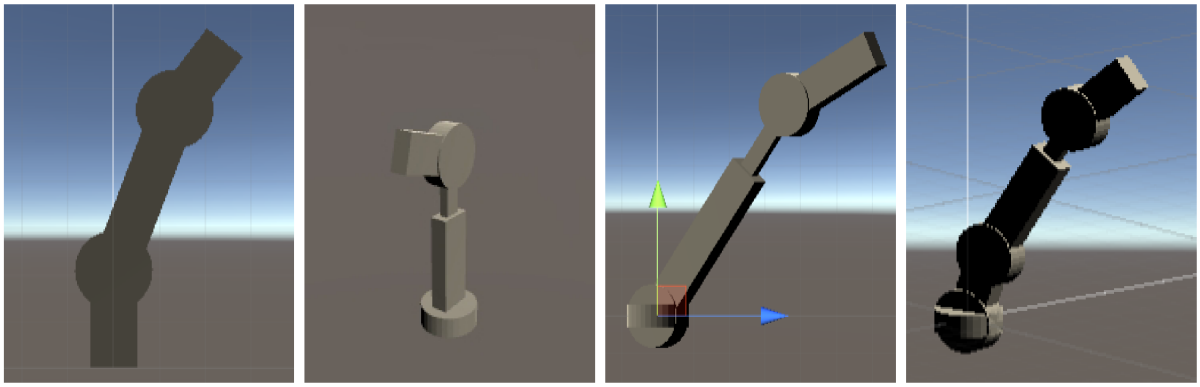
Autores	Ambiente	Método
Banga, Singh e Kumar (2007)	2D	MH
Barac (2010)	2D	MH
Barakat, Gouda e Bozed (2016)	3D	MN
Bessonnet e Lallemand (1990)	2D	MN
Bianco e Piazzi (1999)	2D	MH
Chen (1992)	2D	MN
Chu, Hu e Zhang (2017)	3D	MH
Duka (2014)	2D	MH
Edan et al. (1991)	3D	MH
Ganapathy, Yun e Chien (2011)	2D	MH
Garg e Kumar (2002)	2D	MH
Garriz e Domingo (2019)	3D	MN
Gavilanes et al. (2018)	3D	MH
Goldenberg e Lawrence (1985)	3D	MN
Gupta, Umrao e Kumar (2016)	2D	MH
Hebert et al. (2015)	2D	MH
Jiang, Yao e Zhou (2018)	2D	MH
Kazem, Mahdi e Oudah (2008)	2D	MH
Kim e Shin (1985)	2D	MH
KöKer (2013)	3D	MH
Lembono et al. (2020)	3D	MH
Liu, Chen e Tsai (2007)	3D	MH
Liu et al. (2014)	2D	MH
Mahdavian et al. (2015)	3D	MH
Mohammed e Sunar (2015)	3D	MN
Narayan et al. (2014)	2D	MH

Nearchou (1998)	3D	MH
Park, Park e Manocha (2018)	3D	MH
Pradeep, Medioni e Weiland (2010)	3D	MH
Przybylski e Putz (2017)	2D	MH
Qin e Henrich (1996)	3D	MH
Rajasekaran et al. (2017)	2D	MH
Roshanineshat (2018)	2D	MH
Saha e Julius (2017)	3D	MH
Savsani, Jhala e Savsani (2013)	2D	MH
Savsani, Jhala e Savsani (2014)	2D	MH
Seereeram e Wen (1995)	3D	MN
Sharma, Singh e Singh (2011)	2D	MH
Shaw, Chizari e Hopkins (2018)	2D	MH
Son e Lee (2012)	2D	MH
Števo, Sekaj e Dekan (2014)	3D	MH
Stolle e Atkeson (2006)	2D	MH
Sudhakara et al. (2016)	2D	MH
Tan e Potts (1988)	2D	MN
Tian e Collins (2004)	2D	MH
Toogood, Hao e Wong (1995)	3D	MH
Varedi-Koulaei e Mokhtari (2018)	2D	MH
Voliotis, Panopoulos e Christodoulou (1990)	3D	MN
Whitman e Choset (2018)	3D	MH
Yano e Toyoda (1999)	2D	MH
Yeasmin, Shill e Paul (2017)	2D	MH
Yetim (2009)	3D	MN
Yue et al. (2002)	2D	MH
Zhang et al. (2017)	3D	MH

B | Dados dos robôs utilizados na análise de sensibilidade do NSGA-II

Figura 71 – Geometrias dos robôs para análise de sensibilidade.

(a) 2 G.L. (b) 3 G.L. (c) 4 G.L. (d) 5 G.L.



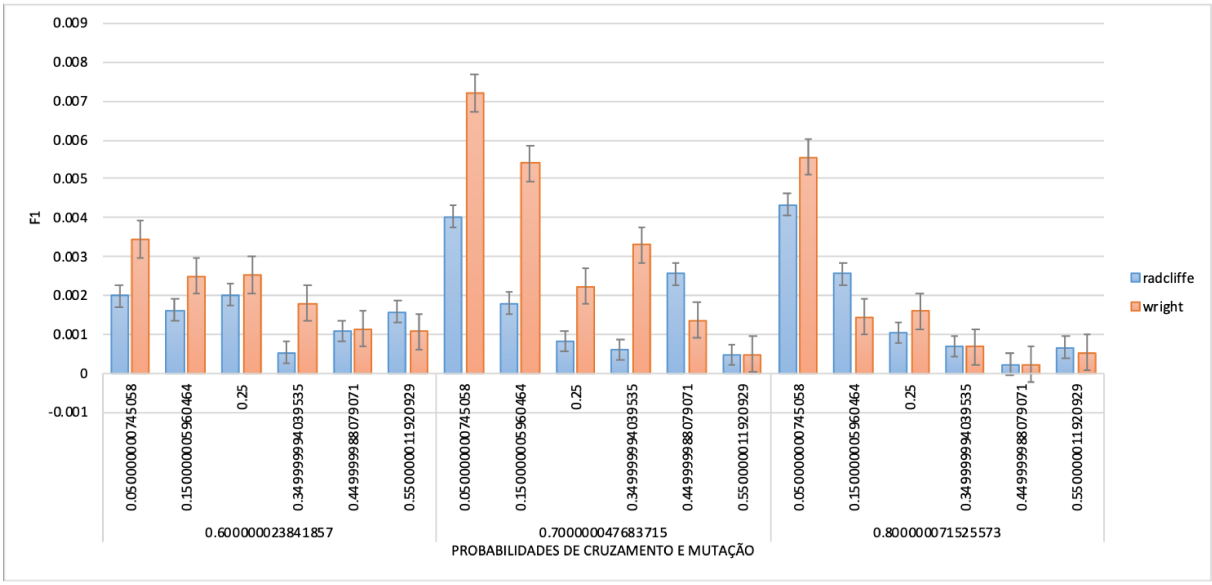
Fonte: Autoria própria

Tabela 16 – Valores usados para os manipuladores da análise de sensibilidade.

