Universidade Federal de Santa Catarina
Centro de Blumenau
Departamento de Engenharia de
Controle e Automação e Computação

Nikolas Sbaraini Hammes

Algoritmo de visão computacional para recomendação de um manipulador robótico baseado na pose de objetos

Blumenau
2020

Nikolas Sbaraini Hammes

Algoritmo de visão computacional para recomendação de um manipulador robótico baseado na pose de objetos

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para a obtenção do Título de Engenheiro de Controle e Automação.
Orientador: Prof. Dr. Daniel Alejandro Ponce Saldías

Universidade Federal de Santa Catarina
Centro de Blumenau
Departamento de Engenharia de
Controle e Automação e Computação

Blumenau
2020

Nikolas Sbaraini Hammes

Algoritmo de visão computacional para recomendação de um manipulador robótico baseado na pose de objetos

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Engenheiro de Controle e Automação.

Comissão Examinadora

Documento assinado digitalmente
Daniel Alejandro Ponce Saldias
Data: 18/12/2020 21:22:28-0300
CPF: 011.397.049-82

Prof. Dr. Daniel Alejandro Ponce Saldías
Universidade Federal de Santa Catarina
Orientador

Documento assinado digitalmente
Leonardo Mejia Rincon
Data: 18/12/2020 22:36:20-0300
CPF: 011.678.849-65

Prof. Dr. Leonardo Mejia Rincon
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Ebrahim Samer El Youssef
Data: 18/12/2020 21:18:58-0300
CPF: 008.252.109-33

Prof. Dr. Ebrahim Samer El'Youssef
Universidade Federal de Santa Catarina

Blumenau, 18 de dezembro de 2020

Agradecimentos

Agradeço aos meus pais, Lucimar e Sergio, e a minha irmã Isabelle, por toda a educação que me proporcionaram, os valores que me ensinaram e por todo apoio, amor e suporte nos momentos que mais precisei.

Agradeço também à minha namorada Michele, por acreditar sempre em mim, por fazer dos meus sonhos os seus e por estar ao meu lado, me apoiando, em todos os momentos e decisões tomadas.

Ao meu orientador Daniel, muito obrigado por todo o auxílio, conselhos e por ter aceitado essa empreitada mesmo em tempos tão difíceis. Aproveito para agradecer a todos os professores que participaram da minha caminhada, me fornecendo todo o conhecimento necessário para que eu chegasse onde estou hoje, em especial o professor Marcelo R. Petry pelas oportunidades e ensinamentos ao longo destes anos.

A todos os amigos que fiz neste período, os quais divido tantas memórias, minha gratidão. A jornada foi mais leve graças a vocês.

Por fim, agradeço à Universidade Federal de Santa Catarina por ter me proporcionado tantas oportunidades e experiências incríveis.

"Toda a nossa ciência, comparada com a realidade, é primitiva e infantil - e, no entanto, é a coisa mais preciosa que temos."

(Albert Einstein)

Resumo

Uma das principais dificuldades na aplicação de manipuladores modulares é a escolha da configuração que atenda aos requisitos de uma tarefa estabelecida. Desta forma, vemos surgindo muitos trabalhos que buscam auxiliar o usuário otimizando esta decisão, dado um ou mais atributos da tarefa. Este trabalho propõe o projeto de um algoritmo de visão computacional que, diferente dos outros trabalhos encontrados, o usuário deve passar uma imagem com marcadores fiduciais (representando as poses da tarefa) e uma lista de manipuladores, para que, com base nos pontos, o programa retorne uma recomendação de manipulador. Através do uso da linguagem de programação Python, da biblioteca de visão computacional OpenCV e da biblioteca de robótica Pybotics, foi desenvolvido o algoritmo, que mostrou uma precisão aceitável na identificação das poses e uma recomendação de configuração satisfatória para estes pontos. Este trabalho contém o desenvolvimento do projeto, questões relevantes quanto a identificação dos pontos e recomendação dos manipuladores, e os testes realizados.

Palavras-Chave: 1.Robótica Modular. 2.Visão Computacional. 3.Marcador Fiducial.

Abstract

One of the main difficulties in the application of modular manipulators is the choice of the configuration that meets the requirements of an established task. In this way, we see many works that seek to help the user optimizing this decision, given one or more attributes of the task. This work proposes the design of a computer vision algorithm that, unlike the other works found, the user must pass an image with fiducial markers (representing the task poses) and a list of manipulators, so that, based on the points, the program returns a manipulator recommendation. Through the use of the Python programming language, the OpenCV computer vision library and the Pybotics robotics library, the algorithm was developed, which showed acceptable accuracy in the identification of the poses and a satisfactory configuration recommendation for these points. This work contain the development of the project, relevant issues regarding the identification of the points and the recommendation of the manipulators, and the tests performed.

Keywords: 1.Modular Robotics. 2.Computer Vision. 3.Fiducial Markers.

Lista de figuras

Lista de tabelas

Lista de Siglas e Abreviaturas

MMR	*Manipulador Modular Reconfigurável*
GDL	*Graus de Liberdade*
DH	*Denavit-Hartenberg*
MDH	*Denavit-Hartenberg Modificado*
RGB	*Red, Green e Blue*
HSV	*Hue, Saturation e Value*
SO	*Sistema Operacional*

Sumário

1 Introdução

É de conhecimento geral que o mercado de robótica está em expansão, segundo dados do relatório da Federação Internacional de Robótica (International Federation of Robotics) referente ao ano de 2019, o número de robôs industriais operacionais alcançou sua maior marca, 2,7 milhões de unidades, esse valor representa um taxa composta de crescimento anual desde 2014 de 13%. Apesar deste crescimento, nem sempre as necessidades do mercado são atendidas, principalmente em áreas onde uma grande customização do manipulador é necessária para a realização de diferentes tarefas, como células de manufatura flexível e ambientes não estruturados.

Nestes locais, um único robô pode ter que realizar tarefas de montagem, corte e furação, de inspeção e de carregar objetos, e cada uma destas tem uma configuração que atende a demanda de espaço de trabalho e que melhor condiz com a operação a ser realizada. Por exemplo, certos indicadores mostram que o manipulador é mais apontado para tarefas de inspeção, ou tarefas de contato ou até o manuseio de carga útil [1]. Logo, é necessário para estas aplicações um robô capaz de alterar entre diversas configurações a fim de se adaptar a múltiplas funções, estes são os manipuladores modulares reconfiguráveis (MMR).

Logo, este tipo de manipulador tem diversas finalidades, principalmente nas aplicações em ambientes não estruturados, como em missões espaciais [1, 2], no fundo do mar e em desastres [3], em manufatura flexíveis e mais responsivas a flutuações de mercado [4] e na área da educação.

Porém um dos desafios para se adotar robôs modulares é determinar a configuração ótima para uma tarefa. A otimização pode ser baseada em diversos fatores como Graus de Liberdade, tamanho do manipulador, capacidade de carga e outros, e caso não seja realizada, a tarefa pode ficar comprometida, não sendo capaz de atingir requisitos de qualidade desejados. Diversas abordagens foram desenvolvidas para realizar esta determinação, e cada um deles aborda de uma forma diferente. Em [5] os autores buscam encontrar a configuração minimizando os Graus de Liberdade, com intuito de alcançar uma configuração simples, com alta capacidade de carga e baixa taxa de consumo de energia. Isso é feito aplicando um algoritmo evolucionário (AE) para buscar a solução ótima. Os autores de [6] determinam um método onde se deve definir um conjunto de tarefas a ser performada e um conjunto de módulos para o manipulador. Esses conjuntos são então utilizados como parâmetros de entrada para um algoritmo que emprega uma técnica de busca heurística para otimização, conhecida como Recozimento Simulado (Simulated Annealing). Um algoritmo para otimizar a configuração, com base em algoritmos genéticos (AG), é visto em [7]. Este algoritmo utiliza os requisitos do manipulador, classificados

com base na sua importância, e a precisão do efetuador para avaliar a configuração. Já em [8] é apresentado um método de síntese de configuração que leva em consideração a detecção de colisão e o planejamento de trajetória.

Neste trabalho abordaremos o problema de determinar a configuração, porém com uma abordagem diferente. Ao invés de otimizá-la, iremos escolher uma das configuração predefinidas da lista, passada pelo usuário, que atinja os pontos desejados. Outra diferença para os trabalhos encontrados na literatura é que, devido ao uso de técnicas de visão computacional, não precisaremos que as poses dos pontos sejam repassadas para o algoritmo a cada nova tarefa, pois com a utilização de marcadores fiduciais o próprio algoritmo identifica estas e realiza os cálculos para decidir a configuração. Por mais que o trabalho pretenda ser aplicado a manipuladores modulares, os testes foram realizados com manipuladores comerciais conhecidos, por conta de que estes podem ter suas características, como parâmetros de Denavit-Hartenberg Modificados e espaço de trabalho, comparadas a documentação oficial. Mas isto não é um problema, já que, com o uso da lista de robôs, é possível predefinir configurações de manipuladores a partir dos módulos e adicioná-los à lista.

Visto que o curso de Engenharia de Controle e Automação tem um grande foco em novas tecnologias e tendências de automação do mercado, este trabalho contribuirá com este movimento da indústria em busca de uma maior modularidade da sua linha de produção, através de células de manufatura flexíveis, e com o interesse da sociedade em explorar ambientes não estruturados de forma autônoma. No contexto apresentado, as habilidades adquiridas nas disciplinas de robótica industrial, mecanismos, programação e visão computacional são relevantes.

1.1 Objetivo Geral

O objetivo geral deste trabalho é, desenvolver um algoritmo que utiliza visão computacional para selecionar uma configuração de manipulador predefinida que atinge todas as poses dos objetos (representados pelos marcadores fiduciais). Desta forma, o programa final busca apoiar a decisão técnica da configuração que se adéqua aos pontos a serem alcançados.

1.1.1 Objetivos Específicos

Para se atingir o objetivo geral, o trabalho deve alcançar os seguintes objetivos específicos:

- Identificar os marcadores fiduciais na imagem capturada pelo usuário e estimar a posição e orientação de cada um;

- Selecionar a primeira configuração predefinida de manipulador da lista, com base nos pontos estabelecidos;
- Retornar o manipulador selecionado, seus parâmetros de Denavit-Hartenberg Modificado e o valor dos ângulos das juntas para atingir cada ponto.

1.2 Estrutura do Documento

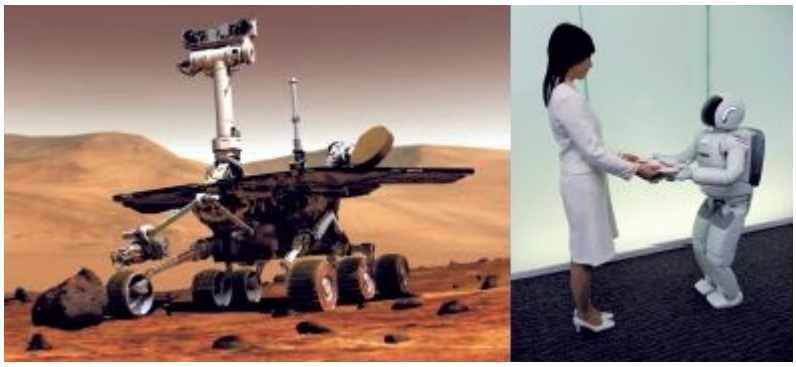
Esta monografia está divida em seis seções. Na Seção 2 é apresentada a revisão da literatura em conjunto com a fundamentação teórica. A Seção 3 contém os métodos e os passos seguidos para a resolução do problema, bem como as decisões que foram tomadas e os porquês destas. Por últimos temos as Seções 4 e 5 que cobrem os resultados, discussão e conclusões.

2 Revisão de Literatura

2.1 Robótica

O termo robô foi cunhado em uma peça de ficção científica Tcheca de 1921 "Rossum's Universal Robots", de Karel Čapek. Os robôs eram pessoas artificiais ou andróides e a palavra, em Tcheco, é derivada da palavra escravo (CORKE, 2017). Atualmente a ideia de robô é um pouco diferente, ainda podemos os ver como escravos, mas não são necessariamente pessoas artificiais, pois hoje os robôs assumem diversas formas, como podemos ver na Figura 1.

