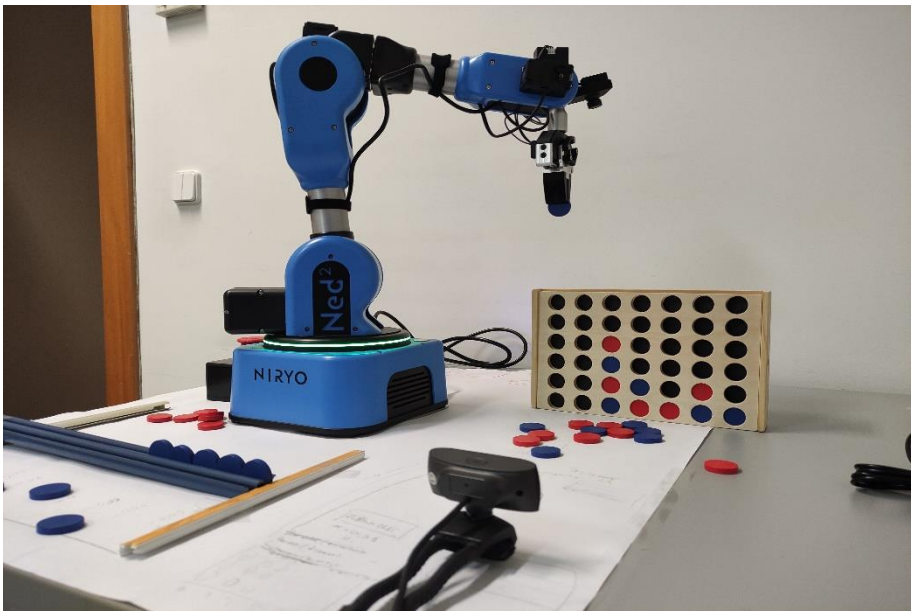
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia Eletrotécnica Energia e Automação



Quatro em Linha com o Robô

GONÇALO PEREIRA GOMES
Licenciado em Engenharia Eletrotécnica

Trabalho Final de Mestrado para obtenção do grau de Mestre
em Engenharia Eletrotécnica – Ramo de Automação e Eletrónica Industrial

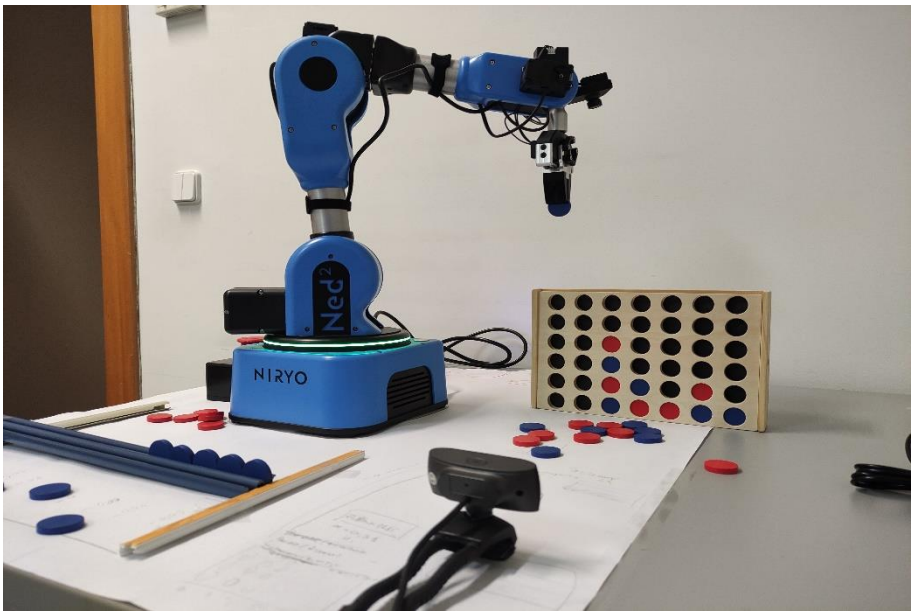
Orientadora:
Prof.ª Doutora Maria da Graça Vieira de Brito Almeida

Júri:
Presidente: Prof. Doutor Luís Manuel dos Santos Redondo
Vogais:
Prof. Doutor Fernando Manuel Fernandes Melício
Prof.ª Doutora Maria da Graça Vieira de Brito Almeida

novembro de 2023

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia Eletrotécnica Energia e Automação



Quatro em Linha com o Robô

GONÇALO PEREIRA GOMES
Licenciado em Engenharia Eletrotécnica

Trabalho Final de Mestrado para obtenção do grau de Mestre
em Engenharia Eletrotécnica – Ramo de Automação e Eletrónica Industrial

Orientadora:
Prof.ª Doutora Maria da Graça Vieira de Brito Almeida

Júri:
Presidente: Prof. Doutor Luís Manuel dos Santos Redondo
Vogais:
Prof. Doutor Fernando Manuel Fernandes Melício
Prof.ª Doutora Maria da Graça Vieira de Brito Almeida

novembro de 2023

Resumo

A robótica tem desempenhado um papel fundamental no avanço da tecnologia e no desenvolvimento da sociedade moderna. Com a capacidade de realizar tarefas complexas e repetitivas de forma autônoma, os robôs têm sido amplamente utilizados numa variedade de aplicações, desde linhas de produção industriais até assistentes pessoais inteligentes. Neste contexto, esta dissertação de mestrado tem como objetivo implementar e desenvolver um sistema que seja capaz de jogar o Quatro em Linha utilizando um manipulador robótico.

O sistema proposto combina técnicas de processamento de imagem, um algoritmo *Minimax* com *Alfa-Beta pruning* e um manipulador robótico para realizar jogadas estratégicas no Quatro em Linha. O processamento de imagem é utilizado para capturar o estado atual do tabuleiro de jogo, permitindo ao sistema obter informações sobre a posição das peças. Com base nessas informações, o algoritmo *Minimax* é aplicado para analisar as possíveis jogadas e escolher a melhor estratégia a ser adotada. O manipulador robótico é responsável por executar as jogadas escolhidas pelo sistema. Assim, é capaz de posicionar as peças no tabuleiro de acordo com as instruções fornecidas pelo algoritmo. Essa integração entre processamento de imagem, algoritmo de tomada de decisão e manipulador robótico permite que o sistema jogue o Quatro em Linha de forma autônoma, rivalizando com jogadores humanos.

Este trabalho de mestrado destaca a importância da robótica e da inteligência artificial na resolução de problemas complexos e na realização de tarefas que requerem análise, planeamento e interações com o ambiente.

Palavras-Chave: Robótica, Quatro em Linha, Manipulador Robótico, Processamento de Imagem, Minimax, Alfa-Beta Pruning, Inteligência Artificial.

Abstract

Robotics has played a fundamental role in advancing technology and shaping modern society. With the ability to perform complex and repetitive tasks autonomously, robots have been widely used in a variety of applications, from industrial production lines to intelligent personal assistants. In this context, the aim of this master's dissertation is to implement and develop a system capable of playing the game Connect Four using a robotic manipulator.

The proposed system combines techniques of image processing, a *Minimax* algorithm with *Alpha-Beta pruning*, and a robotic manipulator to make strategic moves in Connect Four. Image processing is employed to capture the current state of the game board, enabling the system to gather information about the positions of the pieces. Based on this information, the *Minimax* algorithm is applied to analyze the possible moves and choose the best strategy to adopt. The robotic manipulator is responsible for executing the moves selected by the system, positioning the pieces on the board according to the instructions provided by the algorithm. This integration of image processing, decision-making algorithm, and a robotic manipulator allows the system to play Connect Four autonomously, rivaling human players.

This master's work emphasizes the importance of robotics and artificial intelligence in solving complex problems and carrying out tasks that require analysis, planning, and interactions with the environment.

Keywords: Robotics, Connect Four, Robotic Manipulator, Image Processing, Minimax Algorithm, Alpha-Beta Pruning, Artificial Intelligence.

Agradecimentos

Gostaria de expressar os meus sinceros agradecimentos a todos os que contribuíram para a realização desta dissertação de mestrado, tornando possível a conclusão desta jornada acadêmica. Esta conquista não teria sido alcançada sem o apoio, orientação e incentivo de várias pessoas, as quais desejo expressar a minha gratidão.

Em primeiro lugar, agradeço à minha orientadora, a Professora Doutora Maria da Graça Vieira de Brito Almeida, pela dedicação, orientação precisa e incentivo contínuo ao longo de todo o processo de elaboração desta dissetação. As suas valiosas sugestões, conhecimento e experiência foram fundamentais para o sucesso deste trabalho, inspirando-me a superar vários desafios em cada etapa do projeto.

Dedico meus agradecimentos especiais aos meus pais, irmã e restantes familiares, cujo apoio incondicional e amor inabalável sempre me deu a força necessária para seguir em frente e perseguir os meus objetivos académicos e pessoais.

Aos meus amigos e colegas por me acompanharem ao longo desta jornada. As suas palavras de incentivo foram essenciais para que eu me mantivesse perseverante e focado nos momentos de desafio.

Por fim, gostaria de estender meus agradecimentos ao ISEL que forneceu recursos e infraestrutura para a realização deste estudo. Agradeço pela disponibilidade dos equipamentos e materiais que foram essenciais para o desenvolvimento desta dissertação.

A conclusão desta tese de mestrado é a concretização de um esforço coletivo e representa um marco importante na minha vida académica e profissional.

Índice geral

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas, Acrónimos e Siglas

ISO – International Organization for Standardization

RIA – Robot Institute of America

RPY – Roll-Pitch-Yaw

SCARA – Selective Compliance Arm for Robotic Assembly

CCD – Charged-coupled Device

CMOS – Complementary Metal Oxide Semiconductor

A/D – Analógico/Digital

DH – Denavit-Hartenberg

IGM – Inverse Geometric Model

RGB – Red, Green, Blue

HSV – Hue, Saturation, Value

API – Application Programming Interface

URFD – Unified Robotics Description Format

OpenCV – Open Computer Vision

USB – Universal Serial Bus

IA – Inteligência Artificial

Lista de Símbolos e Variáveis

θ_i – Ângulo da junta [rad]

α_i – Ângulo de torção do elo [rad]

a_i – Comprimento do elo [mm]

d_i – Distância de deslocamento do elo [mm]

X – Distância ao longo do eixo x [m]

Y – Distância ao longo do eixo y [m]

Z – Distância ao longo do eixo z [m]

Roll – Ângulo em torno do eixo z da base do manipulador [rad]

Pitch – Ângulo em torno do eixo y da base do manipulador [rad]

Yaw – Ângulo em torno do eixo x da base do manipulador [rad]

1 Introdução

Este Capítulo apresenta uma breve introdução e contextualização da origem da ideia por trás deste trabalho. Os objetivos a serem alcançados são explicitados e, por fim, são fornecidas orientações para a leitura do documento, incluindo um resumo conciso de cada Capítulo.

1.1 Enquadramento e Motivação

A robótica, visa projetar, construir e programar robôs capazes de realizar tarefas de forma autônoma ou em colaboração com humanos. Por outro lado, o processamento de imagem é uma área da ciência da computação que lida com a aquisição, análise e interpretação de imagens digitais. A combinação da robótica com o processamento de imagem oferece um vasto campo de possibilidades, ampliando assim as aplicações da robótica. A capacidade de capturar, processar e interpretar imagens permite que os robôs interajam com o ambiente de maneira mais sofisticada.

A motivação por trás do desenvolvimento desta dissertação reside no desejo de explorar as capacidades dos robôs para realizar tarefas complexas de forma autónoma. Além disso, implementar um sistema que possa jogar o Quatro em Linha com um manipulador robótico representa um desafio técnico significativo, que envolve a integração de várias áreas, como visão artificial, algoritmos para tomar decisões e controlo no domínio da robótica.