	Juntas	Valores
2 graus de liberdade	L_0	21 cm
	R_0	$0° \leq \theta \leq 30°$
	L_1	37 cm
	R_1	$-10° \leq \theta \leq 20°$
	L_2	18 cm
3 graus de liberdade	T_0	$0° \leq \alpha \leq 360°$
	d_1	$30 \leq d \leq 55$ cm
	R_1	$-120° \leq \theta \leq 120°$
	L_1	20 cm
4 graus de liberdade	T_0	$-60° \leq \alpha \leq 60°$
	R_0	$0° \leq \theta \leq 45°$
	d	$50 \leq d \leq 96$ cm
	R_1	$-90° \leq \theta \leq 45°$

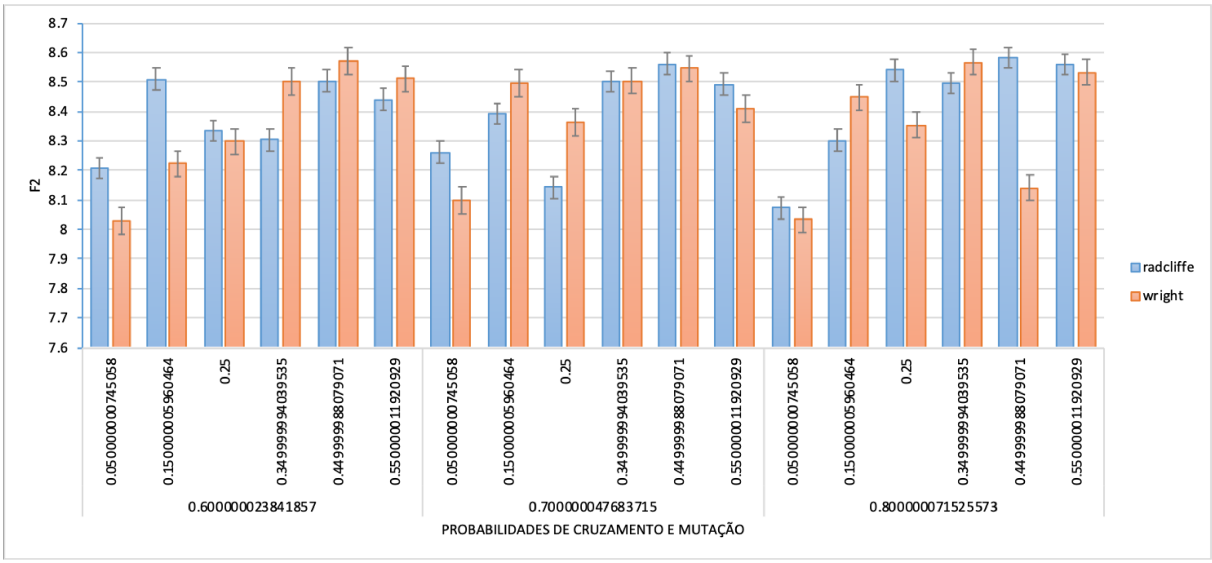
	L_1	33 cm
5 graus de liberdade	T_0	$0° \leq \alpha \leq 360°$
	R_0	$-45° \leq \theta \leq 45°$
	L_0	20 cm
	R_1	$-30° \leq \theta \leq 30°$
	d_1	$35 \leq d \leq 61$ cm
	R_2	$0° \leq \theta \leq 60°$
	L_2	23 cm

Figura 72 – Gráfico do comportamento de f1 para robôs com 2 G.L..



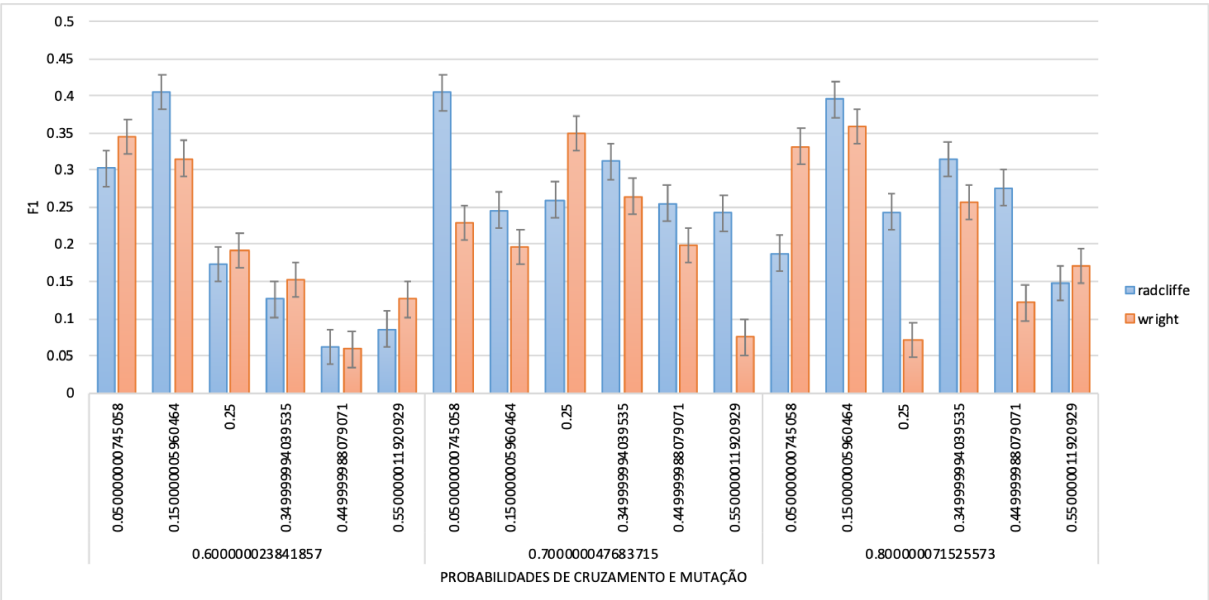
Fonte: Autoria própria

Figura 73 – Gráfico do comportamento de f2 para robôs com 2 G.L..