Figura 1 – Diferentes tipos de robôs encontrados atualmente



Fonte - (CORKE, 2017)

Robótica pode ser entendida como a área que se propõe ao estudo de robôs, desde seu design até a programação e controle. Então esta se mostra como uma área interdisciplinar, por envolver conhecimentos principalmente de mecânica, elétrica, programação e controle.

2.1.1 Robótica Industrial

Robótica industrial pode ser descrita como a utilização de robôs para aplicações de automação para indústria. Porém, há uma certa dificuldade em definir o que são robôs industriais, por conta da dificuldade de se estabelecer os limites dessa definição. Logo, podemos entender que se um dispositivo mecânico pode ser programado para realizar uma ampla variedade de aplicações, provavelmente é um robô industrial (CRAIG, 2012).

Esta dificuldade surge por conta da diferença conceptual existente entre o mercado Japonês e o mercado Euro-Americano (Ocidental). Para os japoneses um Robô Industrial é qualquer dispositivo mecânico dotado de articulações móveis e destinado ao manuseio de peças (Pick and Place). Já para os ocidentais, a definição de um Robô Industrial é

mais restritiva, sobretudo nos aspectos de controle e reprogramação. Algumas definições clássicas pode ser citadas:

RIA: Segundo a Robotic Industries Association (RIA) um robô industrial é definido como: "Um manipulador multifuncional re-programável projetado para manipular materiais, peças, ferramentas ou peças especiais, através de diversos movimentos programados, para o desempenho de uma variedade de tarefas". A ISO (International Organization for Standardization) define o termo robô como sendo: "uma máquina manipuladora com vários graus de liberdade controlada automaticamente, re-programável, multifuncional, que pode ter base fixa ou móvel para utilização em aplicação em automação industrial". A clássica definição da "Robotic Industries Association (RIA)"americana foi substituída pela da norma ISO 8373 da IFR. Segundo esta norma, "um robô industrial é um manipulador programável em três ou mais eixos, controlado automaticamente, re-programável e multifuncional, que pode estar afixado em um local ou ser móvel, e cuja finalidade é a utilização em aplicações de automação industrial". Existe ainda mais uma definição que é aceita de forma mais generalizada, formulada pela IFToMM: "Um robô é um sistema mecânico sob controle automático e que executa operações tais como manipulação e locomoção".

2.1.2 Manipulador Robótico

Manipulador robótico é, provavelmente, a forma mais importante de robôs industriais que podemos encontrar, principalmente, devido a sua capacidade de realizar diversas tarefas.

Podemos entender manipuladores robóticos como sendo constituídos de elos rígidos, que são conectados por juntas que permitem o movimento relativo dos elos vizinhos. No caso de juntas rotativas ou de rotação, esses deslocamentos são chamados de ângulos de junta. Alguns manipuladores contêm juntas deslizantes (ou prismáticas), nas quais o deslocamento relativo entre os elos é uma translação (CRAIG, 2012).

2.1.2.1 Graus de Liberdade

Manipuladores robóticos possuem diversos atributos, e um dos mais importante é o Grau de Liberdade (GdL). Este é definido como o número de variáveis de posição independentes que devem ser especificadas para localizar todas as partes do robô [9].

Esta propriedade é muito importante, pois cada tarefa pode exigir diferentes Graus de Liberdade. Por exemplo, uma tarefa que se preocupa somente com a posição final do efetuador exige menos GdL do que uma tarefa que se preocupa com a posição e a orientação.

Os manipuladores que iremos focar neste trabalho são de cadeia cinemática aberta e com juntas cuja conectividade é igual a um, e isso acarreta numa propriedade em que o número de GdL é igual ao número de juntas [10].

2.1.2.2 Espaço de trabalho

O espaço de trabalho é um dos primeiros requisitos a serem determinados no projeto do manipulador e depende da tarefa a ser realizada. As demais especificações projetivas decorrem deste requisito.

De maneira mais geral, o espaço de trabalho de um manipulador robótico é o volume total varrido pelo efetuador final à medida que o manipulador executa todos os movimentos possíveis. O espaço de trabalho é determinado pela geometria do manipulador e pelos limites dos movimentos da junta (SICILIANO; KHATIB, 2008).

Esse espaço de trabalho pode ser definido em dois: Alcançável e Hábil. O espaço de trabalho alcançável é o espaço onde o efetuador final consegue atingir os pontos com pelo menos uma orientação. Já o espaço de trabalho hábil são os pontos alcançáveis em todas as orientações possíveis [9] [11].

2.1.2.3 Cinemática

A cinemática de um mecanismo é o estudo do movimento relativo entre os vários elos de um mecanismo ou máquina, negligenciando os efeitos da inércia e as forças que causam o movimento. Ao estudar a cinemática de um mecanismo, o movimento de um elo é frequentemente medido em relação a um elo fixo ou um referencial, que pode não estar necessariamente em repouso (TSAI, 2001).

2.1.2.3.1 Cinemática Direta

O movimento das juntas de um manipulador altera o ângulo e a posição de um elo em relação ao seu elo vizinho. Por conta dos manipuladores industriais possuírem diversas juntas, a função que define a posição e orientação do efetuador é complexa e depende do estado de cada uma destas. Para resolver essa função complexa utilizamos a área da cinemática (SICILIANO; KHATIB, 2008).

A cinemática direta trata da transformação das variáveis do espaço de juntas em variáveis de espaço de trabalho, como posição e orientação. Essa transformação é feita utilizando a matriz de transformação, que obtemos por meio dos parâmetros de Denavit-Hartenberg.

2.1.2.3.2 Cinemática Inversa

Normalmente, estamos interessados em encontrar os valores das juntas para uma determinada posição e orientação de um trabalho a ser realizado. Então o que queremos é executar o trabalho oposto à cinemática direta, isto é encontrar as variáveis no espaço de juntas com base nas variáveis do espaço de trabalho, o que é conhecido como cinemática inversa.

Para manipuladores seriais essa operação possui muito mais dificuldades para encontrar uma solução do que a cinemática direta, e isso se da por conta de fatores como: equações não lineares; solução não é admissível do ponto de vista estrutural do manipulador; podem existir múltiplas soluções (algumas vezes infinitas) para manipuladores redundantes (SICILIANO; KHATIB, 2008).

Nos casos em que existem soluções, elas geralmente não podem ser apresentadas em forma de expressões analíticas explícitas (forma fechada), portanto, métodos numéricos são necessários (forma aberta).

2.1.2.3.3 Métodos Numéricos

Uma forma de lidar com as equações não lineares da cinemática inversa é utilizando métodos numéricos, onde um dos modos muito utilizado é a otimização numérica. Esta pode ser definida como "a minimização ou maximização de uma função sujeita a restrições em suas variáveis" (NOCEDAL, 2006) . A otimização utiliza, para resolver o problema, a cinemática direta, minimizando a função erro que corresponde a diferença entre o ponto desejado e o ponto atual, mediante o ajuste de variáveis das juntas. Assim que a localização pretendida for alcançada, a função retorna os valores das juntas que resultaram nesta.

2.1.2.4 Método de Denavit-Hartenberg

Para que possamos começar o estudo da cinemática do manipulador precisamos descrever a sua geometria, e para isto utilizamos os parâmetros de Denavit-Hartenberg (DH). Este método foi proposto em 1955 por Denavit e Hartenberg [13] e se baseia em quatro parâmetros, que podem ser vistos na Tabela 1, para definir a posição espacial de uma junta em relação a sua junta vizinha.

Tabela 1 – Parâmetros para o método de DH

θi	Ângulo da junta	Ângulo entre x_{i-1} e x_i em torno de z_{i-1}	Variável (revolução)
di	Offset	Distância entre O_{i-1} e O_i ao longo de z_{i-1}	Variável (prismáticas)
ai	Comprimento do elo	Distância entre z_{i-1} e z_i ao longo de x_i	Constante
αi	Torção do elo	Ângulo entre z_{i-1} e z_i em torno de x_i	Constante

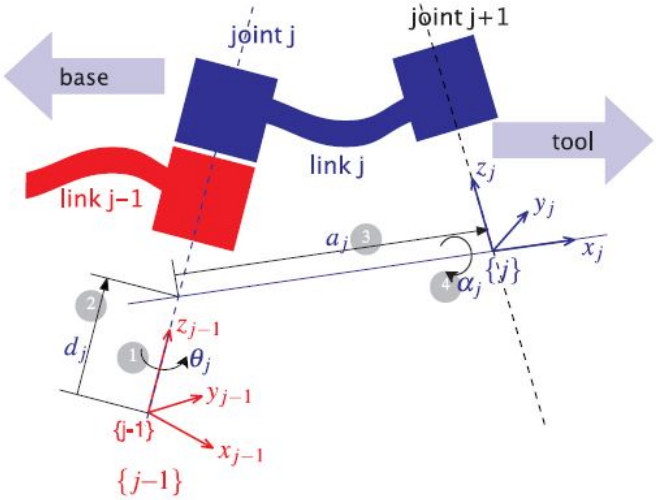
Fonte - Elaborado pelo autor com base em (CRAIG, 2012)

Através destes parâmetros podemos encontrar a matriz de transformação de uma junta em relação a junta vizinha. Esta é encontrada por meio da Equação (2.1).

$$A_i^{i-1} = Rot(Z_{i-1}, \theta_i)Trans(Z_{i-1}, d_i)Trans(X_i, a_i)Rot(X_i, \alpha_i) \tag{2.1}$$

Atualmente este método é amplamente utilizado e estudado, sendo possível encontrar referencias em [9] [11] [14].

Figura 2 – Definição dos Parâmetros de Denavit-Hartenberg



Fonte - (CORKE, 2017)

Na Figura 2 podemos ver a forma como é definido os referenciais de cada junta por meio deste método, bem a representação de cada parâmetro.

2.1.2.4.1 Denavit-Hartenberg Modificado

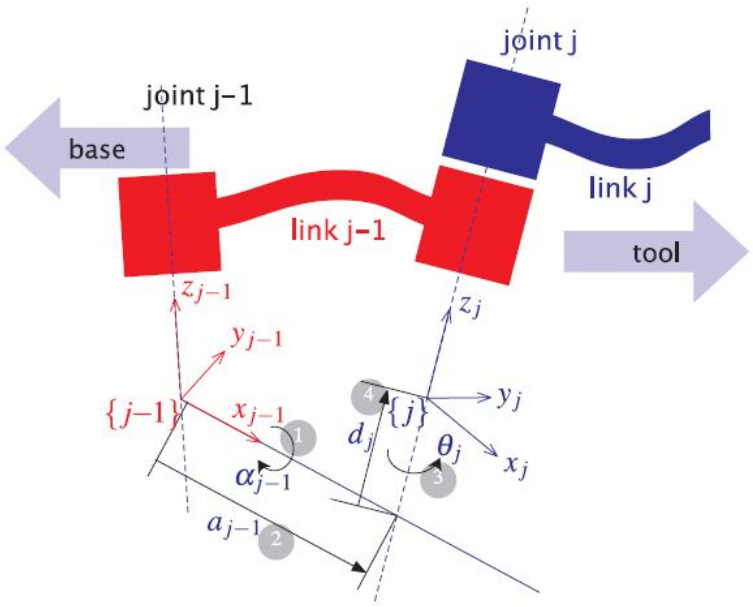
Craig (1986) introduziu pela primeira vez os parâmetros Denavit-Hartenberg Modificados (DHM), onde as coordenadas dos elos são anexadas na extremidade mais próximo da base, ao invés da extremidade mais distante (CORKE, 2017).

Por conta desta mudança nas coordenadas, a ordem das operações e os parâmetros utilizados no cálculo da matriz de transformação são diferentes, como pode ser visto na Equação (2.2)

$$A_i^{i-1} = Rot(X_i, \alpha_{i-1})Trans(X_i, a_{i-1})Rot(Z_i, \theta_i)Trans(Z_i, d_i) \tag{2.2}$$

Essas diferenças e a relação de um método com o outro pode ser vista em [15].

Figura 3 – Definição dos Parâmetros de Denavit-Hartenberg Modificado



Fonte - (CORKE, 2017)

Na Figura 3 podemos ver a forma como é definido os referenciais de cada junta por meio deste método, bem a representação de cada parâmetro.