Ao implementar este sistema, será possível aprofundar o conhecimento não só sobre as capacidades de manipulação e movimentação de robôs, mas também sobre técnicas de processamento de imagem. Adicionalmente, o desenvolvimento de um sistema autônomo para jogar o Quatro em Linha pode ter aplicações em outros jogos e desafios estratégicos, abrindo portas para avanços futuros na robótica e na inteligência artificial.

De igual modo, foi possível com este trabalho tomar conhecimento com outras ferramentas de programação - *python* em combinação com a biblioteca *OpenCV* – assim

como trabalhar com um manipulador com grande capacidade de exploração devido a possuir características recentes (comunicação por *Wi-Fi*, câmara integrada, etc).

1.2 Objetivos

O objetivo desta dissertação é desenvolver e implementar um sistema que permite jogar entre um humano e um manipulador robótico o jogo “Quatro em Linha”.

Para alcançar estes objetivos, serão realizados estudos teóricos sobre manipuladores robóticos (configurações e modelação cinemática), bem como os princípios das técnicas de processamento de imagem. Também será desenvolvido um algoritmo para o reconhecimento das posições das peças no tabuleiro e para a tomada de decisão sobre os movimentos a serem realizados pelo manipulador.

No final da dissertação, espera-se ter um sistema funcional e eficiente para jogar o jogo "Quatro em Linha", que possa ser utilizado como base para o desenvolvimento de sistemas semelhantes em outras aplicações que envolvam processamento de imagem e manipulação robótica.

1.3 Estrutura do Documento

Este documento está organizado em cinco Capítulos.

O Capítulo 1 introduz as motivações e o enquadramento do tema escolhido, também serve para delinear os objetivos e apresentar a estrutura do documento. No Capítulo 2 é apresentado o estado da arte com temas relevantes para a interpretação desta dissertação. O Capítulo 3 descreve o trabalho realizado, nomeadamente a escolha do manipulador, o estudo da cinemática, o tratamento de imagem e o funcionamento do algoritmo *Minimax*. Os resultados obtidos são expostos no Capítulo 4, realçando as vantagens e desvantagens do sistema. Por fim, no Capítulo 5 são apresentadas as conclusões e algumas considerações para trabalho futuro.

2 Estado da Arte

Neste Capítulo pretende-se falar dos elementos essenciais a esta dissertação e a sua evolução ao longo do tempo. Deste modo, os Capítulos posteriores serão compreendidos com mais facilidade. Assim, os dois grandes temas aqui referidos serão a robótica e o processamento de imagem.

2.1 História da Robótica

A história da robótica é destacada por um mundo de fantasia que proporcionou inspiração para converter essas ideias em realidade. É uma história rica em criatividade cinemática, engenhosidade científica e visões empreendedoras. Surpreendentemente, a definição de robótica é controversa mesmo entre as pessoas que estudam este tema. Por um lado, existe a versão de ficção de um robô tipicamente uma forma humana com características antropomórficas. Por outro lado, encontra-se o robô tradicional de automação industrial. Na ISO 8373 (2021), a *International Organization for Standardization* define robô como um "mecanismo atuador programável com um grau de autonomia para efetuar a locomoção, manipulação ou o posicionamento" [1]. A *Robot Institute of America* (RIA) descreve um robô como "um manipulador multiusos, reprogramável projetado para mover materiais, ferramentas, peças ou equipamentos especializados através de vários movimentos programados para a realização de uma variedade de tarefas [2], [3], [4]. O dicionário *Merriam-Webster* declara uma definição mais inspiradora, onde um robô é "uma máquina que se parece com um ser humano e realiza diversas ações complexas (como andar ou falar)" [2], [4].

Influência da cultura

A mitologia está repleta de representações de seres artificias, desde lendas gregas, hebraicas e inuítes até à criação do *Frankenstein*, existem diversos contos que falam em entidades inanimadas que "ganham vida". Estes contos mitológicos e muitos outros têm

algo em comum: os criadores de seres sobrenaturais, muitas vezes vêm as suas obras virarem-se contra eles, resultando tipicamente em finais trágicos.

O advento do cinema deu vida a muitas destas criaturas míticas, bem como uma fonte infinita de novas criaturas artificiais. Em 1926, o filme de *Fritz Lang*, "*Metropolis*", introduziu o primeiro robô num filme. Em 1977, "*Star Wars*" deu vida a dois dos robôs mais cativantes que já visitaram a tela grande, *R2-D2 e C3PO*. O cinema e a televisão deram vida a estes robôs, que tanto representavam papéis malignos como nobres. Embora apenas seja uma pequena amostra, estas personagens ilustram o fascínio da humanidade por criaturas mecânicas que exibem inteligência que rivaliza, e muitas vezes supera, a dos seus criadores [2].

Invenções que levaram à robótica

A área da robótica evoluiu durante vários anos sem ser mencionada a palavra robô até ao início do século XX. *Joseph Jacquard* (1752-1834) inventou um tear que usava uma série de cartões perfurados para controlar a repetição de padrões usados para tecer carpetes e tecidos. Este sistema foi adaptado mais tarde por *Charles Babbage,* para criar uma calculadora automática, princípio que levou mais tarde à criação dos computadores e da programação. *Christopher Miner Spencer,* desenvolveu um torno, que incluía uma árvore de cames e uma torre que avançava automaticamente e permitiu um nível mais elevado e sofisticado no fabrico de parafusos, tendo deste modo ficado reconhecido por dar nascimento às máquinas aparafusadoras. Em 1892, *Seward Babbitt* apresentou uma grua motorizada que usava uma garra mecânica para remover barras de metal fundidas de uma fornalha, isto aconteceu 70 anos antes do primeiro robô industrial da *General Motors* usado para um propósito semelhante [2].

Nascimento do robô industrial

A palavra robô só surgiu no vocabulário em 1920, quando o checo *Karel Capek* (1890–1938) escreveu "*Rossum's Universal Robots*", uma peça de ficção científica onde o tema retrata trabalhadores futuristas criados por seres humanos, com o propósito de automatizar o trabalho dos mesmos, aliviando assim o seu fardo. Contudo, era necessário

dar um nome a estes trabalhadores e recorreu ao seu irmão mais velho, *Josef*, para obter alguma inspiração. O irmão respondeu com uma palavra que ele cunhou - *robota*. A palavra checa *robotnik* refere-se para um camponês ou servo, enquanto *robota* significa trabalho penoso ou servidão [2] - [5].

Além dos irmãos *Capek, Isaac Asimov (1920-1992)* provou ser outro escritor de ficção científica que teve um profundo impacto na história e criatividade do reino que se tornaria a robótica. Asimov escreveu uma série de contos que envolviam temas de robôs. "*I, Robot*", publicado em 1950, incorporou nove desses contos relacionados numa coleção. Foi nestes contos que *Asimov* introduziu as "Três Leis da Robótica". Apesar dessas três leis terem aparecido em várias obras, não foi até "*Runaround*", publicado em 1942, que elas apareceram juntas e de forma concisa além disso, também é a primeira vez que a palavra robótica é usada, e é entendida como a tecnologia que lida com o projeto, construção, e operação de robôs.

Desta forma, enumera-se as famosas "Três Leis da Robótica" [2], [4], [5]:

- Primeira Lei: Um robô não pode ferir um ser humano ou, por inação, permitir que um ser humano se aleije, a menos que isso viole uma lei de ordem superior.
- Segunda Lei: Um robô deve obedecer às ordens que lhe forem dadas por seres humanos, exceto quando tais ordens entrem em conflito com uma lei de ordem superior.
- Terceira Lei: Um robô deve proteger sua própria existência, desde que esta não entre em conflito com uma lei de ordem superior.

Em 1985 modificou a sua lista para incluir a chamada "Lei Zero":

- Lei Zero: Um robô não pode ferir a humanidade ou, por inação, permitir que a humanidade sofra algum mal.

Após a Segunda Guerra Mundial, os Estados Unidos da América experienciaram um forte impulso industrial, fortalecendo a sua economia. A junção dos avanços na tecnologia e o mundo da ficção científica surgiu a partir da visão de *Joseph Engelberger* e *George Devol*, quando se conheceram em 1956 e tiveram uma conversa em torno robótica, automação, *Asimov*. *Devol*, tinha previamente criado uma patente "*A Programmed Article Transfer*", que a imaginação de *Engelberger* traduziu para "robô". Após este encontro

casual, os dois formaram uma parceria que levou ao nascimento do robô industrial, o *Unimate* [2], [5]. Este robô, retirava peças metálicas, de altas temperaturas, de uma máquina de fundição sob pressão, daí que facilitava um trabalho muito desagradável de realizar manualmente (Figura 1).

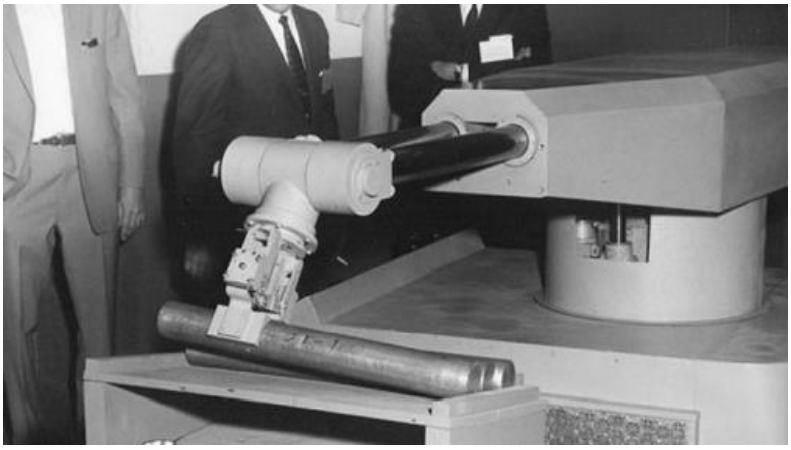


Figura 1 – O primeiro robô industrial "Unimate" [5].

Com efeito, a produção de robôs iniciou no Japão, em 1969, após um acordo entre a *Unimation* (empresa formada por *Engelberger* e *Devol*) e a *Kawasaki*. Por volta de 1973 já existiam 3000 destes robôs a operar globalmente. Também nesse ano, a *Hitachi* tornou-se a primeira empresa a incorporar sensores de visão que permitiam ao robô localizar objetos.

O primeiro robô a ser controlado por um microprocessador, *IRB 6*, foi desenvolvido pela *ASEA* em 1974. Esta máquina, semelhante a um braço humano, podia carregar até 6 kg e era utilizada para polir tubos de aço inoxidável [5].

2.2 Áreas de Aplicação da robótica

Indústria

Como descrito anteriormente, os primeiros robôs foram implementados em aplicações industriais, com o propósito de substituir os trabalhadores humanos na realização de tarefas repetitivas ou perigosas. Atualmente, estes tipos de robôs ou mais especificamente, manipuladores, continuam a realizar as mesmas tarefas, que envolvem

operações de carga e descarga, pintura, soldagem ou montagem (Figura 2). Além disso, estes manipuladores, têm a capacidade de operar sem a presença de humanos. No entanto, a melhor solução é uma colaboração entre humano e robôs, onde os humanos podem, facilmente, reconhecer erros e atuar sobre os mesmos [6].

Figura 2 – Linha de montagem de carros utilizando manipuladores [6].

No início de 2022, a *Boston Dynamics* lançou o *Stretch* (Figura 3), um robô designado para mover caixas em armazéns. O que diferencia este robô dos outros é a sua mobilidade. Geralmente este tipo de manipuladores encontram-se fixos com fluxo de trabalho modelado à volta dos mesmos. Contudo, o *Stretch* possui uma base com rodas e um mastro equipado com camaras e sensores, de modo a poder navegar livremente pelo armazém. Acoplado à base, encontra-se um enorme braço robótico com 7 graus de liberdade e na sua extremidade um conjunto de ventosas que permitem apanhar caixas até 23 kg. Adicionalmente, está equipado com baterias que possibilitam um funcionamento até 8 horas. Como resultado, a empresa estima que o robô consegue mover 800 caixas por hora [7].