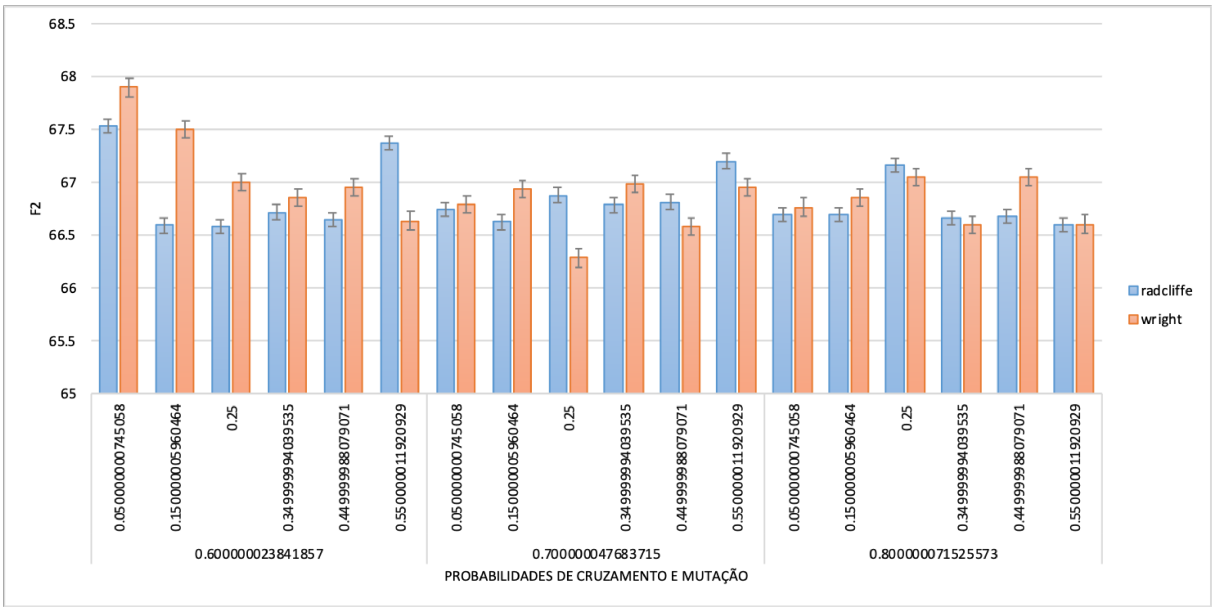
Fonte: Autoria própria

Figura 74 – Gráfico do comportamento de f1 para robôs com 3 G.L..



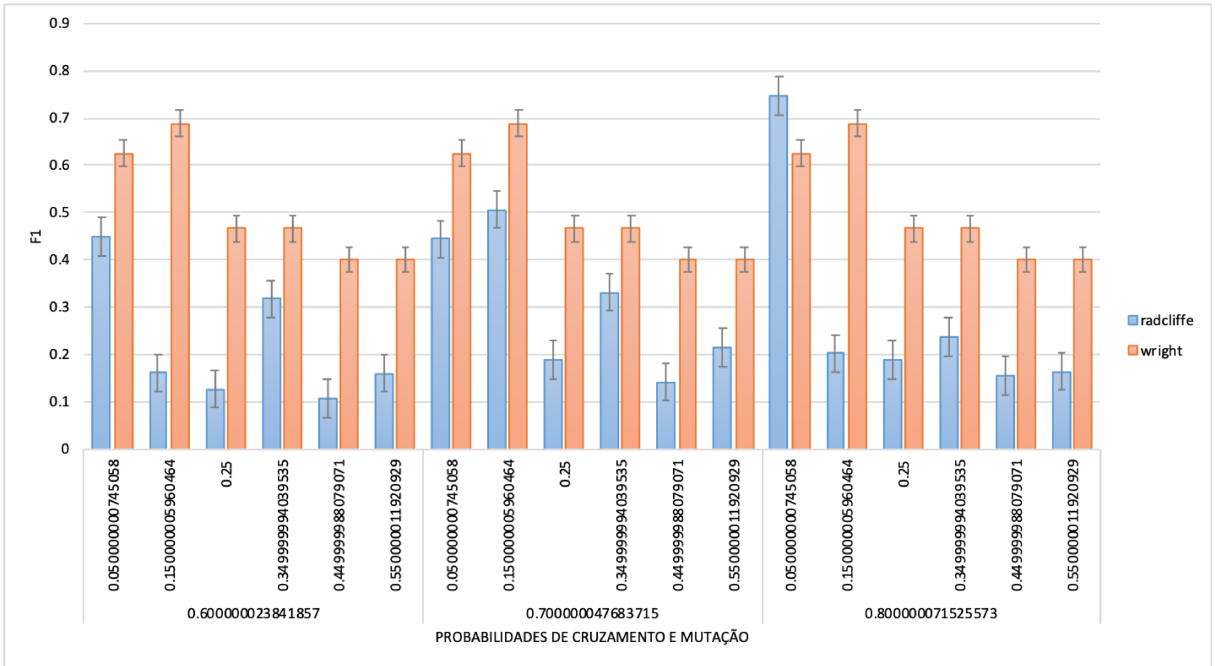
Fonte: Autoria própria

Figura 75 – Gráfico do comportamento de f2 para robôs com 3 G.L..



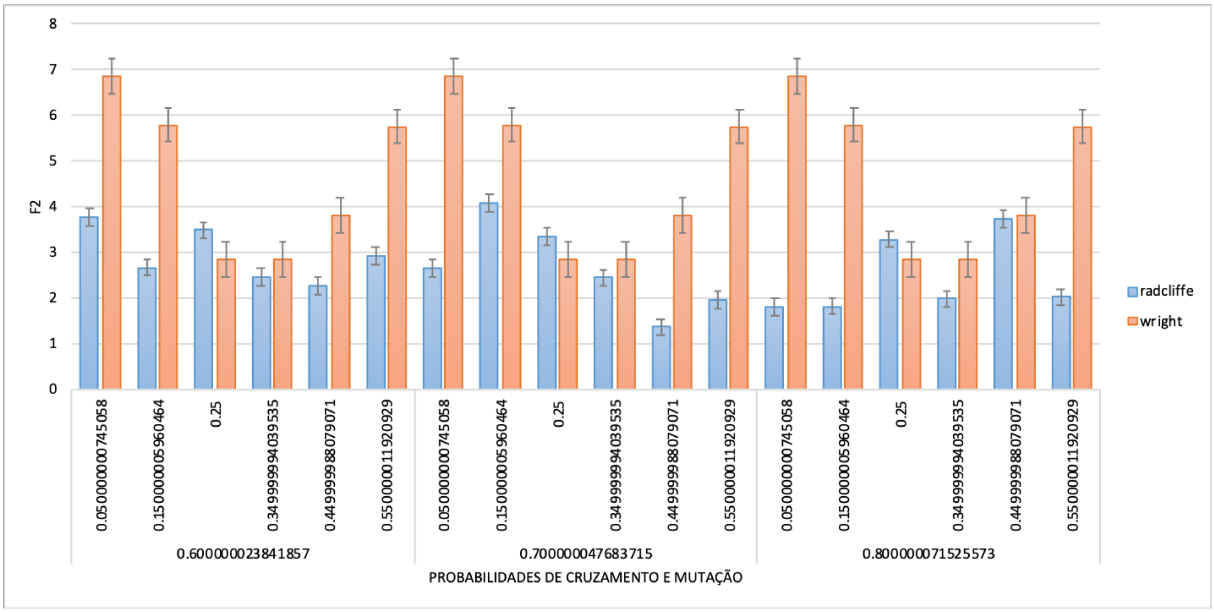
Fonte: Autoria própria

Figura 76 – Gráfico do comportamento de f1 para robôs com 4 G.L..