2.1.3 Robótica Avançada

2.1.3.1 Robótica Modular

2.1.3.1.1 Modularidade e Reconfiguração

Segundo Yim (1994) a modularidade é definida como a característica de ser construído a partir de um conjunto de componentes padronizados que podem ser arranjados de diversas formas, e a reconfigurabilidade pode ser entendida como a habilidade de rearranjar os componentes físicos do robô.

2.1.3.1.2 Manipulador Modular Reconfigurável

Manipulador modular reconfigurável pode ser entendido como um manipulador robótico montado a partir de juntas e elos padrões em uma das muitas configurações possíveis. Este pode ser reconfigurado para o arranjo que melhor se encaixe a tarefa a ser performada (WURST, 1986). Podemos ver alguns exemplos de robôs modulares nas Figuras 4 e 5.

Figura 4 – Robô Modular desenvolvido em parceria com a NASA

Fonte - (TRACLABS, 2011)

Figura 5 – Robô Modular Molecube

Fonte - (ZYKOV, 2008)

2.2 Visão Computacional

Visão computacional é a ciência que tem como objetivo auxiliar na visão de uma máquina, isso é, fazer com que um computador consiga interpretar imagens e que seja possível tomar decisões com base nesta interpretação (CORKE, 2017).

Esse processo inicia com a entrada de uma imagem no sistema. Com a imagem pode ser realizado o processamento, aplicar filtros e extrair características, e a partir destas, é possível tirar diversas interpretações e assim atuar com base no desejado.

2.2.1 Espaço de Cores

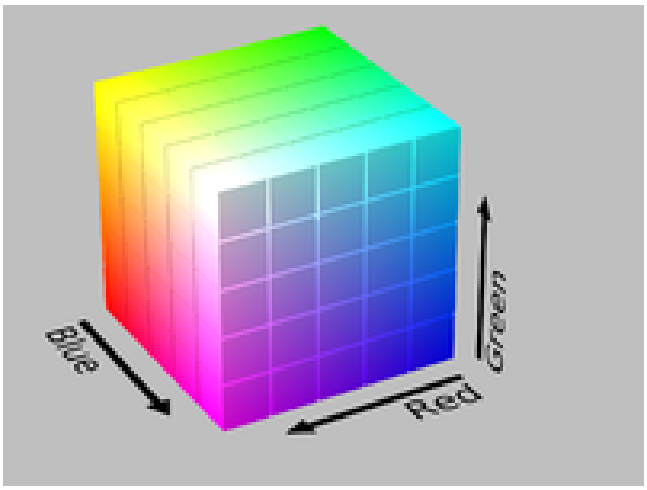
Espaço de cor é um espaço tridimensional que contém todos os valores tristímulus possíveis - todas as cores e todos os níveis de brilho. Como cada cor é um ponto cartesiano neste espaço tridimensional, podemos ter um número infinito destas ao realizar diferentes combinações de valores de coordenadas (CORKE, 2017). Dois espaços de cores muito utilizados são o RGB (Red, Green e Blues) e Escala de Cinza (Greyscale).

2.2.1.1 Espaço RGB

O modelo de cores RGB é baseado na teoria de Young-Helmholtz, ou teoria tricromática. Segundo a teoria o olho humano tem receptores para identificar três cores, que são o vermelho, verde e azul, que ficaram conhecidas como as três cores primarias aditivas. A formação das outras cores pode ser alcançada a partir de combinações feitas pelo cérebro destas três cores.

A representação gráfica do sistema RGB pode ser observada na Figura 6, onde pode ser observado o espaço tridimensional e como as diferentes combinações de vermelho, azul e verde resultam em infinitas cores.

Figura 6 – Representação tridimensional do espaço RGB



Fonte - (WIKIPÉDIA, 2020)

2.2.1.2 Espaço Escala de Cinza

A representação de cores através da escala de cinza se justifica pelo uso de algoritmos de limiarização e identificação de características (features). Para estes algoritmos a informação de cor não é útil, logo uma imagem com três camadas, como em RGB, não traz benefícios (CORKE, 2017).

Como é possível obter muitas informações de uma imagem em escala de cinza, seu uso se torna preferível sobre o RGB, que precisa das três dimensões, pois assim o processamento da imagem fica mais fácil e rápido.

Na Figura 7 podemos ver um exemplo de imagem convertida para escala de cinza.

Figura 7 – Imagem convertida de RGB para Escala de Cinza

Fonte - Elaborado pelo autor

2.2.2 Segmentação de Imagem

A segmentação de imagens é a tarefa de encontrar grupos de pixels que se assemelham. Esta é uma das tarefas mais antigas e mais amplamente estudada da área de visão computacional (SZELISKI, 2010) .

2.2.2.1 Limiarização

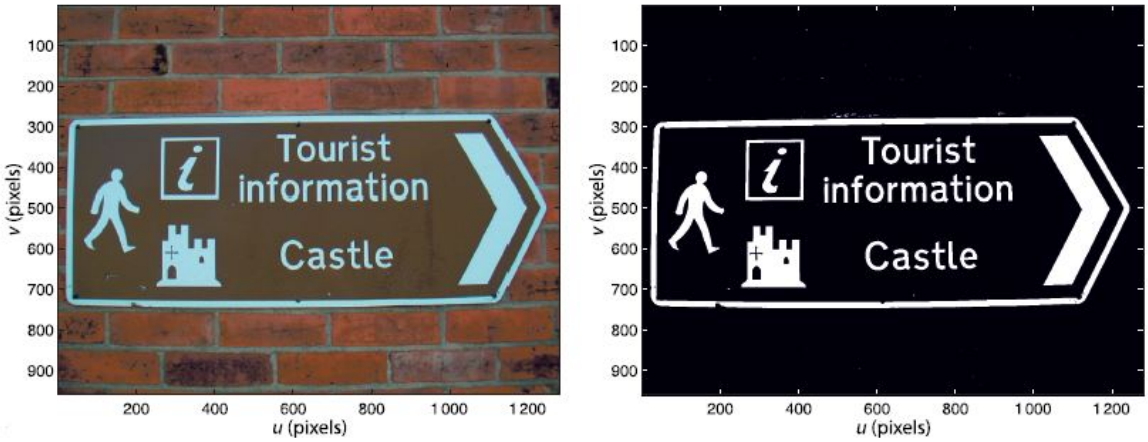
A forma mais simples de segmentação é a limiarização. Neste método escolhemos um valor limiar para os pixels, assim estes são agrupados com base em se o seu valor está acima ou abaixo do limiar (SZELISKI, 2010).

No final obtemos uma imagem binária, onde um grupo de pixels estará com o valor 1 e o outro grupo com o valor 0. Com esses grupos bem separados podemos segmentar a imagem em o que é o objeto, que desejamos, e o que não é o objeto.

2.2.3 Estimação de Pose

Um problema na área de visão computacional é o de estimar a pose de um sistema de coordenadas de um objeto alvo em relação à câmera. Para isto, a geometria do alvo é conhecida, ou seja, sabemos a posição de uma série de pontos no objeto em relação ao referencial do seu sistema de coordenadas. Os parâmetros intrínsecos da câmera, como centro óptico, distância focal e distorção radial, também são conhecidos. Uma imagem é

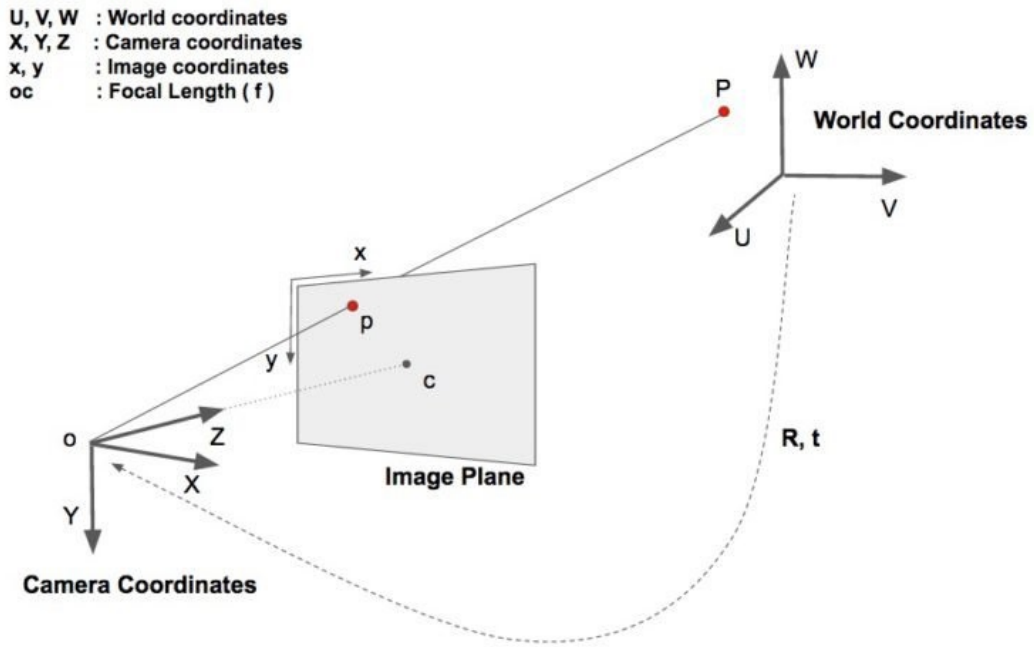
Figura 8 – Exemplo de imagem limiarizada



Fonte - (CORKE, 2017)

capturada e as coordenadas do plano da imagem correspondentes aos pontos 3D do objeto são determinadas usando algoritmos de detecção de características (features). Esta tarefa de estimar a pose usando as coordenadas 2D da imagem, coordenadas 3D do objeto e parâmetros intrínsecos da câmera é conhecido como Perspective-n-Point (CORKE, 2017).

Figura 9 – Representação da estimação de pose com base nos parâmetros intrínsecos, coordenadas 2D e coordenadas 3D dos pontos do objeto



Fonte - (MALLICK, 2016)

O interesse nesse problema vem das suas diversas aplicações, principalmente nas áreas de realidade aumentada e robótica, como para identificar diferentes robôs em um am-

biente e saber a distância entre eles, saber a pose de um objeto com que o robô deve interagir/evitar ou para reproduzir um objeto 3D em uma cena.

2.2.4 Marcadores Fiduciais

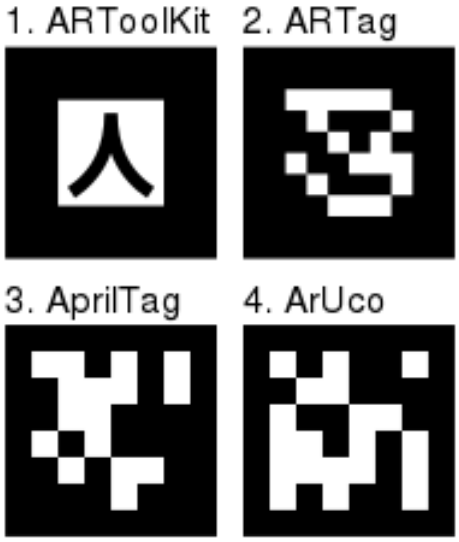
Os marcadores fiduciais são características (features) artificiais projetadas para detecção automática que possuem atributos próprias para torná-los distinguíveis uns dos outros. Embora esses marcadores tenham sido desenvolvidos e popularizados pela primeira vez pela área de realidade aumentada, eles foram amplamente adotados pela comunidade de robótica (WANG; OLSON, 2016). O principal benefício desses marcadores é que um único marcador fornece correspondências suficientes (seus quatro cantos) para obter a pose da câmera, bem como são robustos e funcionam em determinadas oclusões parciais, tronando-os úteis para estimativa de pose ou rastreamento de objetos em aplicações de robótica, como pode ser visto na Figura 10.

Figura 10 – Aplicações de Marcadores Fiduciais

Fonte - (WANG; OLSON, 2016)

Atualmente podemos encontrar diversos tipos de marcadores fiduciais, vistos na Figura 11 , como ARTag, ARToolkit, AprilTag e ArUco. Apesar de muito parecidos, cada um destes marcadores possuem características próprias que acarretam em pontos fortes e fracos.

Figura 11 – Diferentes Marcadores Fiduciais



Fonte - (WIKIPEDIA, 2020)

3 Metodologia

Para realizar o desenvolvimento desse projeto, que visa avaliar a configuração de um manipulador robótico com base na pose dos pontos que este deve alcançar, o trabalho foi dividido em duas principais tarefas: identificar a pose dos marcadores, e calcular a cinemática inversa do manipulador para alcançar cada um destes, e dentro destas tarefas podemos encontrar sub-tarefas que serão apresentadas nesta seção.