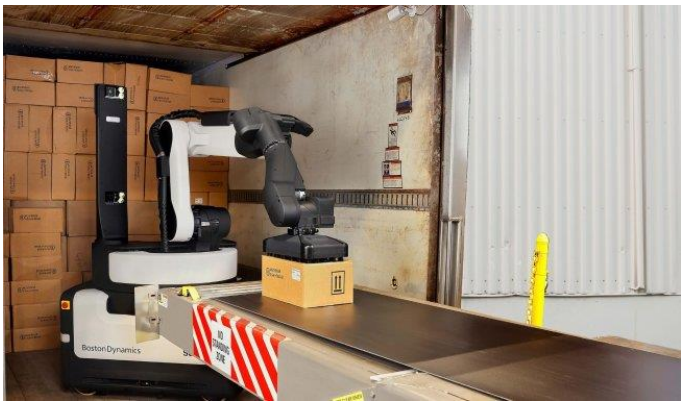


Figura 3 – Robô "Stretch" da Boston Dynamics [7].

Medicina

A robótica também encontrou um caminho para a medicina, principalmente na área cirúrgica. De momento, o objetivo dos robôs neste tipo de aplicações não é substituir o cirurgião, mas oferecer ferramentas de auxílio. Posto isto, podem existir dois tipos de sistemas. No primeiro, o robô é controlado diretamente pelo cirurgião (através de uma consola), permitindo que haja uma maior precisão no movimento. No segundo, o robô funciona como assistente onde pode, por exemplo, segurar em instrumentos e ter interfaces de comando mais diretas, como controlo por voz [8]. A linha sistemas cirúrgicos *da Vinci* da *Intuitive Surgical* é um dos exemplos para o primeiro caso [9]. Estes robôs possuem 7 graus de liberdade e um sistema de visão 3D de alta qualidade que permite ampliar até 10 vezes, o que facilita o trabalho do cirurgião (Figura 4).

Figura 4 – Sistema cirúrgico "da Vinci" [9].

Alternativamente, também se pode usar a robótica na reabilitação. Existem um número de sistemas que foram desenvolvidos para pessoas com dificuldade em efetuar certos movimentos. Para reabilitação de membros superiores (braços), criaram-se quatro estratégias [3]:

- Passiva: o movimento é iniciado e imposto pelo robô;
- Ativa assistida: o utilizador inicia o movimento, mas o robô oferece auxílio ao longo de um caminho predefinido;
- Ativa resistiva: o utilizador inicia o movimento, e o robô oferece resistência;
- Exercício bimanual: movimenta-se o braço que está normal e o robô replica o movimento, através de uma das estratégias anteriores, no braço afetado.

Os sistemas *GENTLE/s* são um exemplo de robôs que usam as três primeiras estratégias mencionadas (Figura 5).

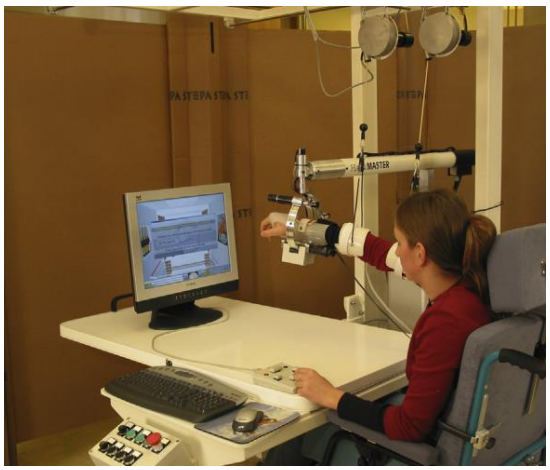


Figura 5 – Sistema GENTLE/s [3].

Exploração Espacial

A introdução da robótica contribuiu bastante para os avanços na exploração espacial. Esta ferramenta permite que os humanos consigam realizar uma variedade de funções no espaço, tal como, manipulação de equipamento e exploração planetária. Ao enviar robôs móveis (*rovers*) para outros planetas, consegue-se recolher dados sobre o terreno e analisar essa informação à distância [2], [8].

O *Mars Perserverance rover* (Figura 6) que está ativo há mais de 2 anos foi desenvolvido para coletar e analisar amostras de minerais do planeta. Portanto, para além de ser um robô móvel tem também um braço robótico (manipulador) com 5 graus de liberdade [10].

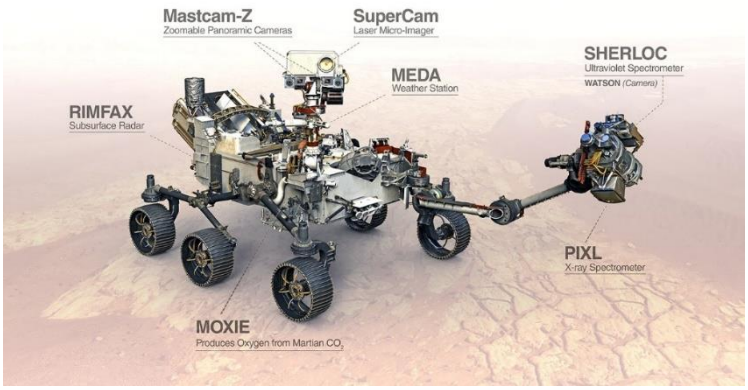


Figura 6 – Mars Preserverance rover [10].

2.3 Configuração de um robô

Todos os sistemas robóticos são constituídos por dois componentes principais. Em primeiro lugar, existem as partes móveis do robô, em segundo lugar existe o sistema de controlo [4]. Além disso, os robôs podem ser divididos em três classes principais: manipuladores, que imitam o braço humano, robôs andantes, que replicam a locomoção de seres humanos ou animais, e robôs móveis que tem a aparência de carros ou outros veículos.

Existem, ainda dois termos que são usados para realçar a flexibilidade intrínseca de um robô, a adaptabilidade e a versatilidade. A adaptabilidade significa que o robô é capaz de ajustar os movimentos, quer quando sujeito a mudanças no ambiente de trabalho, quer devido a alterações resultantes da execução de tarefas. A versatilidade significa que o robô pode efetuar uma variedade de tarefas – ou a mesma tarefa de diferentes maneiras – sem mudar a estrutura mecânica ou o sistema de controlo.

Um robô é composto pelos seguintes subsistemas [11]:

- Mecanismo: consiste numa estrutura mecânica articulada acionada por atuadores elétricos, pneumáticos ou hidráulicos, que transmitem movimento às juntas usando sistemas de transmissão adequados;
- Sistemas de perceção: ajudam o robô a adaptar-se a mudanças no ambiente. Consistem em sensores que fornecem informação sobre o estado do robô (posição/velocidade das juntas) e sensores exteriores que obtêm informação sobre o ambiente (proximidade de contacto, distâncias, visão artificial);
- Controlador: realiza os objetivos associados à tarefa. Gera sinais para os atuadores com base na informação recebida pelo utilizador e pelos sensores;
- Interface de comunicação: através desta aplicação o utilizador programa as tarefas que o robô deve efetuar;
- Ambiente e periféricos: constituem o espaço onde o robô vai operar.

2.3.1 Características de um manipulador

Como referido anteriormente, um manipulador tem por objetivo imitar um braço humano, logo pode-se dividir a sua estrutura em duas partes, o efetor (também referido literalmente como punho, mão ou órgão) terminal e as articulações mecânicas.

Por efetor terminal, quer-se dizer qualquer dispositivo ou ferramenta que o robô usa para interagir com o ambiente, que equivale à mão no braço humano. Nestas "mãos", pode-se instalar garras (magnéticas, pneumáticas ou mecânicas) ou ferramentas como soldadoras, perfuradoras, pistolas de pintura, etc.

A função das articulações mecânicas é colocar o órgão terminal numa dada localização (posição e orientação). Esta estrutura é composta por um conjunto de elos e juntas, que tem origem numa base fixa e a outra extremidade conecta ao órgão terminal [11].

Adicionalmente, existem algumas características que têm de ser consideradas tendo em conta a aplicação ou objetivo do robô [11], [12], [13]:

- Volume de trabalho: volume que consegue ser percorrido pelo efetor terminal, que depende diretamente do tipo e quantidade de juntas, tal como do comprimento dos elos;
- Carga: limite máximo de peso que o robô consegue carregar;
- Exatidão: indica a diferença entre o ponto programado e o ponto obtido;
- Repetibilidade: especifica a precisão com que o robô consegue voltar ao ponto programado;
- Resolução: o mais pequeno incremento de movimento que se pode obter a partir da junta ou efetor terminal do robô.

2.3.2 Tipos de Juntas

Uma junta conecta duas ligações sucessivas (elos). O número de graus de liberdade, ou a mobilidade da junta, m, pode variar entre $0 \leq m \leq 6$. Quando $m = 1$, as juntas, consoante

o seu movimento podem ser rotacionais ou prismáticas. Uma junta complexa, por exemplo esférica, apresenta mais graus de liberdade.

Na maioria do manipuladores, as juntas são normalmente divididas em dois grupos, as juntas principais, que estão mais próximas da base e as secundárias, que estão mais afastadas da base e que estão ligadas ao efetor terminal [11].

Juntas rotacionais

Este tipo de juntas limitam o movimento entre dois elos a uma rotação sobre um eixo comum. A localização relativa entre os dois elos é dada pelo ângulo do eixo. Dentro desta categoria, as juntas podem-se dividir em rotacionais (R), revolventes (V) ou de torção (T) (Figura 7).

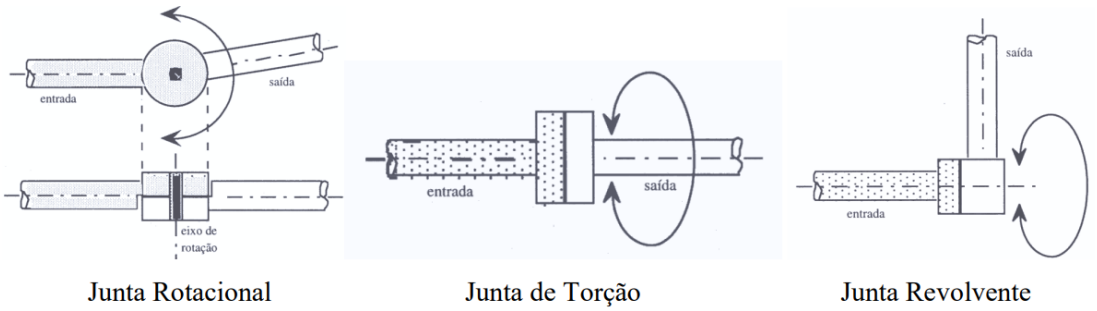


Figura 7 – Distinção entre os tipos de juntas de rotação [13].

Juntas prismáticas

As juntas prismáticas (P ou L), limitam o movimento entre dois elos a uma translação ao longo de um eixo comum. A localização relativa entre os dois elos é determinada pelas distância ao longo do eixo (Figura 8).

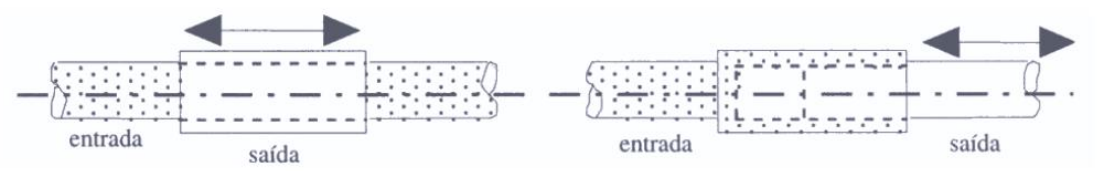


Figura 8 – Juntas prismáticas [13].

Justas esféricas

Este tipo de junta (S), mais complexo pode ser obtido, por exemplo, através de uma combinação de 3 juntas rotacionais cujos eixos intersetam num ponto comum, $m = 3$ (Figura 9).