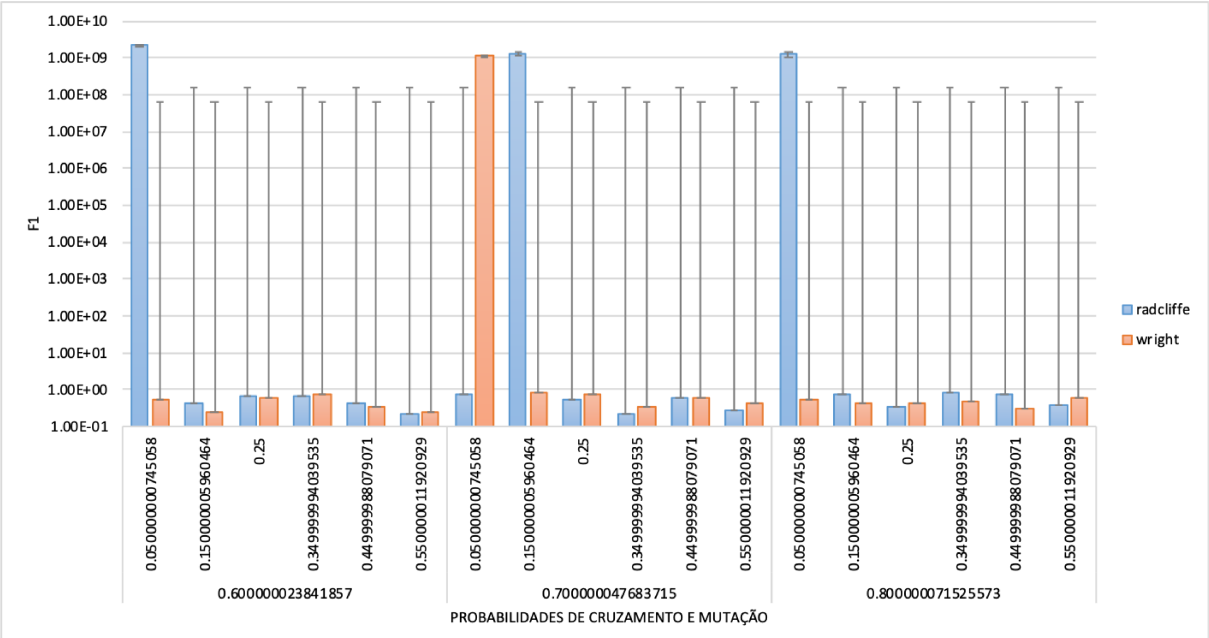
Fonte: Autoria própria

Figura 77 – Gráfico do comportamento de f2 para robôs com 4 G.L..



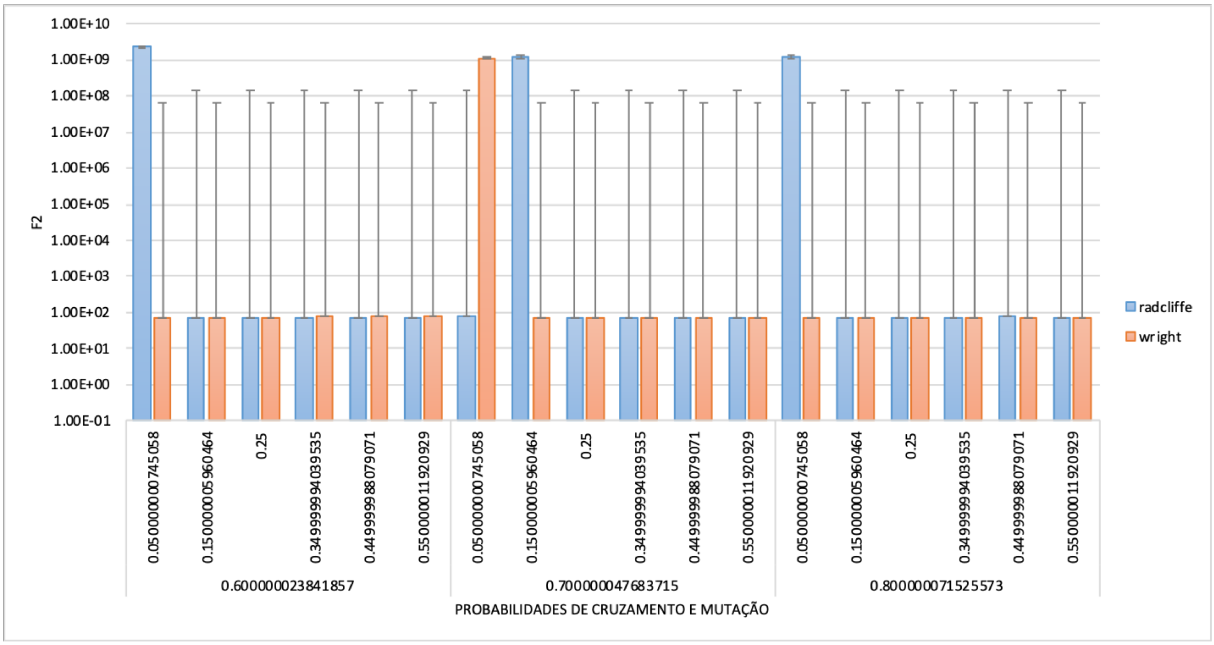
Fonte: Autoria própria

Figura 78 – Gráfico do comportamento de f1 para robôs com 5 G.L..



Fonte: Autoria própria

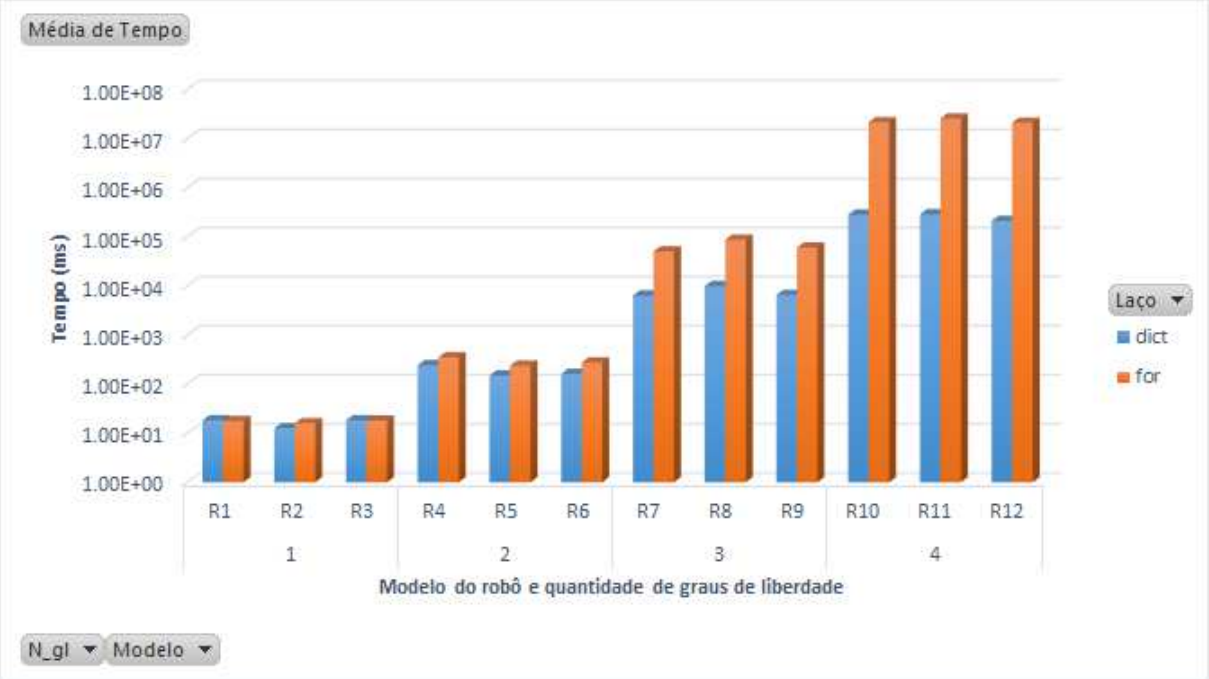
Figura 79 – Gráfico do comportamento de f2 para robôs com 5 G.L..