3.1 Desenvolvimento em Python

Para iniciar o desenvolvimento do programa tivemos que realizar algumas escolhas iniciais, como da linguagem de programação a ser utilizada e do sistema operacional, já que a decisão de ambas acarretam em diferentes complexidades para o projeto.

A escolha inicial foi a do sistema operacional Windows. Como o trabalho não visa ser aplicado em sistemas embarcados este SO é uma alternativa viável, além de oferecer compatibilidade com diversos softwares e linguagens de programação, e o autor possuir uma maior familiaridade com este.

A decisão da linguagem de programação ficou, principalmente, entre Python e Matlab, e se apoiou em alguns pontos que podem ser visto na Tabela 2. Entendemos que o tarefa mais crítica do desenvolvimento do projeto era a identificação da pose dos pontos a serem alcançados pelo manipulador, e através de pesquisas e de conhecimentos passados foi definida que a melhor forma para tal seria utilizando marcadores fiduciais. Logo, a escolha do Python se deu em virtude da biblioteca de visão computacional OpenCV, já que esta possui funções para utilizar estes marcadores, além de desfrutar de uma ampla comunidade, que auxilia muito na resolução de dúvidas.

Tabela 2 – Comparação das Bibliotecas de Visão Computacional e de Robótica

	Python	Matlab
Função para Calibração da Câmera	X	X
Funções de Processamento de Imagem	X	X
Uso de Marcadores Fiduciais	X	
Definição do Robô através de Parâmetros de DH/MDH	X	X
Cálculo da Cinemática Direta e Inversa	X	X

Fonte - Elaborado pelo autor

3.2 Identificação da Pose dos Pontos

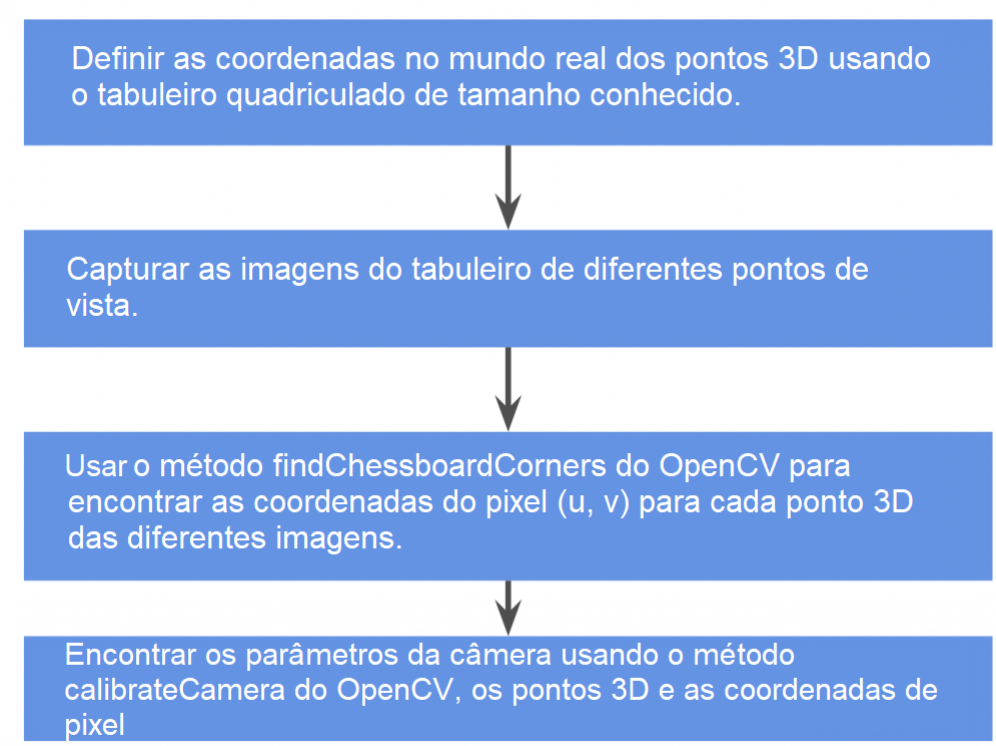
Para esta etapa utilizamos a biblioteca OpenCV [23]. A escolha partiu da significante utilização desta no meio acadêmico e profissional, a ampla base de apoiadores e usuários, e por conta de ser uma biblioteca de código aberto.

3.2.1 Calibração da Câmera

O primeiro passo para a identificação da pose dos marcadores é a calibração da câmera, já que o método de estimativa da pose através destes necessita da matriz dos parâmetros intrínsecos da câmera e dos coeficientes de distorção das lentes.

O processo de calibração segue uma sequência lógica que pode ser vista na Figura 12.

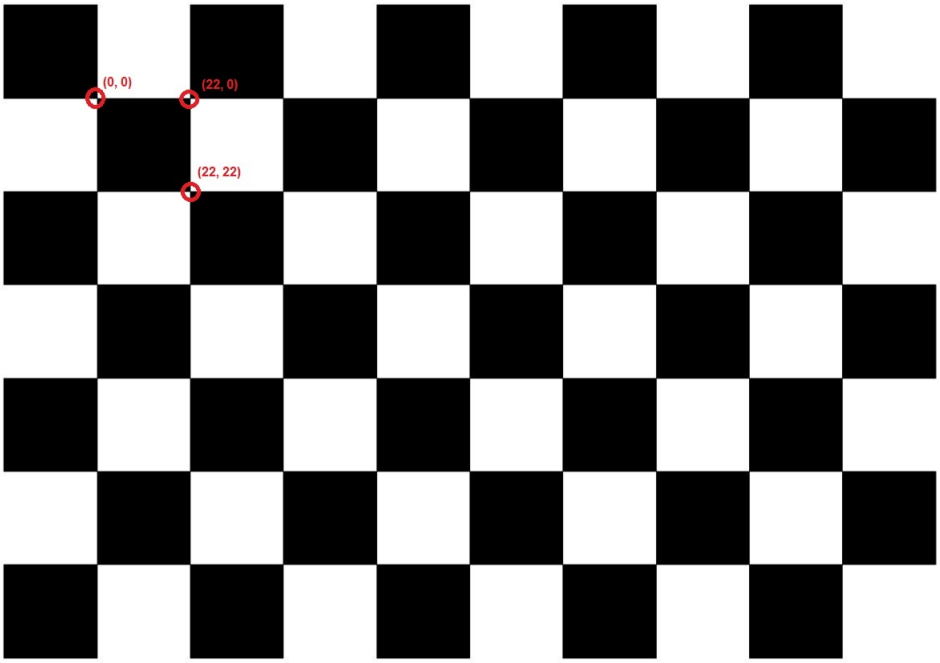
Figura 12 – Sequência de passos para Calibração da Câmera



Fonte - (MALLICK, 2020)

A primeira parte é definir as coordenadas 3D do tabuleiro quadriculado. Para isto escolhemos como referencial do tabuleiro a interseção de dois cantos de quadrados pretos mais a esquerda e mais a cima, e conhecendo o número de colunas e de linhas de quadrados e o tamanho padrão do lado de cada quadrado, podemos definir a posição de cada ponto em relação ao referencial. Na Figura 13 podemos ver um exemplo, onde o tabuleiro possui 10 colunas e 7 linhas, resultando em 9 interseções na horizontal e 6 interseções na vertical, e o tamanho dos lados é de 22 milímetros.

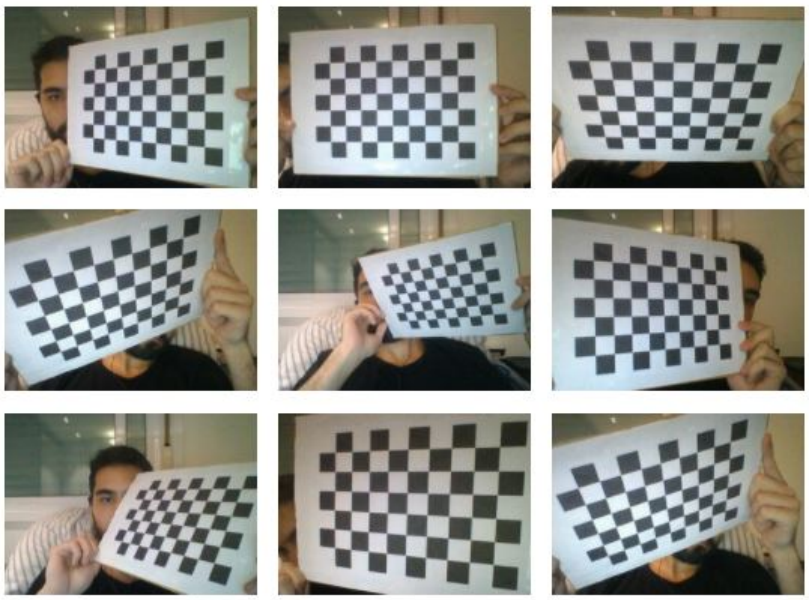
Figura 13 – Exemplo de Tabuleiro para Calibração



Fonte - Elaborado pelo autor

O segundo passo é capturar a imagem d tabuleiro em diferentes pontos de vista. A finalidade deste passo é possuir mais dados e em diferentes configurações para reduzir o erro da estimativa dos parâmetros da câmera.

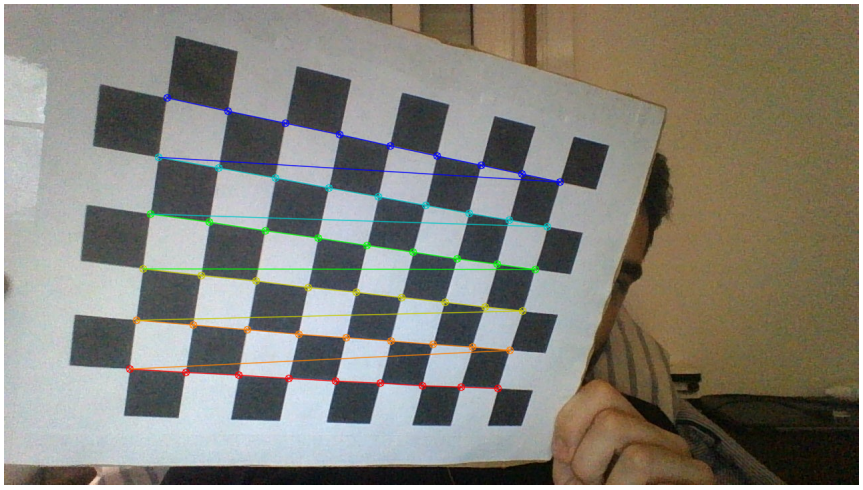
Figura 14 – Exemplo de diferentes Pontos de Vista



Fonte - Elaborado pelo autor

O próximo passo é utilizar a função findChessboardCorners. Esta função identifica as coordenadas em pixel de cada interseção do tabuleiro.

Figura 15 – Resultado da função findChessboardCorners



Fonte - Elaborado pelo autor

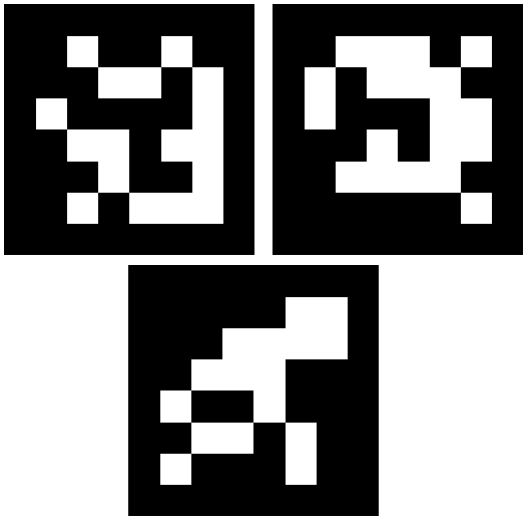
Com os pontos 3D definidos no segundo passo e suas respectivas coordenadas de pixel encontradas no terceiro passo, podemos utilizar a função calibrateCamera, que irá nos retornar a matriz dos parâmetros intrínsecos da câmera e os coeficientes de distorção das lentes. Esta função se baseia no artigo de Zhengyou Zhang [25]

3.2.2 Utilização dos Marcadores Fiduciais

Como já foi explicado anteriormente, a utilização de marcadores fiduciais se da devido a facilidade para determinar sua pose e a robustez deste. A biblioteca OpenCV fornece a utilização dos marcadores ArUco, apresentados no Artigo [26].