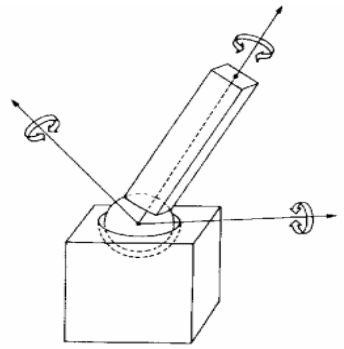


Figura 9 – Junta esférica [12].

2.3.3 Espaço de juntas e espaço operacional

O espaço no qual estão representadas todas as localizações de todos os elos de um robô denomina-se por espaço de juntas. Deste modo, obtém-se as coordenadas do efetor terminal a partir das posições das juntas. Apresenta uma dimensão, *N*, que corresponde ao número de juntas independentes e por sua vez, ao número de graus de liberdade da estrutura mecânica.

Já o espaço operacional, ou espaço de trabalho é definido pela posição e orientação do efetor terminal do robô. Com isto, pode-se obter as coordenadas espaciais das juntas com base no órgão terminal. A dimensão do espaço operacional é dada por, *M*. Consequentemente, $M \leq 6$ ou $M \leq N$.

É de notar que, para uma dada posição e orientação do efetor terminal, podem existir inúmeras combinações no espaço de juntas [11], [12].

2.3.4 Graus de liberdade

Denomina-se por graus de liberdade o número total de movimentos independentes que um manipulador pode efetuar. Um objeto pode sofrer uma translação sobre cada eixo *xyz*, tal como pode sofrer rotação sobre cada um dos mesmos. Isto traduz-se num total de seis graus de liberdade (uma translação e uma rotação para cada um dos 3 eixos).

Como referido no ponto anterior, o grau de liberdade de um manipulador pode ser dado a partir do seu espaço de juntas. Usualmente, um punho/mão de um manipulador pode ter 2 ou 3 graus de liberdade. No que toca à forma de expressar a orientação do punho em relação a um referencial fixo, usam-se os termos *Roll-Pitch-Yaw*:

- Eixo de "*Roll*" (Rotacional): rotação em torno do eixo *x*, num referencial fixo, ou rotação do punho em torno do eixo do braço do robô;
- Eixo de "*Pitch*" (Inclinação): rotação em torno do eixo *y*, num referencial fixo, ou orientação vertical ou inclinação do punho, se o eixo rotacional estiver na posição central corresponde à rotação do punho para cima e para baixo.;
- Eixo de "*Yaw*" (Orientação): rotação em torno do eixo *z*, num referencial fixo, ou se o eixo rotacional estiver na sua posição central corresponde à rotação do punho para a esquerda e para a direita.

Quando o número de graus de liberdade for superior ao espaço operacional, diz-se que o manipulador é redundante. No entanto, esta não é a única característica que determina se um robô é redundante, esta situação vai permitir ao manipulador aumentar o seu volume de trabalho e melhorar o desempenho [11] - [13].

2.3.5 Estruturas de Manipuladores

Tendo em conta os vários tipos de juntas, existem várias configurações possíveis para os robôs industriais. As estruturas mais comuns de manipuladores foram classificadas como [5], [11], [12], [14]:

- Estruturas em série (cadeia aberta):
    - Cartesiano (PPP);
    - Cilíndrico (TPP);
    - Esférico (TRP);
    - SCARA (VRP);
    - Articulado (TRR);
- Estruturas em paralelo ou delta (cadeia fechada);
- Estruturas híbridas.

Cartesiano

Esta categoria de robôs engloba os que apenas realizam movimentos lineares (L), ou seja, apenas têm juntas prismáticas (Figura 10). Geralmente, estão limitados a 3 graus de liberdade, um para cada eixo principal *xyz*. O nome desta configuração deriva do volume de trabalho que consegue alcançar.

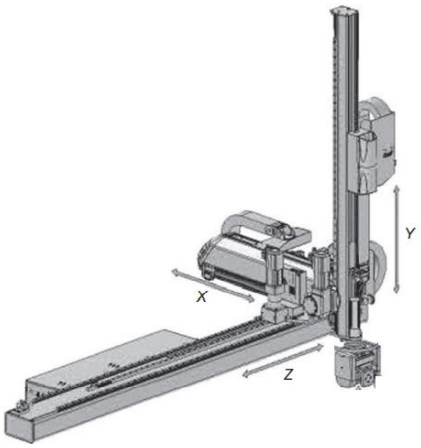


Figura 10 – Estrutura de um robô cartesiano [5].

Cilíndrico

Os robôs cilíndricos são uma combinação de eixos rotacionais e prismáticos, tipicamente, com uma base rotacional (T), seguido por uma junta prismática vertical e outra horizontal, tal como no caso anterior, o nome deriva do volume de trabalho. Fornecem uma estrutura rígida e são fáceis de programar e visualizar. São particularmente adequados para manutenção de máquinas e aplicações gerais do tipo *pick and place* (Figura 11).

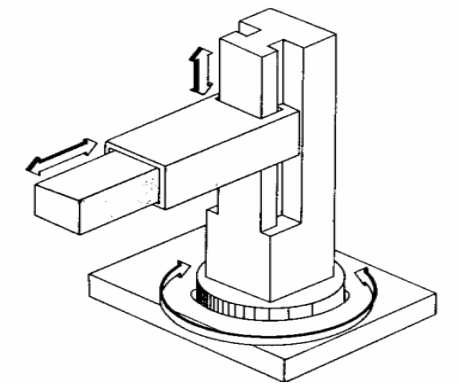


Figura 11 – Estrutura de um robô cilíndrico [12].

Esférico

Geralmente constituído por uma base de torção (T) seguido por uma junta rotacional (R), que finalmente se conecta, por um elo a uma, junta prismática (P). Este conjunto de ligações, permite que robô obtenha, como o nome indica, um volume de trabalho esférico. Assim sendo, são robôs usados para moldagem, soldagem e manuseio de materiais (Figura 12).

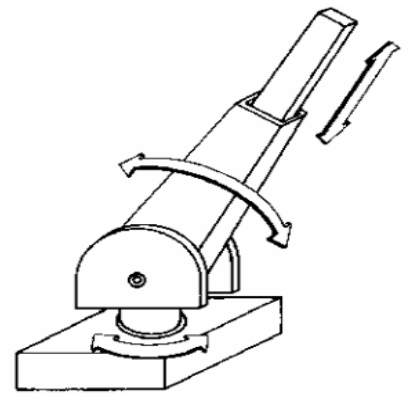


Figura 12 – Estrutura de um robô esférico [12].

SCARA

A configuração SCARA (*Selective Compliance Assembly Robot Arm*) é composta por uma base com uma junta de torção (T) seguida por uma ou mais juntas do mesmo tipo e ainda uma junta prismática vertical (P). São maioritariamente usados para aplicações com

montagens, apesar de também se poder usar em processos de embalamento e de servir como dispensador (Figura 13).

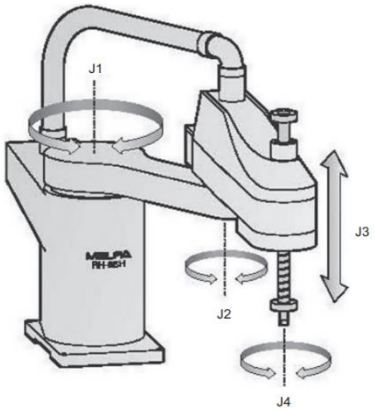


Figura 13 – Estrutura de um robô SCARA [5].

Articulado

A estrutura envolve três ou mais (habitualmente seis) juntas rotacionais (R ou T), cada uma montada à articulação anterior (Figura 14). O movimento de robô de braço articulado é complexo e a construção do braço significa que cada articulação deve suportar o peso de todas as juntas a seguir, ou seja, a junção 1 carrega a 2 e 3. Este tipo de robô é usado para muitas aplicações de processo, incluindo soldagem e pintura.

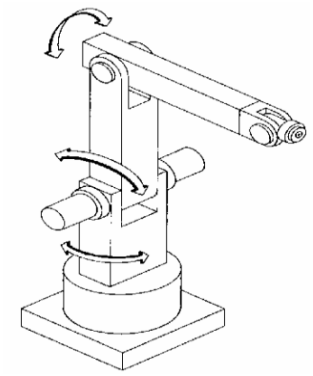


Figura 14 – Estrutura de um robô articulado [12].

Paralelo

A configuração em paralelo ou delta é uma das mais recentes. Possui três, ou mais, braços conectados a uma base comum que está montada sobre a área de trabalho (Figura 15). O

benefício desta aplicação é que pode reduzir o peso sobre os braços e assim obter-se maiores velocidades nos movimentos. O volume de trabalho desta configuração é em forma de cúpula. Adicionalmente, esta aplicação é muito usada em situações de *pick and place* rápido nas indústrias alimentar farmacêutica e eletrónica.

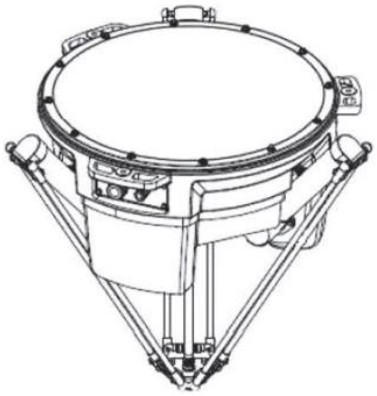


Figura 15 – Estrutura de um robô paralelo [5].

Híbrido

Os manipuladores híbridos são o resultado da combinação de estruturas série e estruturas em paralelo, para aproveitar as melhores qualidades de cada arquitetura. Ainda assim, os manipuladores híbridos podem ser divididos em série-paralelo ou hiper-redundantes. A configuração serie-paralelo é a junção de um manipulador série com um manipulador paralelo. Isto resulta num robô cujos graus de liberdade são somados aumentando a mobilidade do mesmo. Os manipuladores hiper-redundantes (Figura 16) podem ser considerados como manipuladores série no qual os elos são substituídos por cadeias cinemáticas fechadas, excedendo assim a dimensão do espaço de trabalho, preservando as vantagens dos manipuladores paralelos.

Figura 16 – Estrutura de um robô híbrido [14].

2.4 Sensores e Atuadores

O ser humano, recorre ao sistema nervoso sensorial para ser capaz de absorver e processar toda a informação que o rodeia. Na eletrónica um sensor é conhecido como qualquer componente ou circuito eletrónico que permite a análise de uma determinada condição ou propriedade do ambiente. Desta forma, os sensores traduzem a informação captada para um sinal elétrico que é enviado para um recetor onde irá ser decifrado e interpretado pelo sistema.

O uso de sensores na robótica, permite que os robôs tenham um certo sentido de perceção, ou seja, caso exista uma alteração no ambiente de trabalho, o sistema tem capacidade de se adaptar à mudança, isto é, se estiver programado para isso. Apesar de os sensores mencionados abaixo serem mais direcionados para a robótica móvel, pode haver algumas situações onde estes sejam úteis em manipuladores.

Os sensores podem ser divididos nas seguintes categorias [2], [12], [15], [16]:

- Codificadores (*encoders*);
- Sensores analógicos;
- Sensores digitais;

Também será abordada nesta secção a Visão Artificial que é uma tecnologia que utiliza sensores para obter informações visuais.

Codificadores

De forma simplificada, um codificador, ou *encoder*, é um dispositivo que converte movimento para um sinal elétrico, que pode ser lido por um dispositivo de controlo. Estes sinais elétricos enviados podem fornecer informação que determina a posição, velocidade ou direção. Um *encoder* é composto por um disco perfurado (faixa de código), onde num dos lados existe um emissor de luz e no outro um recetor. À medida que o disco gira, a faixa de código interrompe a emissão de luz para o recetor, resultando num sinal digital (Figura 17).