Fonte: Autoria própria

C | Geometrias e valores de tempo obtidos nos testes de desempenho do laço for vs dicionário

Figura 80 – Gráfico de tempo gasto em milissegundos para cada modelo com os diferentes tipos de armazenamento dos nós.



Fonte: Autoria própria

Tabela 17 – Tabela dinâmica dos valores de tempo em milissegundos separadas por graus de liberdade, modelo de geometria e tipo de armazenamento dos nós.

	dict	**for**	**Total Geral**
1	**16.44444444**	**17.22222222**	**16.83333333**
R1	18.33333333	17.66666667	18
R2	12.66666667	16	14.33333333
R3	18.33333333	18	18.16666667
2	**184.6666667**	**287.5555556**	**236.1111111**
R4	242.6666667	350.3333333	296.5
R5	149.6666667	237.3333333	193.5
R6	161.6666667	275	218.3333333
3	**7588.222222**	**65868.77778**	**36728.5**
R7	6364.333333	50348.66667	28356.5
R8	9882.666667	86968.33333	48425.5
R9	6517.666667	60289.33333	33403.5
4	**260674.5556**	**22758464**	**11509569.28**
R10	283200.3333	21520176	10901688.17
R11	288030.3333	25682717.67	12985374
R12	210793	21072498.33	10641645.67
Total Geral	**67115.97222**	**5706159.389**	**2886637.681**

- 1 grau de liberdade
  - R1
    - $0° \leq \theta_0 \leq 270°$
    - $l_1 = 3$
  - R2
    - $4 \leq d_0 \leq 10$
  - R3
    - $l_0 = 5$
    - $15° \leq \theta_0 \leq 130°$
    - $l_1 = 4$
- 2 graus de liberdade
  - R4
    - $-25° \leq \alpha_0 \leq 55°$
    - $-120° \leq \theta_0 \leq 0°$
    - $l_1 = 6$

  - R5
    - $0° \leq \theta_0 \leq 135°$
    - $2 \leq d_1 \leq 5$
  - R6
    - $1 \leq d_0 \leq 4$
    - $15° \leq \theta_0 \leq 130°$
    - $l_1 = 2$

- 3 graus de liberdade
  - R7
    - $0° \leq \alpha_0 \leq 60°$
    - $0° \leq \theta_0 \leq 45°$
    - $3 \leq d_1 \leq 6$
  - R8
    - $0° \leq \alpha_0 \leq 180°$
    - $l_1 = 3$
    - $-30° \leq \theta_1 \leq 30°$
    - $l_2 = 2$
    - $60° \leq \alpha_2 \leq 270°$
    - $l_3 = 1$
  - R9
    - $2 \leq d_0 \leq 6$
    - $30° \leq \theta_0 \leq 60°$
    - $-180° \leq \alpha_0 \leq 120°$
    - $l_1 = 3$

- 4 graus de liberdade
  - R10
    - $0° \leq \alpha_0 \leq 360°$
    - $0° \leq \theta_0 \leq 30°$
    - $3 \leq d_1 \leq 5$
    - $15° \leq \theta_1 \leq 45°$
    - $l_2 = 2$

  - R11
    * $l_0 = 4$
    * $-90° \leq \theta_0 \leq 90°$
    * $0° \leq \alpha_0 \leq 180°$
    * $l_1 = 2$
    * $-120° \leq \theta_1 \leq 120°$
    * $3 \leq d_2 \leq 7$
  - R12
    * $5 \leq d_0 \leq 11$
    * $25° \leq \theta_0 \leq 135°$
    * $4 \leq d_1 \leq 9$
    * $-40° \leq \theta_1 \leq 10°$
    * $l_2 = 3$

- 5 graus de liberdade
  - R13
    * $35° \leq \alpha_0 \leq 190°$
    * $-70° \leq \theta_0 \leq 65°$
    * $l_1 = 5$
    * $-220° \leq \alpha_1 \leq 5°$
    * $-175° \leq \theta_1 \leq 90°$
    * $l_2 = 8$
    * $25° \leq \theta_2 \leq 45°$
    * $l_3 = 4$
  - R14
    * $10 \leq d_0 \leq 12$
    * $15° \leq \theta_0 \leq 140°$
    * $6 \leq d_1 \leq 11$
    * $-50° \leq \theta_1 \leq 105°$
    * $2 \leq d_2 \leq 7$

– R15

  * $l_0 = 9$
  * $-132° \leq \theta_0 \leq 141°$
  * $3 \leq d_1 \leq 8$
  * $30° \leq \alpha_1 \leq 268°$
  * $-27° \leq \theta_1 \leq 49°$
  * $7 \leq d_2 \leq 13$

Tabela 18 – Tabela dos testes realizados.

Grau de Liberdade	**Modelo do robô**	**Armazenamento**	**Tempo (ms)**
1	R1	for	18
1	R1	for	17
1	R1	for	18
1	R1	dict	19
1	R1	dict	21
1	R1	dict	24
1	R2	for	15
1	R2	for	14
1	R2	for	19
1	R2	dict	13
1	R2	dict	15
1	R2	dict	14
1	R3	for	16
1	R3	for	18
1	R3	for	20
1	R3	dict	21
1	R3	dict	22
1	R3	dict	22
2	R4	for	347
2	R4	for	351
2	R4	for	353
2	R4	dict	238
2	R4	dict	240
2	R4	dict	239
2	R5	for	241
2	R5	for	237