A primeira etapa para utiliza-los é gerar o número desejado de marcadores. Isto pode ser feito utilizando a função aruco.drawMarker, que tem como parâmetros de entrada o dicionário de marcadores a ser utilizado e o id do marcador a ser gerado. O id é utilizado para selecionar uma das configuração exclusivas de quadrados brancos e pretos, que geram um código binário reconhecido pela câmera. Esta propriedade é interessante para que possamos saber qual pose está relacionada a cada marcador único.

Figura 16 – Exemplos de diferentes marcadores ArUco gerados

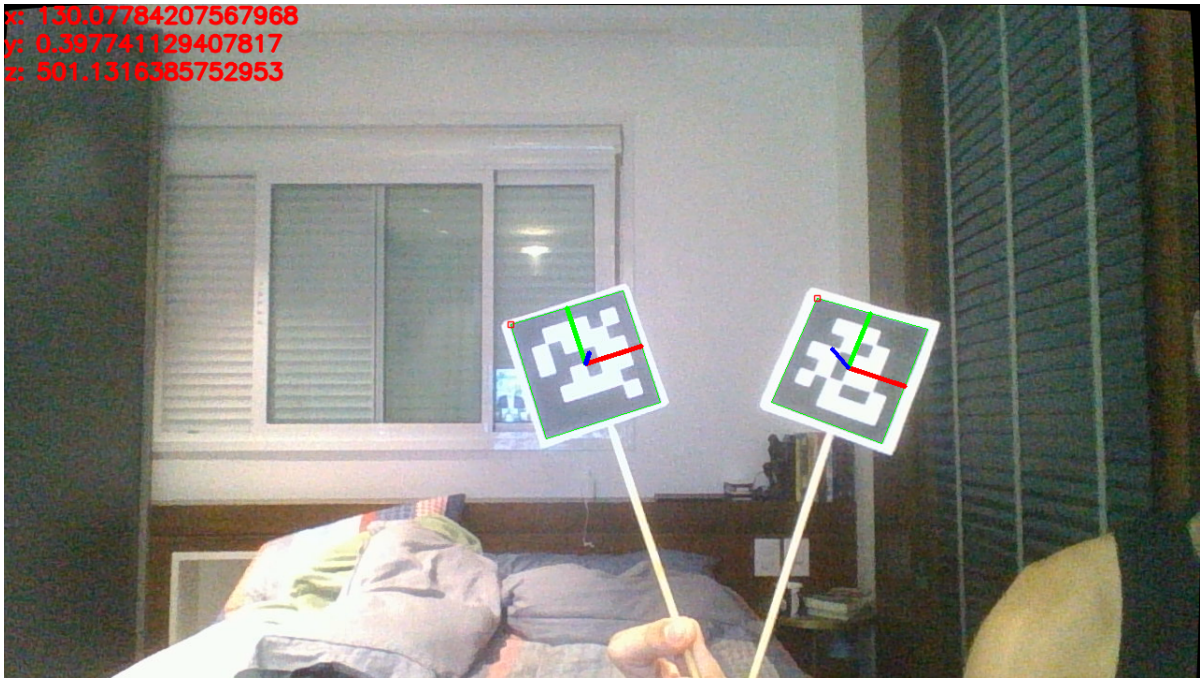


Fonte - Elaborado pelo autor

Com os marcadores gerados e impressos, podemos utilizar as funções para detectar cada um deles na imagem e estimar sua pose. Para realizar tal tarefa necessitamos converter a imagem para escala de cinza, devido a limiarização que a função aruco.detectMarkers utiliza para identificar o padrão do marcador na imagem.

Com o dicionário de marcadores definidos, os marcadores impressos, a imagem da câmera contendo os marcadores em escala de cinza e os parâmetros encontrados na calibração podemos utilizar a função aruco.detectMarkers, que recebe estes como valores de entrada, para detectar os marcadores. Como saída da função obtemos os cantos de cada marcador e seus ids, que então são utilizados na função estimatePoseSingleMarkers, juntamente do tamanho do marcador em milímetros e dos parâmetros resultantes da calibração, para que assim tenhamos a desejada pose de cada marcador. Um exemplo de marcadores identificados e com a pose estimada pode ser visto na Figura 17, onde a pose do marcador pode ser identificada através do eixo de coordenadas desenhando sobre ele.

Figura 17 – Marcadores identificados e pose estimada



Fonte - Elaborado pelo autor

3.3 Cálculo da Cinemática Inversa

Nesta etapa utilizamos a biblioteca Pybotics [27]. A justificativa desta escolha se da devido a ter sido a única biblioteca de robótica de código aberto, que atendia as necessidades do projeto, que foi encontrada.

3.3.1 Definição dos Modelos de Manipuladores

Para iniciar o cálculo da cinemática inversa necessitamos dos modelos dos manipuladores a serem utilizados. A biblioteca de robótica utilizada contém alguns modelos comerciais já definidos, através dos parâmetros de MDH, como o UR10, manipulador da Universal Robots, e o PUMA 560, da Unimation. Os parâmetros de ambos e como são definidos na biblioteca podem ser vistos em 3.1, as referências dos parâmetros para o UR10 pode ser visto em [28] e para o PUMA 560 em [29]

Listing 3.1 – Forma para definir o manipulador com exemplo do UR10 e PUMA 560

```
1 import numpy as np
2
3 def puma560() -> np.ndarray:  # pragma: no cover
4     """Get PUMA560 MDH model. [alfa(i-1), a(i-1), theta(i), d(i)]"""
5     return np.array(
6         [
```

```
            [0, 0, 0, 0],
            [-np.pi / 2, 0, 0, 243.5],
            [0, 431.8, 0, -93.4],
            [np.pi / 2, -20.3, 0, 433.1],
            [-np.pi / 2, 0, 0, 0],
            [np.pi / 2, 0, 0, 0],
        ]
    )


def ur10() -> np.ndarray:  # pragma: no cover
    """Get UR10 MDH model. [alfa(i-1), a(i-1), theta(i), d(i)]"""
    return np.array(
        [
            [0, 0, 0, 128],
            [np.pi / 2, 0, np.pi, 0],
            [0, 612.7, 0, 0],
            [0, 571.6, 0, 163.9],
            [-np.pi / 2, 0, 0, 115.7],
            [np.pi / 2, 0, np.pi, 92.2],
        ]
    )
```

Utilizando este mesmo formato é possível definir o manipulador que for desejado, e deste modo utilizar os métodos implementados pela biblioteca.

3.3.2 Utilização do método de Cinemática Inversa

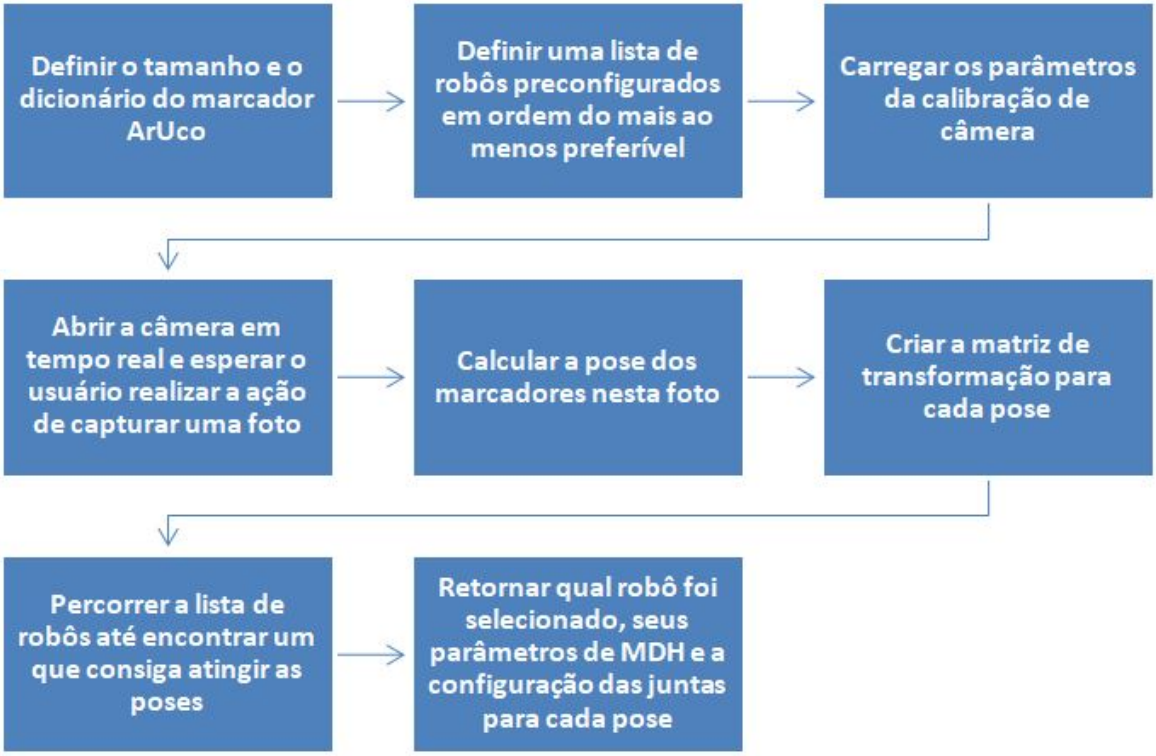
Antes de utilizarmos o método de cálculo da cinemática inversa devemos definir o robô, e isto é feito aplicando o método from_parameters da classe Robot, que tem como função construir o robô a partir dos parâmetros de MDH.

Com o objeto da classe Robot definido, podemos aplicar o método ik, que consiste em calcular a cinemática inversa do ponto informado. O método mencionado utiliza a pose do ponto em formato de matriz de transformação homogênea, isto é, uma matriz com a orientação e posição final do efetuador, e funciona a partir de otimização. Exemplificando, o que o método faz é, a partir de um conjunto de valores de juntas, calcular a cinemática direta e obter a pose do efetuador final, para então avaliar o erro desta pose para a pose desejada. Caso o erro seja maior do que o esperado, a função utiliza novos valores de junta e recalcula o erro, isto ocorre até que seja atingido um valor de erro aceitável ou até um limite de interações.

3.4 Algoritmo

O algoritmo final utiliza estas duas seções apresentadas de forma estruturada e lógica, para que seja possível uma interação com o usuário. Um fluxograma do estrutura do algoritmo pode ser visto na Figura 18.

Figura 18 – Fluxograma do algoritmo



Fonte - Elaborado pelo autor

O primeiro passo, já explicado, é simplesmente carregar o dicionário de marcadores a ser utilizado e definir o tamanho dos marcadores. Em seguida, foi definida uma lista com os manipuladores desejados, onde o primeiro da lista é o preferido para a tarefa e o último é o menos desejado. A escolha da ordem dos robôs na lista fica a cargo do usuário e das suas preferencias em relação aos parâmetros. Para este trabalho a ordem foi escolhida com base no tamanho total do manipulador esticado, isto é, o primeiro é o menor robô em comprimento total e o último é o maior.

Os passos 3 e 4 são, respectivamente, carregar os parâmetros obtido através da calibração da câmera, estes estão salvos em um arquivo de texto, e abrir a câmera em tempo real para que o usuário possa organizar a cena antes de capturar uma imagem do ambiente com os pontos a serem atingidos. A partir desta foto, o programa identifica os marcadores e calcula a pose de cada um em relação a câmera. Como a função de cálculo da pose retorna uma matriz de rotação e um vetor de translação, precisamos montar a

matriz de transformação homogênea, a forma como isto é feito pode ser visto na Figura 19.