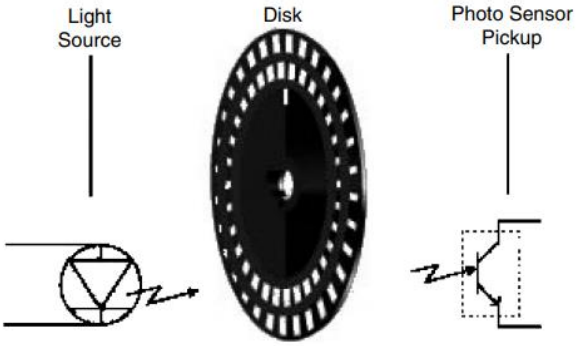


Figura 17 – Representação genérica de um codificador [2].

Adicionalmente, os codificadores podem produzir sinais incrementais ou absolutos. Os sinais incrementais não indicam a posição específica, apenas que houve uma alteração da mesma. Um exemplo de um dispositivo desta categoria pode ser o codificador em quadratura, que pode ser composto por dois sinais desfasados 90º entre si. Desta forma, pode-se saber o sentido de rotação com base no sinal que vai à frente.

Contrariamente aos incrementais, os codificadores absolutos, usam faixas de código para identificar diferentes posições. O número de faixas é proporcional à resolução do *encoder*, ou seja, um *encoder* de 8-bits apresenta oito faixas e um de 6-bits apresenta seis faixas (Figura 18) [2], [12], [15].

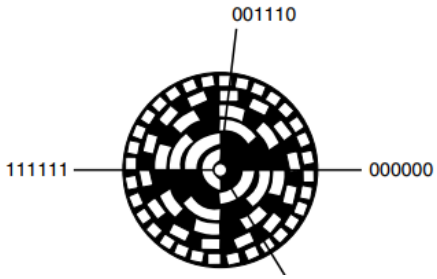


Figura 18 – Exemplo de um codificador absoluto (6-bits) [2].

Sensores analógicos

Os tipos de sensores analógicos vulgarmente usados em aplicações robóticas são de deslocamento, pressão, temperatura, entre outros.

Os sensores de deslocamento medem alterações na posição de um objeto, relativamente a uma posição de referência. Estes tipo de sensores, fornece um sinal de saída variável, que pode ser proporcional à posição do objeto. Para além disso, podem ser usados para determinar a altura, espessura ou largura de um objeto.

Dentro da categoria dos sensores de pressão, podem-se encontrar os piezo-resistivos (ou *strain gage*). O sensor é composto por um transportador isolante, uma folha ou grelha metálica ligada ao transportador isolante e uma camada isolante adicional aplicada no topo da grelha. A resistência elétrica vai alterando conforme a pressão/deformação a que o sensor está sujeito [2], [16].

Sensores Digitais

A saída de um sensor digital apenas apresenta dois estados elétricos "*on*" ou "*off*" (ou 0 e 1). Além dos *encoders*, a maioria dos sensores digitais usados em aplicações robóticas são estáticos, ou seja, o seu valor é baseado no estado digital da saída, em vez da frequência de um gerador de impulsos.

O tipo mais simples e mais económico de sensores digitais é o uso de um interruptor mecânico. Por um lado, quando o interruptor está aberto não existe continuidade (ou resistência infinita), simulando assim o estado "*off*". Por outro lado, quando se encontra fechado, permite a passagem de sinal, ficando assim "*on*".

No entanto, existem outros tipos de sensores que não recorrem a interrutores para reduzir problemas de contacto. Estes fazem uma conversão de sinais analógicos para sinais digitais [2].

Visão Artificial

Quando uma imagem é capturada por uma câmara digital, a luz passa sobre a lente e incide sobre o sensor. O sensor de imagem regista a quantidade de luz que incide sobre o mesmo. De seguida, converte essa quantidade de luz num número correspondente de eletrões. O sinal elétrico obtido é posteriormente convertido num valor correspondente à intensidade de luz capturada, com recurso a um conversor analógico/digital.

Atualmente, os dois principais tipos de sensores de visão, são os *Charge Coupled Device* (CCD) e os *Complementary Metal Oxide Silicon* (CMOS). A principal diferença entre as duas é que um sensor CMOS já vem incorporado com amplificadores e conversores A/D, enquanto os sensores CCD requerem que essas operações sejam feitas no seu exterior. Adicionalmente, os CMOS têm menor consumo energético [17].

Muitos robôs utilizam câmaras para detetar peças, inspecionar elementos ou até mesmo como forma de orientação. Um sistema de visão é instalado num computador e é utilizado para adquirir e examinar a imagem. Através da análise da imagem, consegue-se extrair as características essenciais e tomar uma determinada ação com base na imagem obtida [2].

Outra alternativa é o uso de câmaras inteligentes (*smart cameras*), que são sistemas de visão que já vêm embebidas com *software* de processamento de imagem (Figura 19). A arquitetura destas câmaras pode ser descrita por um microprocessador de bordo e dispositivos de lógica programável que permitem a incorporação de algoritmos de processamento de imagem [18].

Figura 19 – Exemplo de uma câmara inteligente [18].

Atuadores

Como visto anteriormente, os sensores são ferramentas que permitem a perceção do ambiente e do estado atual do robô. No entanto, para traduzir essa informação obtida numa ação, e dar origem ao movimento do manipulador é necessário a existência de um sistema de acionamento, capaz de converter uma fonte de energia em movimento. Assim sendo, pode-se dividir os atuadores em diferentes categorias com base na sua fonte de energia [2], [12], [13], [19]:

Elétricos: os atuadores elétricos tornaram-se muito atrativos, especialmente no que toca à robótica, devido à sua facilidade de comando e flexibilidade. Os motores passo-a-passo (*stepper motors*), são um exemplo de atuadores elétricos que podem ser utilizados na robótica para aplicações de posicionamento de equipamentos. Estes motores podem ter configurações rotacionais ou lineares. Para o seu funcionamento, são enviados um conjunto de impulsos elétricos para os enrolamentos, que posteriormente se convertem em movimento, onde o número de passos será dado pelo número de impulsos e o espaçamento entre impulsos controla a velocidade.

Hidráulicos: estes tipos de atuadores são usados geralmente em aplicações que necessitam de manusear cargas pesadas. Os atuadores hidráulicos produzem movimento por meio do controlo de fluxo ou pressão de um fluído incompressível. Como se usam fluidos incompressíveis, estes atuadores são adequados para controlo de força, posição e velocidade, apesar dos atuadores ele´tricos serem mais rápidos e precisos. Além disso podem ser usados para suspender uma carga sem consumo significativo de energia.

Pneumáticos: estes atuadores são semelhantes aos hidráulicos, ambos operam através de fluxo sob pressão. A diferença é que o ar é compressível contrariamente ao óleo, usado nos atuadores hidráulicos. A utilização de ar pressurizado para gerar movimento, pode ter algumas limitações, tal como menor capacidade de manusear carga e baixa precisão. No entanto, apresentam um peso e tamanho relativamente baixo e ainda conseguem trabalhar com velocidades elevadas.

2.5 Cinemática

2.5.1 Cinemática de um corpo rígido

A cinemática é o ramo da mecânica que estuda a geometria do movimento de um corpo sólido sem considerar as causas que o originaram [14].

Efetuar uma manipulação implica mover algo no espaço através de um mecanismo [20]. Assim sendo, é necessário representar a posição e orientação dos elementos que estão

inseridos no meio. Para isto, é preciso definir sistemas de coordenadas e um modelo de representação.

Um ponto no espaço tridimensional pode ser representado por um vetor posição de três coordenadas (Figura 20). Um vetor é associado a uma deslocação com uma certa direção e sentido, a partir da origem de um sistema de eixos de referência, com coordenadas na sua extremidade [12]. Adicionalmente, a soma de um ponto com um vetor resulta noutro ponto com as coordenadas da deslocação efetuada. Ao movimentar um ponto no espaço está-se a alterar as suas coordenadas.

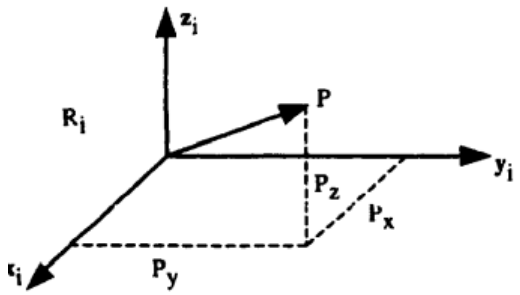


Figura 20 – Representação de um vetor posição.

Seja *P* um ponto expresso em relação a um referencial ortogonal *OXYZ* representado por *R*. Este ponto pode ser representado por um vetor ***p*** com origem em *O*, tal como:

$$ {}^{R}\vec{p} = {}^{R}\boldsymbol{p} = \begin{bmatrix} {}^{R}p_x \\ {}^{R}p_y \\ {}^{R}p_z \end{bmatrix} \qquad (2.1) $$

Na robótica, não se pretende mover apenas um ponto, mas sim um conjunto deles. No caso de um manipulador, cada um dos seus elos é um corpo rígido, ou seja, um conjunto de partículas de tal modo que a distancia entre quaisquer duas partículas se mantém. Apesar da distância entre pontos ser fixa, pode haver uma alteração da sua orientação. Portanto, de maneira a descrever a orientação de um corpo rígido, cria-se um referencial associado a mesmo, com coordenadas relativas a um referencial fixo [12], [20], [21].

Supondo então que se tem dois referenciais *OXYZ*, um referencial fixo (R_0) e um referencial que pode rodar em relação ao anterior (R_1). Sejam ***i***, ***j***, ***k*** os versores ao longo do eixo-*X*, eixo-*Y* e eixo-*Z*, respetivamente. Quando se representa o referencial móvel em função do sistema de coordenadas fixo serão representados por ${}^{0}\boldsymbol{i}_1$, ${}^{0}\boldsymbol{j}_1$, ${}^{0}\boldsymbol{k}_1$. Como as

componentes de cada vetor são as projeções desse vetor em relação aos versores do seu eixo de referência, pode-se escrever cada componente de ${}^{0}R_{1}$ como o produto escalar de um par de vetores unitários [20], [22].

$$ {}^{0}R_{1} = [{}^{0}\boldsymbol{i}_{1} \quad {}^{0}\boldsymbol{j}_{1} \quad {}^{0}\boldsymbol{k}_{1}] = \begin{bmatrix} \boldsymbol{i}_1 \cdot \boldsymbol{i}_0 & \boldsymbol{j}_1 \cdot \boldsymbol{i}_0 & \boldsymbol{k}_1 \cdot \boldsymbol{i}_0 \\ \boldsymbol{i}_1 \cdot \boldsymbol{j}_0 & \boldsymbol{j}_1 \cdot \boldsymbol{j}_0 & \boldsymbol{k}_1 \cdot \boldsymbol{j}_0 \\ \boldsymbol{i}_1 \cdot \boldsymbol{k}_0 & \boldsymbol{j}_1 \cdot \boldsymbol{k}_0 & \boldsymbol{k}_1 \cdot \boldsymbol{k}_0 \end{bmatrix} \qquad (2.2) $$

É de notar que as colunas da matriz ${}^{0}R_{1}$ representam as coordenadas dos eixos principais do referencial R_0 em relação ao referencial R_1, isto é, representam os cossenos diretores dos eixos do referencial R_1 em relação ao referencial R_0. Por analogia, as linhas correspondem às coordenadas do referencial R_0 em relação a R_1.

Isto significa que o referencial R_0 representado em função de R_1 é obtido através de uma transposição:

$$ {}^{1}R_{0} = {}^{0}R_{1}^{T} = {}^{0}R_{1}^{-1} \qquad (2.3) $$

Para além disso, como as matrizes são ortogonais, a sua transposta também é igual à sua inversa.