continua para próxima página

continuação da página anterior

2	R5	for	234
2	R5	dict	153
2	R5	dict	141
2	R5	dict	150
2	R6	for	279
2	R6	for	267
2	R6	for	279
2	R6	dict	163
2	R6	dict	181
2	R6	dict	167
3	R7	for	50458
3	R7	for	50197
3	R7	for	50391
3	R7	dict	6200
3	R7	dict	6246
3	R7	dict	6117
3	R8	for	85958
3	R8	for	88182
3	R8	for	86765
3	R8	dict	9454
3	R8	dict	9325
3	R8	dict	9502
3	R9	for	58637
3	R9	for	62769
3	R9	for	59462
3	R9	dict	6300
3	R9	dict	6285
3	R9	dict	6394
4	R10	for	21832499
4	R10	for	21542619
4	R10	for	21185410
4	R10	dict	282130
4	R10	dict	287235
4	R10	dict	286612
4	R11	for	25683894
4	R11	for	25850626

continua para próxima página

continuação da página anterior			
4	R11	for	25513633
4	R11	dict	280343
4	R11	dict	282019
4	R11	dict	275628
4	R12	for	21557771
4	R12	for	20778423
4	R12	for	20881301
4	R12	dict	211814
4	R12	dict	210144
4	R12	dict	210328
5	R13	dict	14094314
5	R13	dict	13961849
5	R13	dict	14007031
5	R14	dict	7386310
5	R14	dict	7644430
5	R14	dict	7619376
5	R15	dict	10483502
5	R15	dict	10277703
5	R15	dict	10252333

D | Dados dos testes de resultados

Tabela 19 – Resultados obtidos usando a anatomia de manipulador 3R de Banga, Singh e Kumar (2007).

Iteração	θ_0	θ_1	θ_2	**Pontos de Controle**
1	0	0	0	(0.0, 9.2, 0.0)
2	-1.4482	-15.9269	53.9036	(0.0, 8.482, 0.538)
3	-1.4057	-15.3476	71.2951	(0.0, 7.901, 1.155)
4	-1.1527	-12.8159	81.8533	(0.0, 7.403, 1.616)
5	-0.7601	-9.3537	93.0232	(0.0, 6.777, 2.024)
6	0.2927	-1.2577	100.1097	(0.0, 6.078, 2.579)
7	1.8897	8.5580	103.3054	(0.0, 5.378, 3.115)
8	3.3801	16.8903	110.9025	(0.0, 4.551, 3.299)
9	4.8279	24.4780	114.5578	(0.0, 3.939, 3.408)
10	7.4446	34.1415	106.8487	(0.0, 3.408, 3.939)
11	10.2165	42.1145	95.1317	(0.0, 2.969, 4.544)
12	10.5984	43.2190	90.6070	(0.0, 2.977, 4.733)
13	12.9680	48.8639	85.3016	(0.0, 2.5, 5.0)

Tabela 20 – Resultados obtidos usando a anatomia de manipulador 3R sem obstáculos.

Iteração	θ_0	θ_1	θ_2	**Pontos de Controle**
1	0.0	20.1717	-63.8534	(0.0, 2.3, 0.0)
2	-0.0384	18.0218	-89.9447	(0.0, 2.106, -0.167)
3	-0.3650	5.3971	-108.2751	(0.0, 1.882, -0.405)
4	-0.9169	-10.7202	-113.6116	(0.0, 1.691, -0.626)
5	-2.0612	-27.7934	-111.6367	(0.0, 1.476, -0.845)
6	-3.8749	-42.2074	-100.6994	(0.0, 1.273, -1.063)
7	-6.7077	-54.3563	-84.8481	(0.0, 1.063, -1.273)
8	-10.7697	-64.9072	-64.6060	(0.0, 0.845, -1.476)

continua para próxima página

continuação da página anterior				
9	-17.0619	-74.5101	-35.6671	(0.0, 0.626, -1.691)
10	-22.7281	-80.1980	-24.1729	(0.0, 0.397, -1.76)
11	-33.2226	-75.1946	-18.4248	(0.0, 0.221, -1.897)
12	-39.1349	-72.2512	-15.2987	(0.0, 0.112, -1.964)
13	-43.9298	-70.4026	-13.6940	(0.0, 0.0, -2.0)

Tabela 21 – Resultados obtidos usando a anatomia de manipulador 3R com obstáculo.

Iteração	θ_0	θ_1	θ_2	**Pontos de Controle**
1	0.0	20.1811	-63.8290	(0.0, 2.3, 0.0)
2	-0.0263	26.5963	-89.9710	(0.0, 2.118, 0.0)
3	-0.0434	22.5092	-114.9831	(0.0, 1.902, -0.118)
4	0.0605	16.5172	-143.7932	(0.0, 1.655, -0.112)
5	25.4688	-45.3386	-129.1178	(0.0, 1.415, -0.167)
6	57.4986	-108.5704	-16.5134	(0.0, 1.356, -0.397)
7	49.2725	-111.3596	-7.6049	(0.0, 1.294, -0.595)
8	40.3444	-113.7341	-2.8564	(0.0, 1.167, -0.797)
9	31.4200	-115.0679	3.1588	(0.0, 1.047, -0.966)
10	22.3375	-116.2234	4.5351	(0.0, 0.863, -1.118)
11	13.2684	-116.6767	8.3321	(0.0, 0.697, -1.242)
12	-1.6694	-114.4322	23.3486	(0.0, 0.536, -1.427)
13	-16.0337	-107.7192	33.7281	(0.0, 0.405, -1.608)
14	-28.6722	-99.9878	34.9811	(0.0, 0.221, -1.76)
15	-39.0686	-90.5053	37.1188	(0.0, 0.118, -1.902)
16	-42.8829	-85.6557	38.8473	(0.0, 0.112, -1.964)
17	-46.2446	-82.4054	30.9369	(0.0, 0.0, -2.0)

Tabela 22 – Resultados obtidos usando a anatomia de manipulador 3R de Sharma, Singh e Singh (2011).

Iteração	θ_0	θ_1	θ_2	**Pontos de Controle**
1	-1.9506	-4.2480	103.0661	(0.0, 2.5, 1.9)
2	-4.3670	-9.1401	121.3311	(0.0, 2.079, 1.595)
3	-5.0920	-10.1831	137.3624	(0.0, 1.595, 1.319)
4	-3.1496	-8.3816	145.8773	(0.0, 1.265, 1.171)
5	3.8069	-3.9693	149.4854	(0.0, 0.95, 1.17)
continua para próxima página				

continuação da página anterior				
6	6.9371	-1.9233	152.2377	(0.0, 0.807, 1.105)
7	14.2851	1.7549	153.4500	(0.0, 0.6, 1.1)

Tabela 23 – Resultados obtidos usando a anatomia de manipulador 3R de Jiang, Yao e Zhou (2018) com um obstáculo.