Figura 19 – Forma de obter a matriz de transformação homogênea a partir da matriz de rotação e vetor de translação

$$ {}^{A}T_{B} = \begin{pmatrix} {}^{A}R_{B} & {}^{A}t_{B} \\ 0_{1x3} & 1 \end{pmatrix} $$

Fonte - (SICILIANO; KHATIB, 2008)

Na etapa seguinte o algoritmo percorre a lista de configurações de manipuladores para realizar o calculo da cinemática inversa. Em outras palavras, o programa entra em um ciclo de selecionar o próximo robô da lista, calcular a cinemática inversa deste para cada um dos pontos e checar se, para todas as poses, existe uma solução. Caso a resposta seja sim, isto é, existe uma solução possível para o cálculo da cinemática inversa de cada um dos pontos, o algoritmo para de percorrer a lista de manipuladores, e caso seja não, o ciclo continua, até encontrar um manipulador que satisfaça esta condição ou até percorrer toda a lista de manipuladores.

Ao final o algoritmo retorna o nome e os parâmetros de MDH da configuração robótica escolhida, a lista com as matrizes de transformação homogênea de cada um dos pontos e uma lista de ângulos das juntas para atingir as respectivas poses. Um exemplo das informações retornadas pode ser visto na Figura 20.

Figura 20 – Exemplo de informações retornadas

```
Manipulador escolhido: ABB irb120

Parâmetros de MDH [alfa(i-1), a(i-1), theta(i), d(i)]
[  0.          0.          0.        290.        ]
[ -1.57079633   0.         -1.57079633   0.        ]
[  0.        270.          0.          0.        ]
[ -1.57079633  70.          0.        302.        ]
[  1.57079633   0.          0.          0.        ]
[ -1.57079633   0.          3.14159265  72.       ]

Lista de Poses
[array([[   0.71250499,    0.60216946,    0.36017855, -126.52631624],
       [   0.67136036,   -0.73429255,   -0.10044758,   24.9412171 ],
       [   0.20398996,    0.313379  ,   -0.92745981,  516.78675431],
       [   0.        ,    0.        ,    0.        ,    1.        ]])
 array([[   0.86125535,   -0.13450442,    0.49004875, -234.17729038],
       [  -0.01411766,   -0.97029636,   -0.24150708,  -11.9335266 ],
       [   0.        ,    0.        ,    0.        ,    1.        ]])]

Lista de Ângulos das Juntas (rad)
[array([-0.20797838, -1.55105106,  0.58504445, -0.02865351,  2.15408914,
        2.19140841])
 array([-0.02024204, -1.84463673,  0.47071076, -0.34263773,  2.38125314,
        2.88524498])]
```

Fonte - Elaborado pleo autor

4 Resultados e Discussão

À presente seção, caberá apresentar os testes realizados e os resultados obtidos. Estes podem ser divididos em duas principais categorias: testes de visão computacional e testes de robótica.

Nos testes de visão computacional, desejamos avaliar principalmente a robustez dos marcadores fiduciais e a identificação da posição espacial de cada um destes. Além de testar o uso de múltiplos marcadores ao mesmo tempo. Já nos testes de robótica, devido a limitação de não possuirmos os manipuladores reais/fisicamente, a possibilidade e qualidade dos testes a serem realizados é prejudicada, desta forma os experimentos focaram na validação dos manipuladores predefinidos e na recomendação dos diferentes manipuladores.

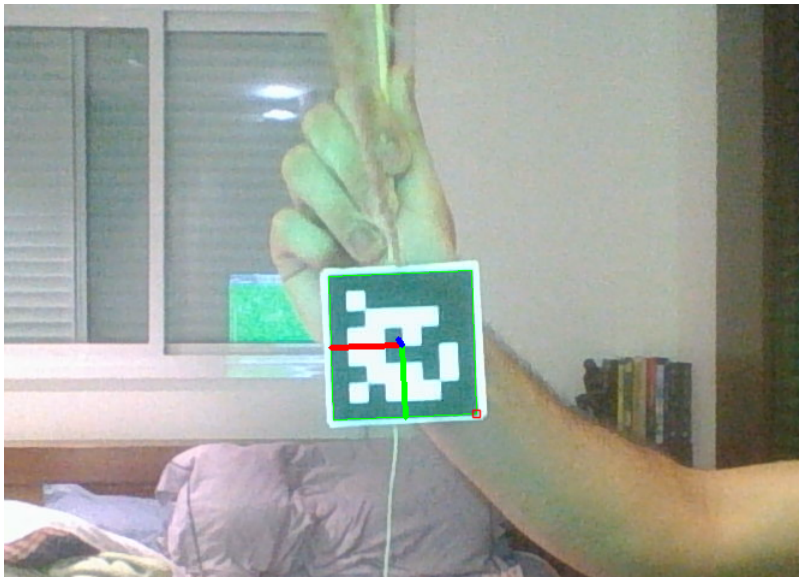
4.1 Experimentos

4.1.1 Identificação dos marcadores em diferentes condições

Este experimento consiste em condicionar os marcadores a duas situações diferentes e avaliar se ainda é possível os identificar na imagem. A primeira é, rotacionar estes em seus próprios eixos x e eixos y, para determinar qual o ângulo de inclinação limite em relação a câmera. E a segunda é realizar a oclusão parcial dos marcadores.

Para a primeira condição utilizamos um fio, preso a câmera e ao marcador, como auxílio para garantir que os eixos z de ambos fossem colineares, como pode ser visto na Figura 21 (onde o eixo z é representado pelo segmento de reta azul).

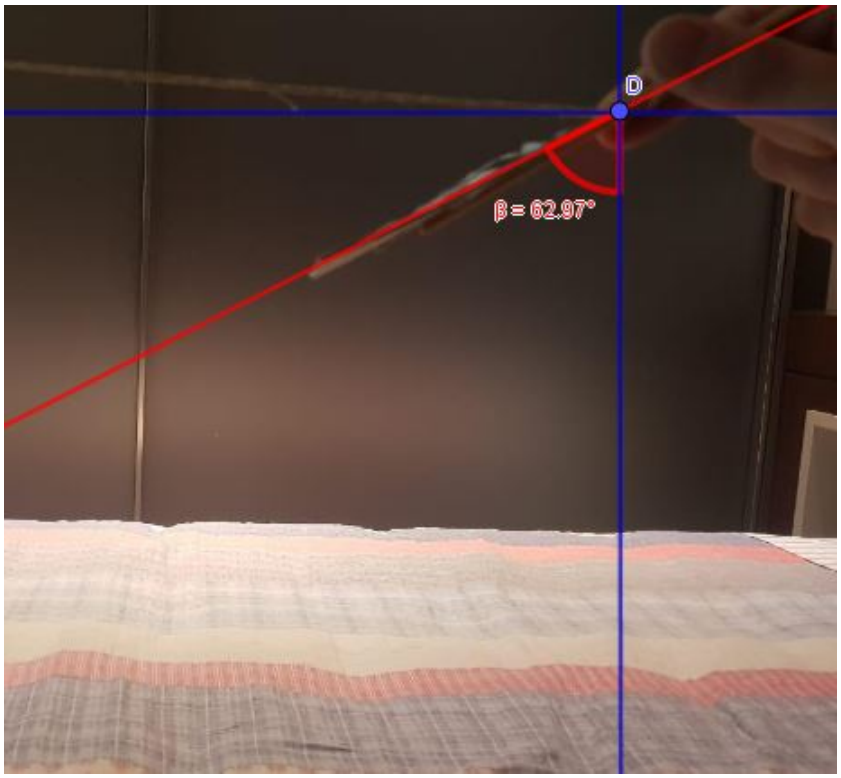
Figura 21 – Posição base com os eixos z colineares



Fonte - Elaborado pelo autor

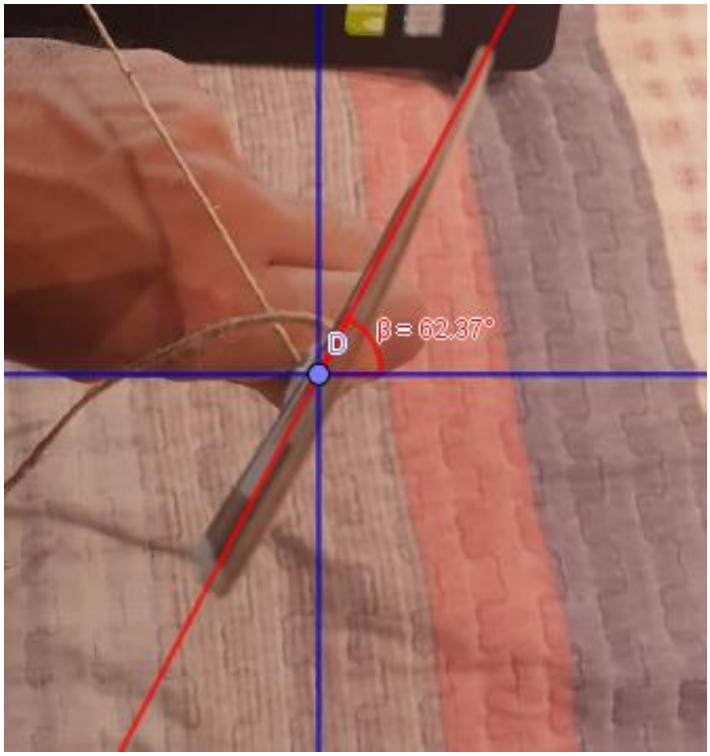
O ângulo foi medido com um transferidor, por ser uma forma mais acessível e barata, e o resultado do ângulo máximo de inclinação em x e em y foi de aproximadamente 63°para ambos, como podem ser vistos na Figura 22 e na Figura 23

Figura 22 – Ângulo máximo de rotação sobre o eixo X



Fonte - Elaborado pelo autor

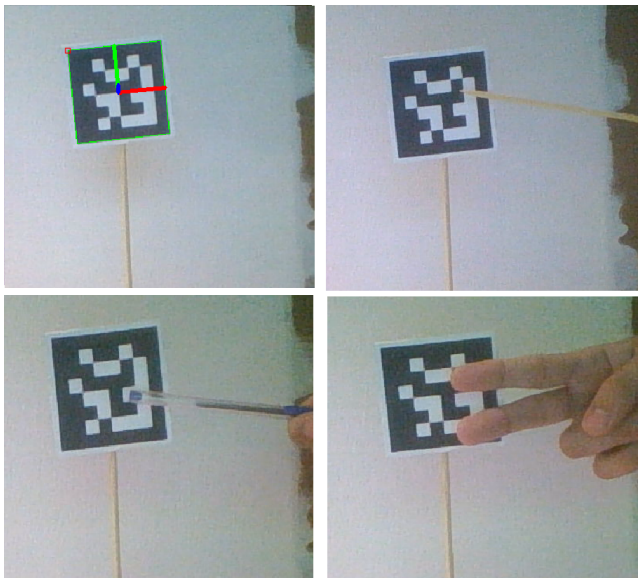
Figura 23 – Ângulo máximo de rotação sobre o eixo Y



Fonte - Elaborado pelo autor

A segunda condição utiliza a mão ou qualquer objeto, como uma caneta, para ocluir o marcador. Na Figura 24 vamos um primeiro estágio, onde o marcador não está ocluso, e é identificado normalmente. Nos outros três estágios, observados na mesma figura, realizamos diferentes níveis de oclusão, resultando na não identificação do marcador.

Figura 24 – Diferentes oclusões e seus resultados



Fonte - Elaborado pelo autor

Como podemos observar nesta seção de teste, o marcador possui um ângulo limite de rotação sobre seus próprios eixos razoável, o que permite que, com uma única câmera possamos cobrir uma angulação de aproximadamente 126 °, tanto ao redor do eixo X como do eixo Y. Para a nossa aplicação, estes valores são adequados já que movimentando a câmera para 4 posições seria possível cobrir grande parte da área útil de trabalho.

Já o teste de oclusão não foi satisfatório, pois com uma pequena oclusão, resultante do uso de um palito fino, já fez com que o marcador não fosse identificado. Apesar dos marcadores ArUco serem considerados robustos a oclusão, estes são quando utilizados diversos marcadores em conjunto e principalmente para aplicações de realidade aumentada, como posto por Garrido-Jurado et al em seu artigo de apresentação destes marcadores [26].

4.1.2 Estimação da posição do marcador

Com os marcadores identificados, o próximo teste visa aferir se a posição do marcador em relação a câmera condiz com os valores calculados. Para tal, medimos a posição em x, y e z do marcador em relação ao referencial da câmera, fazendo uso de uma trena, e confrontamos os valores medidos com os valores gerados pelo algoritmo.