Desta forma, consegue-se determinar transformações que representam rotações de um referencial em relação a outro. Uma rotação em torno de um eixo significa que a coordenada desse eixo se mantém enquanto os restantes alteram dado um ângulo (Figura 21). Por exemplo, uma rotação de θ em relação ao eixo *x* é dada por:

$$ Rot(x,\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -\sin\theta \\ 0 & \sin\theta & cos\,\theta \end{bmatrix} \qquad (2.4) $$

Enquanto uma rotação de θ em relação ao eixo *y* é dada por:

$$ Rot(y,\theta) = \begin{bmatrix} cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & cos\,\theta \end{bmatrix} \qquad (2.5) $$

Para o caso de se efetuar em torno de *z*:

$$Rot(z,\theta) = \begin{bmatrix} cos\theta & -\sin\theta & 0 \\ \sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.6)$$

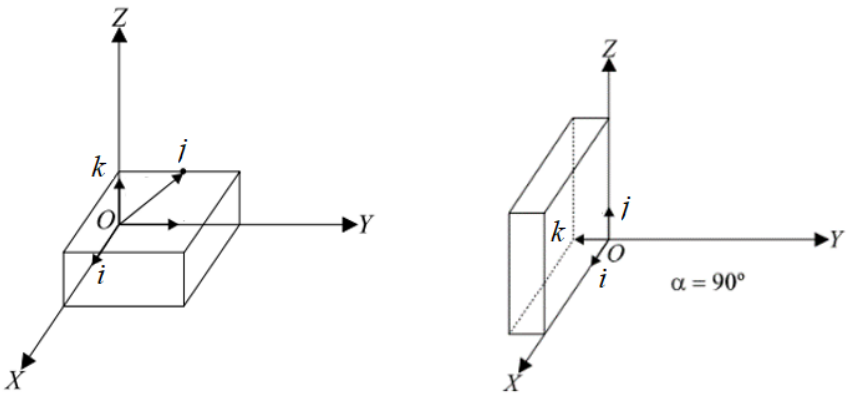


Figura 21 – Rotação de 90º de um corpo em torno do eixo x.

Estas matrizes representam rotações elementares e são muito úteis quando apenas ocorre uma rotação em torno de um ângulo. Todavia, quando se pretende efetuar um conjunto de rotações em torno de eixos diferentes a transformação final pode ser obtida pelo produto das matrizes de rotação. Uma rotação em torno de *x* por ângulo *γ*, seguida de um rotação em *y* por *β*, e em *z* por *α*, pode ser dada pela equação *(*2.7*)*.

Onde, *cα* = *cosα*, *sα* = *sinα* e assim sucessivamente:

$$RPY(\alpha,\beta,\gamma) = Rot(z,\alpha)Rot(y,\beta)Rot(x,\gamma) =$$

$$\begin{bmatrix} cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & cos\beta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & cos\gamma \end{bmatrix} = \quad (2.7)$$

$$\begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}$$

Cada uma destas três rotações dá-se sobre o referencial fixo R_0. Como as operações de rotação não são comutativas é importante seguir a ordem correta da rotação, onde a primeira matriz de rotação, *Rot(x, γ)*, é a que se encontra mais à direita. Esta representação

é conhecida como uma orientação com ângulos fixos *Roll-Pitch-Yaw* (Figura 22) [12], [20], [21].

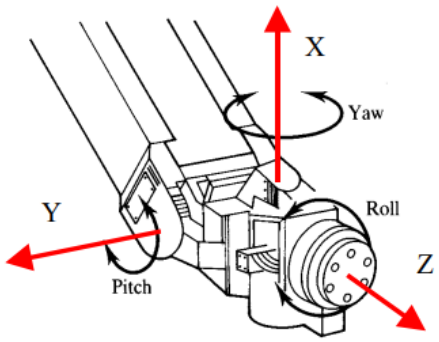


Figura 22 – Representação Roll-Pitch-Yaw na garra de um manipulador [12].

Para além desta representação, também existem os ângulos de *Euler* [2], [20], [21]. Contrariamente ao caso anterior, cada rotação dá-se sobre o referencial que move. A variante *Z-Y-X* de *Euler* começa com uma rotação sobre o eixo *z* do referencial, seguido por uma rotação no novo eixo *y'* e por finalmente uma rotação em *x''* (Figura 23).

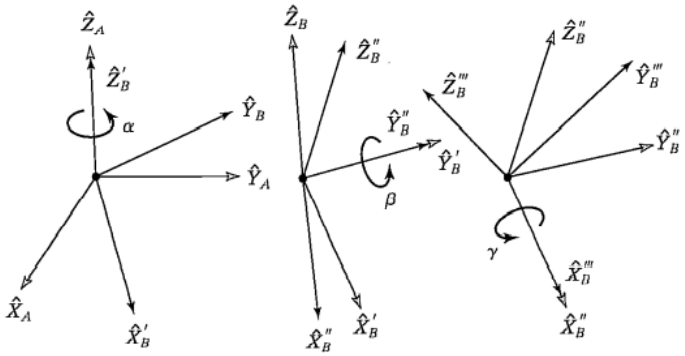


Figura 23 – Composição de rotações em torno de um referencial móvel [20].

$$Euler(\gamma,\beta,\alpha) = R(z,\alpha)R(y,\beta)R(x,\gamma) =$$

$$= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \qquad (2.8)$$

Como se pode verificar, as matrizes resultantes das equações *(*2.7*)* e *(*2.8*)* são iguais, apesar dos métodos aplicados serem diferentes. De modo geral, três rotações sobre um

eixo fixo resultam na mesma orientação final que três rotações feitas na ordem inversa sobre um referencial móvel.

2.5.2 Transformadas homogéneas

A construção de uma matriz de transformação homogénea resulta da utilidade de representar rotações e translações em conjunto. Esta transformada pode ser decomposta em quatro sub-matrizes, das quais duas já são conhecidas – matriz de rotação (normalmente representada pelos vetores *noa*) e o vetor posição. As outras componentes adicionais são a transformação de perspetiva (*Pres*) e o fator de escala (*S*) [11] - [13], [20] - [22]. Em robótica o fator de escala é 1 e não se utiliza a transformação de prespetiva.

A matriz de transformação homogénea de R_1 em relação a R_0 é dada por:

$$ {}^{0}H_{1} = \begin{bmatrix} & & & \vdots & \\ & Rot & & \vdots P & \\ \cdots & \cdots & \cdots & \vdots \cdots & \\ & Pres & & \vdots S & \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.9) $$

Se quisermos ver na perspetiva de R_0 em relação a R_1:

$$ {}^{1}H_{0} = {}^{0}H_{1}^{-1} = \begin{bmatrix} n_x & n_y & n_z & -p_x \cdot n \\ o_x & o_y & o_z & -p_y \cdot o \\ a_x & a_y & a_z & -p_z \cdot a \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.10) $$

Caso se queira fazer uma translação de R_1 em relação a R_0 para um ponto *P* de coordenadas *a*,*b* e *c*:

$$ {}^{0}H_{1} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.11) $$

Caso se queira fazer uma rotação de θ graus em R_1, ao longo do eixo *x* do referencial R_0, equação essa similar à *(*2.4*)*:

$$ {}^{0}H_{1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta & -s\theta & 0 \\ 0 & s\theta & c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.12) $$

2.5.3 Denavit-Hartenberg

Anteriormente viu-se que atribuir um referencial a um corpo rígido facilitava interpretação da sua posição e orientação no espaço. Ora, tendo em conta que um elo de um manipulador é um corpo rígido e se for atribuído a cada elo um referencial, é possível criar um modelo que descreva a localização dos referenciais no espaço.

A convenção *Denavit-Hartenberg* (*DH*) é um método de associar cada referencial, expresso em transformadas homogéneas a uma junta, relacionando assim os elos com as juntas [11]. Os elos são numerados de maneira que o *elo 0* represente a base do manipulador, e o *elo n* seja o elo terminal ($n = N$ = dimensão do espaço de juntas). Uma *junta i* conecta o *elo i* ao *elo i-1* ($i = 1, 2, \dots, n$) (Figura 24), [11], [13], [14], [20] - [22].

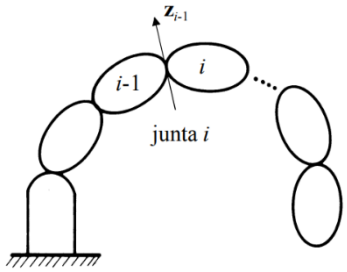


Figura 24 – Numeração de elos e juntas num manipulador [22].

É importante referir que existem dois métodos de aplicar esta convenção, o clássico e o modificado [23], [24]. Este documento recorre ao método clássico.

Tendo em conta as considerações anteriores, os referenciais podem ser estabelecidos de maneira que:

- O eixo Z_{i-1} coincide com o eixo de movimento da *junta i*;
- O eixo X_i é normal ao eixo Z_{i-1};

- O eixo Y_i completa o sistema de coordenadas de acordo com a regra da mão direita.

Após a atribuição dos referenciais, podem-se determinar quatro parâmetros geométricos, os quais dois descrevem a ligação entre elos vizinhos (θ_i, d_i) e os restantes determinam a estrutura do elo (a_i, α_i). Os parâmetros *DH* são definidos da seguinte forma:

- α_i - Ângulo entre Z_{i-1} e Z_i em torno de X_i;
- a_i - Distância desde a origem do referencial *i* até à interseção de Z_{i-1} com X_i ao longo de X_i;
- θ_i - Ângulo entre X_{i-1} e X_i em torno de Z_{i-1};
- d_i - Distância desde a origem do referencial *i-1* até à interseção de Z_{i-1} com X_i ao longo de Z_{i-1};

Se a *junta i* for rotacional, o parâmetro θ_i é um valor que pode variar e d_i é fixo. Se a *junta i* for prismática, acontece o inverso.

Após a obtenção dos referenciais e dos parâmetros *DH* é possível determinar a matriz de transformação ${}^{i-1}A_i$, que relaciona o *referencial i* com o *referencial i-1*, através de uma sequência de transformações elementares (Figura 25):

1. Rotação θ_i em torno do referencial Z_{i-1};
2. Translação de d_i unidades ao longo de Z_{i-1};
3. Translação de a_i unidades ao longo de X_i;
4. Rotação α_i em torno do referencial X_i.

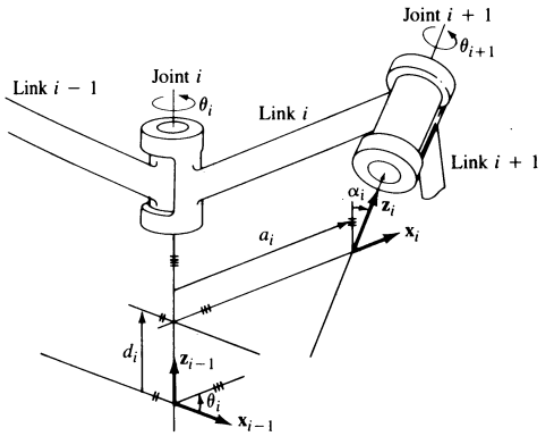


Figura 25 – Representação das transformações Denavit-Hartenberg [22].

A tradução da sequência de transformações descrita para uma equação é dada por:

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

$$\begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & s\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.13)$$

$$= \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i c\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Esta matriz homogénea é a chave para o estudo da cinemática de manipuladores.