Iteração	θ_0	θ_1	θ_2	**Pontos de Controle**
1	29.8057	-3.4380	-88.1932	(0.0, 2.0, 0.5)
2	29.6881	-6.1096	-102.3975	(0.0, 1.882, 0.405)
3	29.6320	-10.1250	-117.5213	(0.0, 1.742, 0.333)
4	29.8593	-16.4240	-132.7803	(0.0, 1.595, 0.294)
5	30.3400	-29.3114	-133.5584	(0.0, 1.524, 0.155)
6	58.0810	-78.5772	-65.5237	(0.0, 1.5, -0.0)
7	55.0879	-83.4574	-58.9468	(0.0, 1.476, -0.155)
8	52.0741	-87.7226	-56.8499	(0.0, 1.405, -0.294)
9	48.6941	-92.0944	-52.8542	(0.0, 1.333, -0.433)
10	44.9639	-95.8722	-45.1835	(0.0, 1.285, -0.567)
11	40.5141	-99.6950	-37.4963	(0.0, 1.214, -0.706)
12	36.2338	-102.9189	-34.8918	(0.0, 1.103, -0.817)
13	30.1608	-106.3499	-20.2418	(0.0, 1.047, -0.966)
14	23.2165	-108.4131	-6.0079	(0.0, 0.992, -1.103)
15	16.5892	-109.6015	0.2308	(0.0, 0.882, -1.214)
16	9.2483	-107.6266	0.2554	(0.0, 0.771, -1.324)
17	0.3626	-101.9546	0.1880	(0.0, 0.7, -1.464)
18	-8.6650	-93.2554	-0.2816	(0.0, 0.676, -1.618)
19	-13.3740	-89.9127	-0.2109	(0.0, 0.626, -1.691)
20	-21.7128	-89.4739	15.1769	(0.0, 0.515, -1.801)
21	-29.2334	-84.7185	27.7367	(0.0, 0.5, -1.902)
22	-35.4347	-77.6988	33.8630	(0.0, 0.515, -1.992)
23	-35.8127	-77.2240	32.2326	(0.0, 0.5, -2.0)

Tabela 24 – Resultados obtidos usando a anatomia de manipulador 3R de Jiang, Yao e Zhou (2018) com dois obstáculos.

Iteração	θ_0	θ_1	θ_2	**Pontos de Controle**
1	17.0161	22.0461	-96.8616	(0.0, 2.0, 0.5)
continua para próxima página				

continuação da página anterior

2	16.9847	19.6930	-112.3981	(0.0, 1.882, 0.405)
3	17.0210	15.4334	-129.0699	(0.0, 1.742, 0.333)
4	17.3116	7.4772	-147.1144	(0.0, 1.595, 0.294)
5	18.7652	-12.3918	-152.6012	(0.0, 1.524, 0.155)
6	64.1002	-89.2623	-46.2559	(0.0, 1.5, -0.0)
7	60.6888	-93.0649	-46.9131	(0.0, 1.427, -0.155)
8	56.9960	-96.7247	-43.5805	(0.0, 1.372, -0.297)
9	53.1086	-100.2030	-42.4058	(0.0, 1.285, -0.433)
10	49.4497	-103.4518	-41.3719	(0.0, 1.191, -0.547)
11	45.4628	-106.3150	-37.9250	(0.0, 1.112, -0.655)
12	39.2514	-109.6716	-22.0655	(0.0, 1.088, -0.809)
13	31.3511	-112.0441	-5.7577	(0.0, 1.047, -0.966)
14	22.7334	-112.9222	5.3236	(0.0, 0.964, -1.112)
15	16.2000	-112.6285	10.2872	(0.0, 0.882, -1.214)
16	8.9741	-110.5086	9.6069	(0.0, 0.771, -1.324)
17	0.2473	-104.2104	7.2344	(0.0, 0.7, -1.464)
18	-8.7262	-94.8804	4.7212	(0.0, 0.676, -1.618)
19	-13.4180	-91.3126	4.0945	(0.0, 0.626, -1.691)
20	-21.9835	-90.0546	17.7379	(0.0, 0.515, -1.801)
21	-29.4135	-84.8818	28.8888	(0.0, 0.5, -1.902)
22	-35.5314	-77.7129	34.3166	(0.0, 0.515, -1.992)
23	-35.9028	-77.2723	32.8176	(0.0, 0.5, -2.0)

Tabela 25 – Resultados obtidos usando a anatomia de manipulador 2R de Tian e Collins (2004) no primeiro experimento.

Iteração	θ_0	θ_1	**Pontos de Controle**
1	-58.5531	-50.2307	(0.0, 0.2, -1.8)
2	-44.9916	-62.9952	(0.0, 0.398, -1.658)
3	-35.9709	-62.9952	(0.0, 0.653, -1.575)
4	-18.0331	-80.9153	(0.0, 0.795, -1.297)
5	-8.9929	-90.0000	(0.0, 0.831, -1.144)
6	-0.0676	-89.9324	(0.0, 1.0, -1.0)
7	8.9992	-89.9801	(0.0, 1.144, -0.831)
8	17.9836	-89.9306	(0.0, 1.26, -0.642)
9	27.0043	-89.9817	(0.0, 1.345, -0.437)
10	36.0142	-89.9817	(0.0, 1.397, -0.221)

continua para próxima página

continuação da página anterior			
11	45.0037	-89.9817	(0.0, 1.414, 0.0)
12	53.9574	-89.9163	(0.0, 1.397, 0.221)
13	63.0121	-89.9934	(0.0, 1.345, 0.437)
14	63.0121	-80.9948	(0.0, 1.405, 0.582)
15	72.0174	-80.9948	(0.0, 1.297, 0.795)
16	72.0174	-63.0455	(0.0, 1.297, 1.107)
17	81.0336	-63.0455	(0.0, 1.107, 1.297)
18	81.0336	-54.0469	(0.0, 1.047, 1.442)
19	81.0336	-45.0855	(0.0, 0.965, 1.575)
20	80.9804	-35.9708	(0.0, 0.864, 1.695)
21	84.8172	-40.0432	(0.0, 0.8, 1.7)

Tabela 26 – Resultados obtidos usando a anatomia de manipulador 2R de Tian e Collins (2004) no segundo experimento.

Iteração	θ_0	θ_1	**Pontos de Controle**
1	-58.5531	-50.2307	(0.0, 0.2, -1.8)
2	-44.9916	-62.9952	(0.0, 0.398, -1.658)
3	-35.9709	-62.9952	(0.0, 0.653, -1.575)
4	-18.0331	-80.9153	(0.0, 0.795, -1.297)
5	-8.9929	-90.0000	(0.0, 0.831, -1.144)
6	-0.0676	-89.9324	(0.0, 1.0, -1.0)
7	8.9992	-89.9801	(0.0, 1.144, -0.831)
8	17.9836	-89.9306	(0.0, 1.26, -0.642)
9	27.0043	-89.9817	(0.0, 1.345, -0.437)
10	36.0142	-89.9817	(0.0, 1.397, -0.221)
11	45.0037	-89.9817	(0.0, 1.414, 0.0)
12	53.9574	-89.9163	(0.0, 1.397, 0.221)
13	63.0121	-89.9934	(0.0, 1.345, 0.437)
14	63.0121	-80.9948	(0.0, 1.405, 0.582)
15	72.0174	-80.9948	(0.0, 1.297, 0.795)
16	81.0110	-80.9948	(0.0, 1.156, 0.988)
17	89.9718	-80.9948	(0.0, 0.988, 1.156)
18	90.0000	-72.0175	(0.0, 0.951, 1.309)
19	90.0000	-63.0170	(0.0, 0.891, 1.454)
20	90.0000	-54.0071	(0.0, 0.809, 1.588)
21	84.9253	-40.2740	(0.0, 0.8, 1.7)

Tabela 27 – Resultados obtidos usando a anatomia de manipulador 2R de Tian e Collins (2004) no terceiro experimento.