Na Figura 25 podemos ver um exemplo de marcadores identificados e suas coordenadas X, Y e Z em relação à câmera. As coordenadas reais, medidas em milímetros, com o auxilio de uma trena foram, da esquerda para a direita, P1(-186, 90, 534), P2(-28, -47, 686) e P3(244, -27, 443).

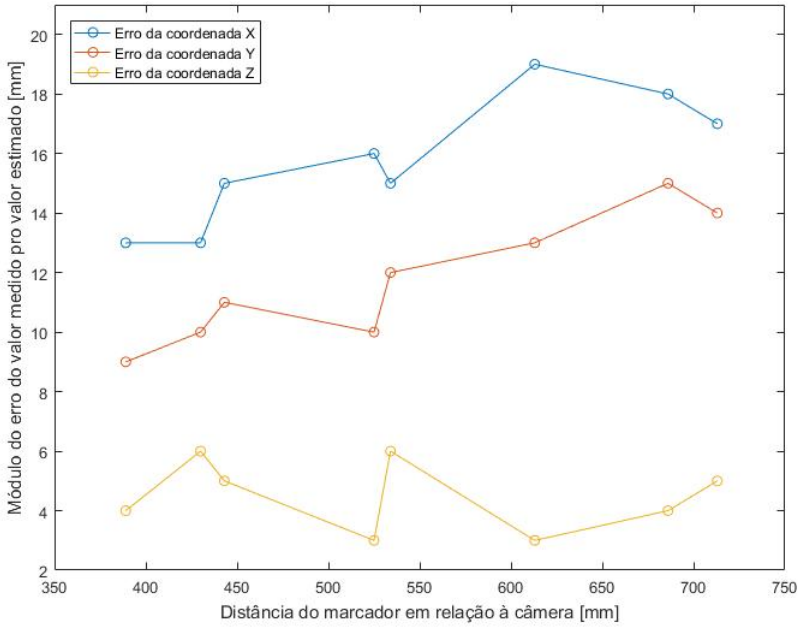
Figura 25 – Exemplo de medição da posição dos marcadores

Fonte - Elaborado pelo autor

Com a estimação e medição de oito pontos, foi possível construir o gráfico visto na Figura 26. Este apresenta o módulo do erro de cada uma das coordenadas de cada ponto em função da distância do ponto, com os valores em milímetros. Como pode ser observado, neste caso, a distância não influenciou no erro, mas esta não causalidade não pode ser

confirmada, já que diversas variáveis que podem influenciar na identificação dos marcadores não foram mantidas constantes, como por exemplo a angulação dos marcadores em relação à câmera e a iluminação. Para uma melhor análise serão necessários mais testes controlados.

Figura 26 – Gráfico do módulo do erro em função da distância do marcador



Fonte - Elaborado pelo autor

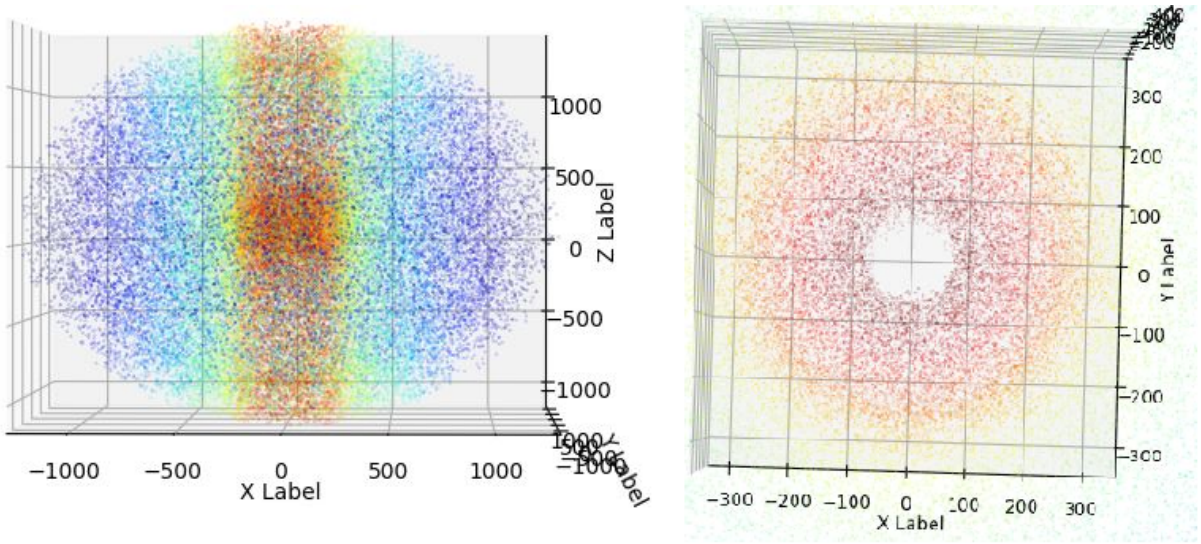
Outro ponto importante a se destacar é que, o erro da posição não passou de 20 milímetros, um valor aceitável para um trabalho desenvolvido num curto espaço de tempo e com equipamentos que não são indicados para usos comerciais, como a câmera do próprio notebook. Porém, este valor de erro pode não ser aceitável para algumas aplicações que necessitam de extrema precisão, logo, são necessários novos testes, com os equipamentos adequados, para que possamos avaliar a real precisão e aplicação deste projeto.

4.1.3 Aferição do espaço de trabalho das configurações predefinidas

Para o primeiro teste da biblioteca de robótica, criamos uma função que plota o espaço de trabalho para um manipulador predefinido (foi explicado anteriormente como predefinir um manipulador na biblioteca Pybotics). Esta função gera n ângulos aleatórios para cada uma das juntas do robô, e após calcula n poses do efetuador final utilizando os ângulos gerados e a função de cinemática direta. Com as coordenadas 3D das poses, plotamos uma nuvem de pontos que representa o espaço de trabalho. Então comparamos este com o espaço de trabalho encontrado na documentação do robô.

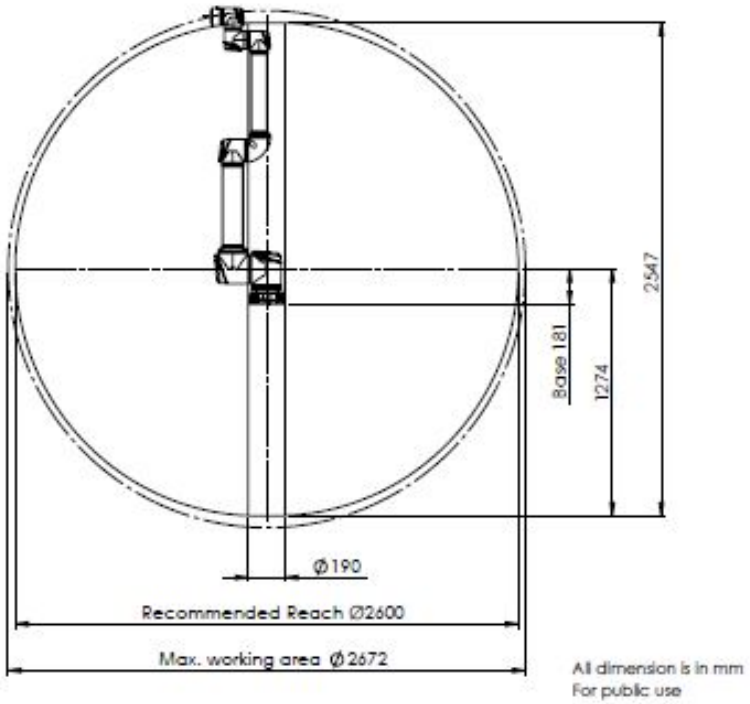
O procedimento descrito foi feito para o manipulador UR10. Podemos ver o espaço de trabalho gerado na Figura 27 e o espaço de trabalho real, obtido na documentação do robô, na Figura 28.

Figura 27 – Espaço de trabalho do UR10 gerado pelo programa



Fonte - Elaborado pelo autor

Figura 28 – Espaço de trabalho do UR10 oficial



Fonte - (UNIVERSAL ROBOTS, 2020

Como podemos constatar, os espaços de trabalho possuem algumas similaridades, sendo: o alcance na vertical é menor que na horizontal; a área interna que não é alcançável possui um diâmetro de aproximadamente 190 milímetros.

Em posse de tais observações podemos afirmar que, a biblioteca de robótica utilizada consegue definir, num grau de proximidade aceitável, um manipulador através dos parâmetros de MDH.

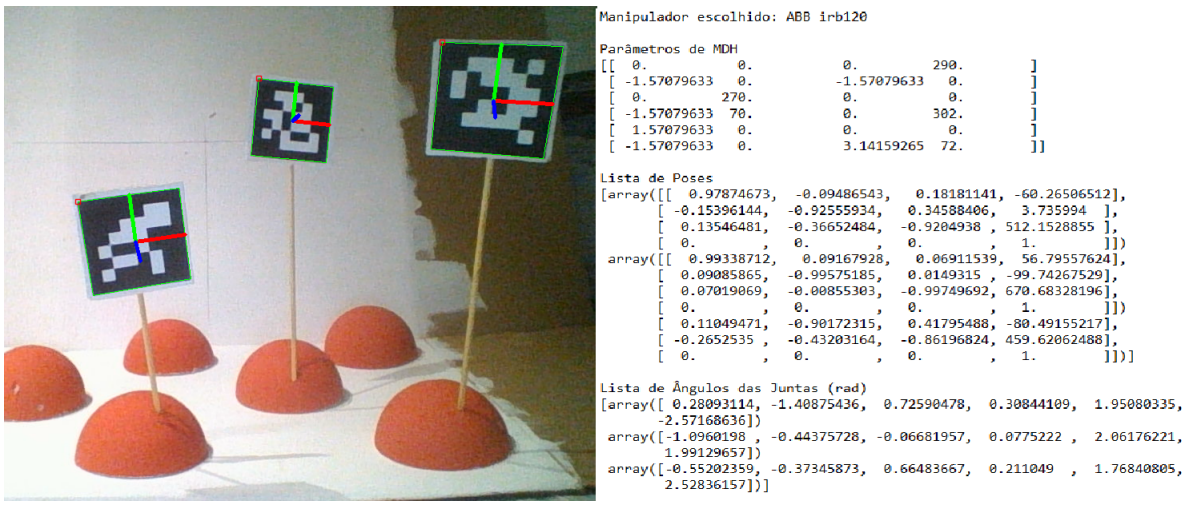
4.1.4 Recomendação de diferentes manipuladores

O último teste serve para identificar se realmente diferentes modelos predefinidos, ou até nenhum modelo, são recomendados para determinados conjuntos de pontos a serem alcançados.

Para este teste definimos diversos conjuntos de pontos, com combinações distintas de poses entre eles, e avaliamos se diferentes manipuladores são recomendados, ou caso nenhum dos manipuladores atinja todos os pontos, o algoritmo não deve sugerir robô, e sim retornar "Manipulador escolhido: Nenhum".

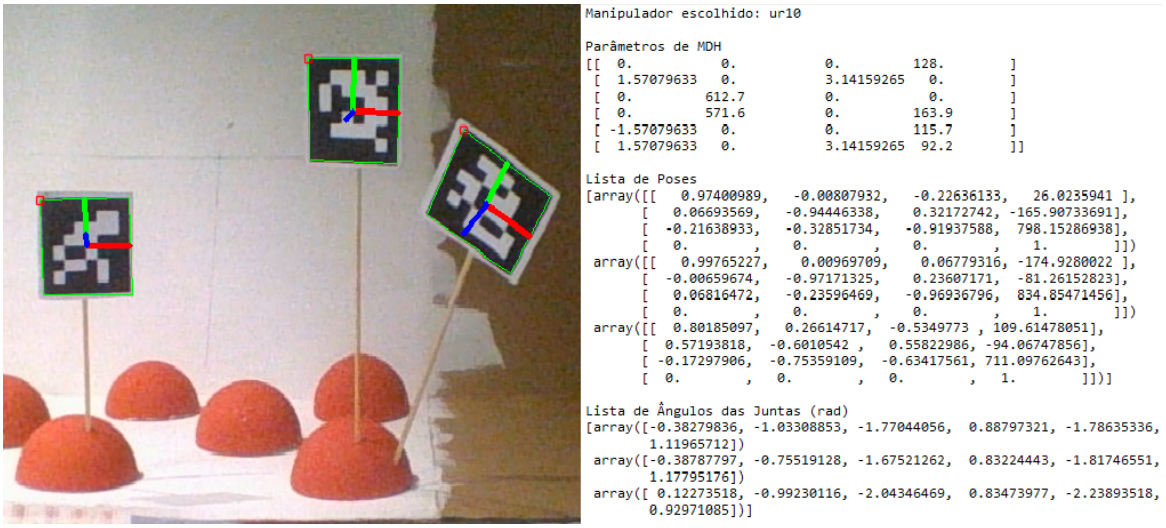
Nas Figuras 29, 30 e 31 podemos ver três exemplos de diferentes arranjos de poses e diferentes manipuladores recomendados.