2.5.4 Cinemática direta

A cinemática direta de um robô determina a configuração do efetor terminal tendo em conta as posições relativas das juntas do manipulador. Posto isto, a matriz homogénea ^0A_n, que especifica a localização do referencial do efetor terminal em relação ao referencial da base é dada por:

$$^0A_n = {}^0A_1\,{}^1A_2 \dots {}^{i-1}A_i, \qquad para\ i = 1, 2, \dots, n \qquad (2.14)$$

Como se pode verificar pela equação anterior, a convenção de *Denavit-Hartenberg* é bastante útil para a resolução do problema da cinemática direta. Assim sendo, para fazer o estudo da cinemática direta de um manipulador série é necessário seguir os seguintes passos [12], [20] - [24]:

1. Definir um sistema de coordenadas para cada elo, começando pela base com eixo Z_0 manipulador ao longo do eixo de movimento da junta 1. Os eixos X_0 e Y_0 são escolhidos livremente, desde que respeitem a regra da mão direita (Figura 26) [25];
2. Fazer coincidir Z_i com o eixo da junta $i+1$, até $n-1$. No caso de Z_n, este coincide com a direção do punho do manipulador;

3. Atribuir a origem do referencial *i* na intersecção de Z_i com Z_{i-1} ou na intersecção da normal comum aos eixos Z_i e Z_{i-1} com o eixo Z_i;
4. Apontar X_i ao longo da normal comum caso Z_i e Z_{i-1} sejam paralelos. Se Z_i e Z_{i-1} forem perpendiculares, X_i será a normal resultante ($Z_i \times Z_{i-1}$).
5. Y_i completa o referencial de acordo com a regra da mão direita;
6. O último referencial será colocado na ponta da garra , onde Z_n tem o sentido de aproximação (aponta para o objeto que vai apanhar), Y_n tem sentido de abertura/fecho da garra, e X_n completa o sistema de eixos de acordo com a regra da mão direita.
7. Após definir todos os referenciais, calcular os parâmetros *DH* para as *n* juntas.
8. Aplicar a equação (*2*.13) e (*2*.14) para determinar a localização do efetor terminal em relação à base ${}^0A_n$.

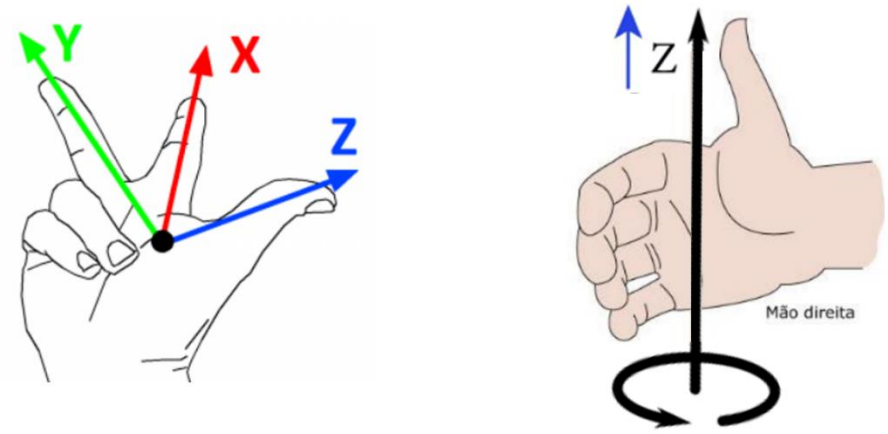


Figura 26 – Regra da mão direita.

Sendo assim, através de um vetor de junta, que indica como está cada uma das juntas posicionadas, é possível obter a matriz homogénea que indica a posição e orientação do efetor terminal relativamente ao referencial base

2.5.5 Cinemática inversa

Enquanto o objetivo da cinemática direta é determinar a posição e orientação do órgão terminal para uma dada configuração de juntas, a cinemática inversa consiste na obtenção

do espaço de juntas do manipulador com base na localização (posição e orientação) do efetor terminal. Ora, isto significa que a complexidade desta tarefa é maior, pois apenas se sabe a posição desejada e pretende-se encontrar uma sequência de comandos que levem o punho do manipulador até essa posição. Ainda assim, a principal dificuldade da cinemática inversa é o facto de poder ter uma ou várias soluções, ou até mesmo nenhuma.

O modelo que permite obter todas as soluções possíveis para este problema denomina-se por Modelo Geométrico Inverso (*Inverse Geometric Model*, *IGM*). Existem vários métodos para obter este modelo. Neste caso, será apresentado o método de *Richard P. Paul* [11], [13], [20], [21] :

Seja ${}^{W}A_{G}$ a matriz de transformação homogénea que representa a localização desejada (*G*) em relação ao referencial mundo, ou universo (*W*). Normalmente define-se um referencial mundo quando existe um ambiente com vários elementos (outros manipuladores ou sensores). Esta matriz pode ser dada por:

$$ {}^{W}A_{G} = {}^{W}A_{0}\,{}^{0}A_{n}(q)\,{}^{n}A_{G} \qquad (2.15) $$

Onde:

- ${}^{W}A_{0}$ é a matriz de transformação que define a base do manipulador em relação ao referencia mundo;
- ${}^{0}A_{n}(q)$ é a matriz de transformação que define a localização do efetor terminal em relação à base, em função do vetor das variáveis de junta (*q*):

$$ \boldsymbol{q} = [q_1\ q_2\ \dots\ q_n] $$

- ${}^{n}A_{G}$ é a matriz de transformação que define o referencial da localização desejada em relação ao efetor terminal.

Colocando todas as matrizes conhecidas em relação ao lado esquerdo da equação:

$$ {}^{W}A_{0}^{-1}\,{}^{W}A_{G}\,{}^{n}A_{G}^{-1} = {}^{W}A_{0}^{-1}\,{}^{W}A_{0}\,{}^{0}A_{n}(q)\,{}^{n}A_{G}\,{}^{n}A_{G}^{-1} \qquad (2.16) $$

Como o produto de uma matriz com a sua inversa resulta na matriz identidade, obtém-se:

$$ {}^{W}A_0^{-1}\,{}^{W}A_G\,{}^{n}A_G^{-1} = {}^{0}A_n(q) \qquad (2.17) $$

A inversa de uma matriz ${}^{i-1}A_i$ resulta numa matriz ${}^{i}A_{i-1}$, logo:

$$ {}^{0}A_W\,{}^{W}A_G\,{}^{G}A_n = {}^{0}A_n(q) \qquad (2.18) $$

Seja o produto entre ${}^{0}A_W$, ${}^{W}A_G$, e ${}^{G}A_n$ a matriz de transformação que representa a localização desejada ${}^{0}T_n$:

$$ {}^{0}T_n = {}^{0}A_n(q) \qquad (2.19) $$

De tal modo que:

$$ {}^{0}T_n = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.20) $$

Assim sendo, o *IGM* é obtido, através da resolução da seguinte equação:

$$ {}^{0}T_n = {}^{0}A_1(q_1)\,{}^{1}A_2(q_2) \ldots {}^{n-1}A_n(q_n) \qquad (2.21) $$

Para resolver a equação, move-se cada dependência da variável de junta q_i para o lado esquerdo da equação:

$$ {}^{1}A_0(q_1)\,{}^{0}T_n = {}^{1}A_2(q_2) \ldots {}^{n-1}A_n(q_n) \qquad (2.22) $$

Simplificando:

$$ {}^{1}T_n = {}^{1}A_n(q) \qquad (2.23) $$

As matrizes ${}^{i-1}A_i$ podem ser facilmente obtidas através do método de transformação *DH* da cinemática direta. Após obter as matrizes resultantes, efetua-se uma igualdade entre os elementos das matrizes que contêm as incógnitas (variáveis de junta), de modo a permitir o seu cálculo.

Através deste método, consegue.se saber quais as variáveis de junta que vão permitir ao manipulador alcançar o objeto. Para simplificação da solução do problema da cinemática inversa poder-se-á impor algumas limitações em certas juntas.

2.6 Processamento de Imagem

O processamento de imagens digitais é uma área da computação que se dedica à decomposição de imagens através de técnicas matemáticas e algoritmos computacionais. Estas técnicas são amplamente utilizadas em diversas áreas, uma das quais é a robótica. O objetivo deste processamento é melhorar a qualidade de imagens digitais, corrigir imperfeições e extrair informações úteis para análise, decomposição e interpretação das mesmas. Este complexo processamento de imagens digitais pode ser divido em várias etapas, como amostragem, quantificação, compressão, melhoramento e análise [26]. Consoante a aplicação em causa algumas destas etapas podem ser misturadas, como por exemplo o melhoramento de uma imagem e uma análise. Igualmente, consoante a literatura, os métodos de melhoramento e análise podem, por sua vez ser decompostos em várias etapas.

2.6.1 Amostragem e Quantificação

A amostragem é o processo de converter uma imagem analógica numa imagem digital, onde a imagem analógica é extraída em pontos discretos, ou *pixéis*, com uma determinada resolução e transformada numa matriz. Já a quantificação é o processo de converter a intensidade (ou amplitude) dos *pixéis* num valor digital discreto. Isso é feito atribuindo um valor numérico a cada *pixel*, com base na intensidade da imagem analógica original. A quantidade de valores que podem ser atribuídos a cada *pixel* depende da precisão da digitalização e do número de bits disponíveis para armazenar cada valor. Atualmente, com as câmaras digitais o processo de amostragem é automático, assim como a quantificação. A imagem resultantes da amostragem e quantificação resulta num ficheiro que basicamente quando lido por um programa, não será mais que uma matriz onde os seus elementos são os pixéis de uma imagem, cada um com intensidades diferentes consoante o mapa de cores utilizado. Os mais comuns são o *RGB*, *HSV*, escala de cinzentos e preto e branco (ou imagem binária).

Uma imagem digital pode ser vista como uma função *f(x,y)* onde *x* e *y* são variáveis discretas que representam as coordenadas (colunas, linhas) dos *pixéis* (Figura 27) [13].

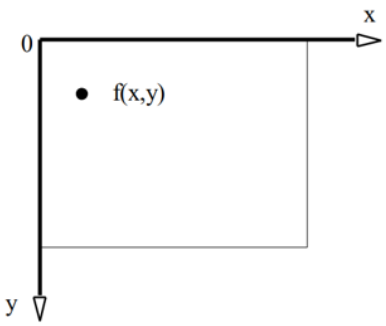


Figura 27 – Representação de uma imagem num sistema xy.

Na forma matricial uma imagem é representada pela seguinte equação, onde N e M representação o número de linhas e colunas, respetivamente:

$$f(y,x) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \dots & \dots & \dots & \dots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix} \quad (2.24)$$

Como se pode verifica pela Figura 27, o eixo *y* irá corresponder à linhas da matriz e o eixo *x* às colunas.

2.6.2 Compressão de Imagens

Em aplicações robóticas, a velocidade de processamento é essencial e quanto maior for o tamanho de uma imagem, maior será o tempo de processamento [13]. A compressão de imagens reduz a quantidade de dados necessários para armazenar uma imagem sem perda significativa de qualidade visual.

O tamanho de uma imagem pode ser medido em *bits* ou *bytes*. Um *bit* é a unidade básica de informação num computador, enquanto um *byte* é um conjunto de 8 *bits*. Assim sendo, o tamanho de uma imagem é dado pelo produto entre o número de *pixéis* na imagem e o número de *bits* necessários para representar a cor de cada *pixel*. Uma imagem a preto e

branco, requer apenas 1 *bit* por *pixel*. Uma imagem a cores, por exemplo *RGB*, utiliza 24 *bits* por *pixel* para representar a cor, pois esta é composta por 3 imagens (dos tons vermelhos, verdes e azuis) onde cada pixel tem 8 *bits*.

Além disso, o número de *bits* de um pixel define a quantidade de níveis de cor existentes. Uma imagem de tonalidade cinzenta, com 8 *bits*, tem $2^8 = 256$ níveis de cinzento, enquanto uma imagem a preto e branco $2^1 = 2$ só tem dois níveis, o preto e o branco.

Pode-se reduzir o tamanho de uma imagem através de uma conversão digital, que reduz o número de *bits* de uma imagem, ou pode-se fazer um enquadramento e apenas usar uma certa secção da imagem, reduzindo a quantidade de *pixéis* apenas na zona que se está a trabalhar. Isto fará com que o processamento necessário a essa parte da imagem seja mais rápido e como tal a ação que se quererá fazer após este processo.

2.6.3 Melhoramento de Imagens

Muitas vezes as imagens contêm ruído que pode ser provocado por diversos fatores como componentes óticos, eletrónicos ou a deficiente iluminação resultando numa imagem escura ou clara, com pouco detalhe. Por esta razão, usam-se algoritmos de melhoramento de imagem que alteram os valores dos *pixéis* [6]. Existem várias técnicas para realizar estas operações, quer seja no domínio espacial ou no domínio da frequência [13]. Os algoritmos de processamento espacial atuam diretamente sobre um conjunto de *pixéis* que constituem a imagem, enquanto o domínio da frequência lida com a taxa a que os valores dos *pixéis* estão a alterar no domínio espacial, daí ser mais útil em situações onde existe movimento de objetos. Este documento vai-se focar apenas em técnicas do domínio espacial.