Iteração	θ_0	θ_1	**Pontos de Controle**
1	-58.5160	-50.2572	(0.0, 0.2, -1.8)
2	-52.8306	-62.3303	(0.0, 0.179, -1.701)
3	-48.0062	-71.9859	(0.0, 0.169, -1.609)
4	-43.2176	-76.7455	(0.0, 0.229, -1.551)
5	-38.3823	-86.4154	(0.0, 0.213, -1.442)
6	-33.6808	-95.9083	(0.0, 0.195, -1.324)
7	-28.7754	-105.5989	(0.0, 0.177, -1.196)
8	-23.9847	-115.1909	(0.0, 0.157, -1.06)
9	-14.4219	-120.0000	(0.0, 0.269, -0.963)
10	-9.6245	-115.2030	(0.0, 0.415, -0.988)
11	-0.0271	-120.0000	(0.0, 0.5, -0.866)
12	4.7964	-119.9745	(0.0, 0.571, -0.821)
13	9.6147	-115.2411	(0.0, 0.717, -0.796)
14	19.2388	-120.0000	(0.0, 0.757, -0.653)
15	23.9568	-119.9971	(0.0, 0.809, -0.588)
16	28.8186	-115.2389	(0.0, 0.939, -0.516)
17	38.3944	-120.0000	(0.0, 0.93, -0.368)
18	43.1747	-120.0000	(0.0, 0.957, -0.289)
19	47.9662	-115.1835	(0.0, 1.057, -0.179)
20	57.5564	-119.9757	(0.0, 0.999, -0.042)
21	62.3879	-115.1140	(0.0, 1.068, 0.09)
22	72.0098	-120.0000	(0.0, 0.978, 0.208)
23	76.7464	-115.1542	(0.0, 1.012, 0.352)
24	86.3656	-119.9676	(0.0, 0.896, 0.445)
25	91.2233	-120.0000	(0.0, 0.855, 0.518)
26	95.9883	-115.1917	(0.0, 0.84, 0.666)
27	105.5580	-119.9849	(0.0, 0.7, 0.714)
28	110.4384	-120.0000	(0.0, 0.637, 0.771)
29	115.2184	-115.1944	(0.0, 0.574, 0.905)
30	110.4181	-105.6371	(0.0, 0.648, 1.021)
31	110.4181	-100.8393	(0.0, 0.637, 1.104)
32	110.4181	-96.0403	(0.0, 0.62, 1.186)
33	110.3811	-91.1997	(0.0, 0.596, 1.266)
34	105.5941	-81.5811	(0.0, 0.645, 1.37)

continua para próxima página

continuação da página anterior			
35	100.8155	-72.0661	(0.0, 0.689, 1.464)
36	96.0386	-62.4286	(0.0, 0.728, 1.548)
37	91.2415	-52.8876	(0.0, 0.763, 1.621)
38	86.3893	-43.1695	(0.0, 0.792, 1.683)
39	84.9440	-40.2981	(0.0, 0.8, 1.7)

Tabela 28 – Resultados obtidos para o experimento com obstáculo dinâmico usando a anatomia TRD.

Iteração	α_0	θ_0	d_0	**Pontos de Controle**
1	0.0000	90.0000	1.0	(0.0, 0.0, 1.0)
2	0.0000	90.0000	0.85	(0.0, 0.0, 0.85)
3	0.0000	90.0000	0.7	(0.0, 0.0, 0.7)
4	0.0000	90.0000	0.6	(0.0, 0.0, 0.6)
5	0.0000	81.0274	0.5	(0.0, 0.078, 0.494)
6	18.0317	81.0718	0.5003	(0.153, 0.078, 0.47)
7	35.9447	81.0288	0.5003	(0.29, 0.078, 0.4)
8	54.0605	81.0288	0.5003	(0.4, 0.078, 0.29)
9	71.9709	81.0288	0.5003	(0.47, 0.078, 0.153)
10	71.9617	71.9225	0.5000	(0.452, 0.155, 0.147)
11	89.9774	71.9225	0.5009	(0.476, 0.155, 0.0)
12	107.9930	71.9225	0.5000	(0.452, 0.155, -0.147)
13	126.0000	72.0000	0.5000	(0.385, 0.155, -0.28)
14	143.9452	71.9483	0.5008	(0.28, 0.155, -0.385)
15	161.9570	71.9871	0.5000	(0.147, 0.155, -0.452)
16	179.9726	71.9516	0.5009	(0.0, 0.155, -0.476)
17	180.0000	81.0000	0.5000	(0.0, 0.078, -0.494)
18	180.0000	90.0000	0.5000	(0.0, 0.0, -0.5)

Tabela 29 – Resultados obtidos para o experimento com obstáculo dinâmico usando a anatomia TRLTRL.

Iteração	α_0	θ_0	α_1	θ_1	**Pontos de Controle**
1	-90.0000	36.0000	-6.0000	24.0000	(-0.8, 0.5, 0.0)
2	72.0000	-45.0000	-60.0000	24.0000	(-0.757, 0.767, -0.017)
3	66.2588	-42.1049	-52.1584	24.4419	(-0.705, 0.811, -0.081)
continua para próxima página					

continuação da página anterior					
4	68.2883	-34.0619	-43.9851	6.1002	(-0.637, 0.858, -0.137)
5	56.6923	-30.1766	-49.2648	15.1213	(-0.559, 0.927, -0.191)
6	39.0802	-26.7181	-51.7565	21.0373	(-0.451, 0.973, -0.201)
7	45.0000	-18.0000	-60.0000	-3.0000	(-0.342, 0.999, -0.242)
8	-6.3048	-20.0087	-49.5379	30.0000	(-0.234, 1.042, -0.26)
9	2.7303	-19.5339	-0.2493	21.1627	(-0.125, 1.028, -0.251)
10	-12.4828	-4.2817	-16.6879	-12.0336	(-0.011, 1.06, -0.222)
11	-65.0378	-18.1987	-22.6100	20.0312	(0.059, 1.082, -0.152)
12	-60.6197	-15.1837	-59.0081	-3.5771	(0.173, 1.06, -0.139)
13	-63.0000	-18.0000	-42.0000	-12.0000	(0.272, 1.057, -0.129)
14	-90.0000	-31.5000	-42.0000	24.0000	(0.378, 1.019, -0.13)
15	-71.6040	-24.8924	-46.9801	-15.8461	(0.454, 0.972, -0.109)
16	-72.5277	-28.8237	-42.1823	-18.4498	(0.523, 0.953, -0.09)
17	-72.5277	-28.8237	-42.1823	-18.4498	(0.6, 0.9, -0.1)