Figura 29 – Exemplo 1 de arranjo de poses e manipulador recomendado



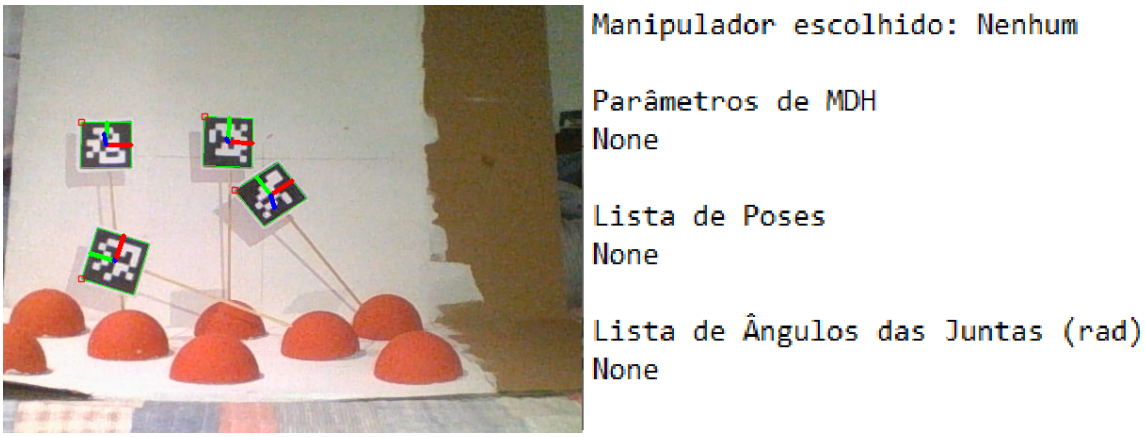
Fonte - Elaborado pelo autor

Figura 30 – Exemplo 2 de arranjo de poses e manipulador recomendado



Fonte - Elaborado pelo autor

Figura 31 – Exemplo 3 de arranjo de poses e manipulador recomendado

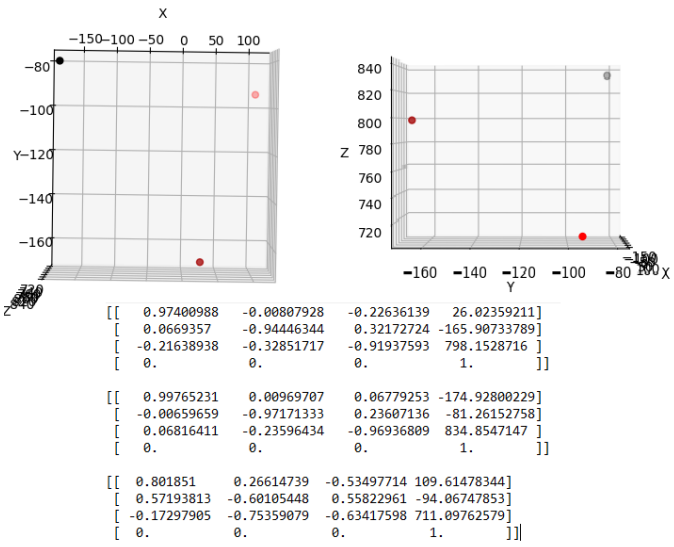


Fonte - Elaborado pelo autor

Para conferir se não ocorreu um erro na recomendação do manipulador, utilizamos os ângulos das juntas retornados pela função para calcular a cinemática direta do manipulador sugerido e avaliar se o ponto resultante é igual ao ponto estimado a partir do marcador.

Utilizando essa abordagem para o exemplo 2, visto na Figura 30, obtemos o gráfico dos pontos e a matriz de transformação de cada um deles, que podem ser vistos na Figura 32.

Figura 32 – Resultado do teste da recomendação do manipulador



Fonte - Elaborado pelo autor

Através deste teste podemos confirmar que o programa realmente recomenda diferentes manipuladores, dependendo da orientação e posição dos pontos a serem atingidos, e estes, provavelmente, atingem os pontos.

Esta conclusão de que o manipulador atinge os pontos ficou limitada, por conta de não possuirmos um destes fisicamente e nem mesmo seu simulador, logo são necessários mais testes para avaliar tão parâmetro e se os resultados condizem com a realidade.

5 Conclusões

O presente trabalho apresentou uma forma de definir a configuração de um manipulador robótico com base em duas entradas: uma imagem com os pontos a serem atingidos; uma lista de manipuladores na ordem de preferência definida pelo usuário.

A decisão final de qual manipulador utilizar para uma tarefa não depende somente da pose dos pontos, já que diversos fatores, não levados em consideração neste projeto, influenciam nesta escolha, como a capacidade de carga ou a velocidade de trabalho. Logo, o que buscamos foi uma forma de auxiliar o usuário numa primeira tomada de decisão do manipulador a ser aplicado, para que ele possa partir de um ponto inicial e busque se a recomendação supre as outras propriedades da tarefa.

Como visto na seção de resultados, o algoritmo conseguiu, a partir dos parâmetros de entrada, identificar os pontos a serem atingidos e recomendar o manipulador preferível para isto. Assim como, o programa já retorna os ângulos das juntas para que o robô escolhido alcance as poses determinadas. Este retorno dos ângulos é importante a fim de se confirmar os resultados obtidos, e, principalmente, para que o usuário possa ser capaz de já colocar o robô para trabalhar.

Por mais que o trabalho tenha utilizado somente robôs comerciais, ele não fica limitado a estes, visto que, para a utilização de robôs modulares basta predefinir a configuração, empregando os parâmetros de MDH, e carregá-las na lista de robôs que é consultada. Assim o projeto se encaixa nas novas tendências de mercados já pontuadas, como modularidade de manufatura e aplicações espaciais, contribuindo para esta área que vem crescendo muito, dentro de um mercado em ligeira expansão.

Para trabalhos futuros, visando um possível mestrado na área, temos como objetivos os seguintes aperfeiçoamentos:

- Realizar os testes discutidos anteriormente, para confirmar os dados obtidos;
- Implementar uma forma de recomendação baseada em mais requisitos de tarefa e requisitos estruturais;
- Otimizar a configuração recomendada dados os requisitos de trabalho definidos.

Com tais aperfeiçoamentos, acreditamos que o algoritmo possa auxiliar diversos usuários na tomada de decisão, ou até automatizar esta, sobre qual a melhor configuração de manipulador para realizar a tarefa.

Referências Bibliográficas

1 AGHILI, F.; PARSA, K. A reconfigurable robot with lockable cylindrical joints. *IEEE Transactions on Robotics*, IEEE, v. 25, n. 4, p. 785–797, 2009. 13

2 YIM, M. et al. Modular reconfigurable robots in space applications. *Autonomous Robots*, Springer, v. 14, n. 2-3, p. 225–237, 2003. 13

3 ROMANISHIN, J. W.; GILPIN, K.; RUS, D. M-blocks: Momentum-driven, magnetic modular robots. In: IEEE. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.], 2013. p. 4288–4295. 13

4 TOLOUEI-RAD, M.; DHULL, A. Design of modular robotic joints for achieving various robot configurations. *International Journal of Mechanical and Mechatronics Engineering*, v. 6, n. 12, p. 2665–2669, 2012. 13

5 YANG, G.; CHEN, I.-M. Task-based optimization of modular robot configurations: minimized degree-of-freedom approach. *Mechanism and machine theory*, Elsevier, v. 35, n. 4, p. 517–540, 2000. 13

6 PAREDIS, C. J.; KHOSLA, P. Synthesis methodology for task based reconfiguration of modular manipulator systems. Carnegie Mellon University, 1993. 13

7 GAO, W. et al. Task-based configuration synthesis for modular robot. In: IEEE. *2012 IEEE International Conference on Mechatronics and Automation*. [S.l.], 2012. p. 789–794. 13

8 ICER, E.; GIUSTI, A.; ALTHOFF, M. A task-driven algorithm for configuration synthesis of modular robots. In: IEEE. *2016 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2016. p. 5203–5209. 14

9 CRAIG, J. J. *Robótica. 3ª edição*. [S.l.]: São Paulo: Editora Pearson, 2012. 17, 18, 20

10 TSAI, L. *Mechanism Design: Enumeration of Kinematic Structures According to Function*. [S.l.]: Taylor & Francis, 2000. (Mechanical and Aerospace Engineering Series). ISBN 9780849309014. 17

11 SICILIANO, B.; KHATIB, O. *Springer Handbook of Robotics*. [S.l.]: Springer Berlin Heidelberg, 2008. ISBN 9783540239574. 18, 20

12 NOCEDAL, J.; WRIGHT, S. *Numerical optimization*. [S.l.]: Springer Science & Business Media, 2006.

13 DENAVIT, J.; HARTENBERG, R. S. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics*, v. 22, p. 215–221, June 1955. 19

14 CORKE, P. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. [S.l.]: Springer Berlin Heidelberg, 2011. (Springer Tracts in Advanced Robotics). ISBN 9783642201431. 20

15 WANG, H. et al. Research on the relationship between classic denavit-hartenberg and modified denavit-hartenberg. In: *2014 Seventh International Symposium on Computational Intelligence and Design*. [S.l.: s.n.], 2014. v. 2, p. 26–29. 20

16 WURST, K. The conception and construction of a modular robot system. In: *Proc. of 16th Int. Conf. Industrial Robots*. [S.l.: s.n.], 1986. p. 37–44.

17 TRACLABS. *Reconfigurable Modular Manipulator (RMM)*. 2011. [Online; acessado em 07-Dezembro-2020]. Disponível em: <https://traclabs.com/projects/rmm/>.

18 ZYKOV, V. et al. Molecubes extended: Diversifying capabilities of open-source modular robotics. In: *IROS-2008 Self-Reconfigurable Robotics Workshop*. [S.l.: s.n.], 2008. p. 22–26.

19 SZELISKI, R. *Computer vision: algorithms and applications*. [S.l.]: Springer Science & Business Media, 2010.

20 MALLICK, S. *Head Pose Estimation using OpenCV and Dlib*. 2016. [Online; acessado em 28-Novembro-2020]. Disponível em: <https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/>.

21 WANG, J.; OLSON, E. Apriltag 2: Efficient and robust fiducial detection. In: IEEE. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2016. p. 4193–4198.

22 WIKIPEDIA. *Fiducial marker*. [Online; acessado em 28-Novembro-2020]. Disponível em: <https://en.wikipedia.org/wiki/Fiducial_marker>.

23 BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 29

24 MALLICK, S. *Camera Calibration using OpenCV*. [Online; acessado em 30-Novembro-2020]. Disponível em: <https://www.learnopencv.com/camera-calibration-using-opencv/>.

25 ZHANG, Z. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 22, n. 11, p. 1330–1334, 2000. 31

26 GARRIDO-JURADO, S. et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, Elsevier, v. 47, n. 6, p. 2280–2292, 2014. 31, 40

27 NADEAU, N. Pybotics: Python toolbox for robotics. *Journal of Open Source Software*, The Open Journal, v. 4, n. 41, p. 1738, set. 2019. Disponível em: <https://doi.org/10.21105/joss.01738>. 33

28 ROBOTS, U. *Parameters for Calculations of Kinematics and Dynamics*. [Online; acessado em 02-Dezembro-2020]. Disponível em: <https://www.universal-robots.com/articles/ur/parameters-for-calculations-of-kinematics-and-dynamics/>. 33

29 CORKE, P. I.; ARMSTRONG-HELOUVRY, B. A search for consensus among model parameters reported for the puma 560 robot. In: IEEE. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. [S.l.], 1994. p. 1608–1613. 33