Assim sendo, duas maneiras de melhorar a qualidade de uma imagem são através da aplicação de máscaras de convolução (também conhecidos como filtros, janelas ou *kernels*) ou manipulação de histogramas [6], [26]. Desta formar, é possível suavizar ou realçar determinadas características de uma imagem, removendo ruído ou realçando contornos.

Filtros Espaciais

As máscaras de convolução são essencialmente um conjunto de valores bidimensionais, onde cada desses valores chama-se coeficiente (*w*) e será escolhido consoante a natureza do processo a realizar (Figura 28). Esta máscara percorre a imagem toda, pixel a pixel, e efetua uma série de operações resultando numa nova imagem [13], [27]. Assim, será criado um pixel *g(x,y)*, de acordo com a equação *(*2.25*)*.

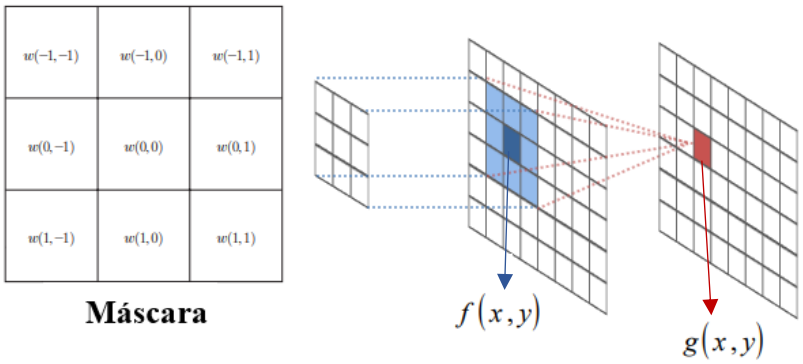


Figura 28 – Aplicação de uma máscara (3x3) a uma imagem f(x,y), que resulta numa nova imagem g(x,y).

Matematicamente, pode-se representar a aplicação de um filtro de dimensão $n \times m$ a uma imagem *f(x,y)* por:

$$g(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) f(x+i, y+j) \qquad (2.25)$$

Onde $a = (n-1)/2$ e $b = (m-1)/2$, no caso da Figura 28, $n = m = 3$. Note-se o que coeficiente *w(0,0)*, correspondente ao centro da máscara, coincide com o valor *do pixel* a alterar [28].

Outra maneira de representar matematicamente a resposta, *R*, a uma máscara (Figura 29) em qualquer ponto *(x,y)* é:

$$R = \sum_{i=1}^{D} w_i \cdot p_i \qquad (2.26)$$

Onde *w* é o coeficiente, *p* é a amplitude do pixel correspondente a esse coeficiente e *D* é o número total de coeficientes. No exemplo da figura abaixo, *D = 9*.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figura 29 – Outra representação de uma máscara 3x3.

Dois exemplos de filtros espaciais são a vizinhança média e a mediana. Ambos são considerados como filtros de alisamento, usados para atenuar ruído [13], [28].

O filtro da vizinhança média substitui o valor de cada pixel numa imagem pela amplitude média dos pixéis na sua vizinhança. Apesar de suavizar o ruído, as imagens ficam menos nítidas. Um exemplo de uma máscara para o filtro da vizinhança média é apresentado na Figura 30 e descrita pela equação (2.27).

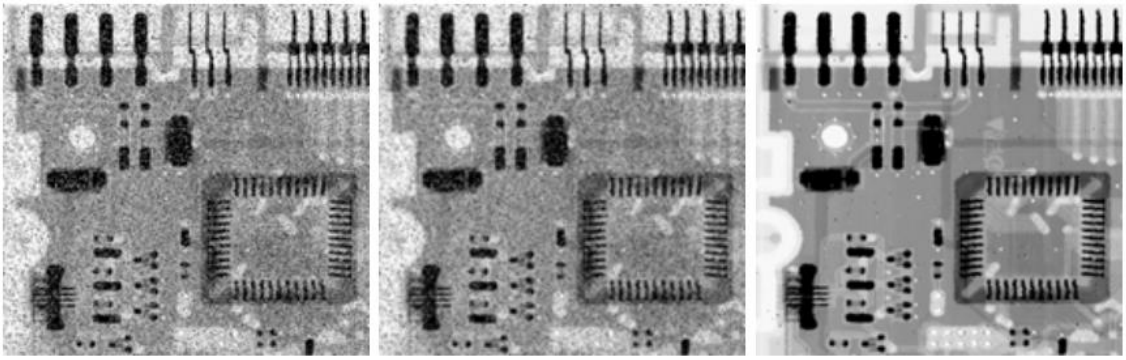
$$R = \frac{1}{9}\sum_{i=1}^{9} w_i \cdot p_i \qquad (2.27)$$

$\frac{1}{9} \times$

1	1	1
1	1	1
1	1	1

Figura 30 – Máscara para o filtro da vizinhança média.

O filtro da mediana, como o nome indica, substitui o valor de um pixel pela mediana da amplitude dos seus pixéis vizinhos. Este tipo consegue eliminar ruído com facilidade sem a consequência desfoque apresentada no caso da vizinhança média. De maneira a aplicar este filtro, ordena-se os valores dos pixéis em questão por ordem crescente, e retira-se o valor mediano.



(a) *(b)* *(c)*

Figura 31 – Imagem com ruído (a); Filtro da vizinhança média (b); Filtro da mediana (c) [28].

A Figura 31 apresenta uma comparação entre os filtros da vizinhança média (b) e da mediana (c), quando aplicados a uma imagem com ruído.

Manipulação de Histogramas

O histograma de uma imagem é uma função que regista a frequência de cada nível de amplitude dessa imagem. É uma ferramenta muito útil para se saber como a cena foi iluminada. Para além disso com base nestes valores pode-se saber qual o valor ótimo para converter uma imagem de vários níveis de intensidade em preto e branco. Para melhorar a aparência de uma imagem, pode-se fazer uma redistribuição dos níveis de modo a ficar mais uniforme. Isto é útil em situações onde a maioria dos *pixéis* está concentrada numa parte do espectro.

A *equalização* ou *linearização* de histogramas é uma transformação para fazer a redistribuição dos níveis de intensidade de cada *pixel* de modo usar uma gama maior de valores e criar mais contraste [6], [13], [28]. Esta transformação baseia-se em funções de densidade de probabilidade e pode ser descrita matematicamente por:

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) \qquad k = 0,1,\ldots,L-1 \tag{2.28}$$

Onde s_k é o nível da imagem equalizada, r_k é o nível da imagem de entrada, L é o número total de níveis (dado pelo número de *bits*) e p_r a função de densidade de probabilidade de r.

A Figura 32 apresenta o processo de equalização de um histograma. Nos histogramas *(b)* e *(d)*, os valores no eixo horizontal representam o nível (r_k) e os valores na vertical representam a quantidade de *pixéis* com cada nível k (n_k). Pode-se verificar que a imagem original *(a)* apresenta pouco contraste e como representado pelo histograma *(b)*, existe uma grande concentração de pixéis com níveis baixos. Após efetuar a equalização, houve uma redistribuição dos *pixéis* para níveis mais altos *(d)*, criando mais contraste na imagem *(c)*.

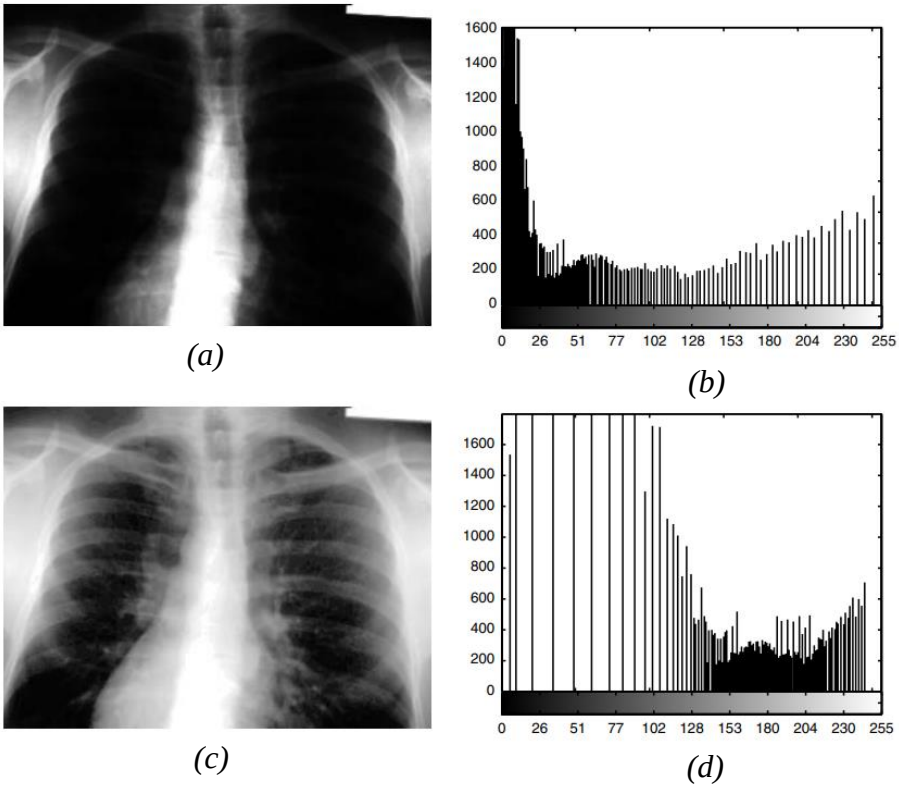


Figura 32 – Imagem original (a); Histograma da imagem original (b); Imagem após a equalização do histograma (c); Histograma equalizado (d) [26].

2.6.4 Análise de Imagens

Para uma imagem ser analisada, as suas características principais têm de ser isoladas. A segmentação é o processo de dividir a imagem em regiões homogêneas, permitindo a análise de cada região separadamente e por sua vez a redução da quantidade de dados a analisar [13], [28].

As técnicas de segmentação mais relevantes são:

- Deteção de contornos (ou arestas);
- Técnica do limiar (*Threshold*);
- Região.

Deteção de Contornos

A *deteção de contornos* refere-se ao processo de identificar as bordas de um objeto numa imagem. Essas bordas são pontos em que há uma mudança significativa na intensidade dos pixéis. Os contornos podem ser usados para separar objetos numa imagem e para extrair informações importantes, como a forma e o tamanho dos objetos.

Pixéis de contorno são pixéis cuja intensidade da imagem muda abruptamente, já os *contornos* são conjuntos de *pixéis de contorno* que estão conectados.

Para detetar mudanças de intensidade usam-se derivadas de primeira e segunda ordem. As derivadas de uma imagem digital são definidas em termos de diferenças. No entanto, qualquer definição usada para representar uma primeira derivada deve:

- Ser zero em zonas onde a intensidade é constante;
- Ser diferente de zero *no início* de uma rampa (*ramp*) ou um degrau (*step*);
- Ser diferente de zero ao longo de uma rampa.

De igual modo, para se definir uma derivada de segunda ordem, esta deve:

- Ser zero em zonas onde a intensidade é constante;
- Ser diferente de zero *no início e no fim* de uma rampa ou um degrau;
- Ser zero ao longo de rampas com declive constante.

Uma aproximação de uma derivada de primeira ordem num ponto *x* de uma função unidimensional *f(x + Δx)*, onde *Δx = 1* pode ser dada por [28]:

$$\frac{\partial f}{\partial x} = f'(x) = f(x+1) - f(x) \tag{2.29}$$

Aplicando a segunda derivada a (2.29), obtém-se:

$$\frac{\partial^2 f}{\partial^2 x} = f'(x+1) - f'(x) = f(x+2) - 2f(x+1) + f(x) \tag{2.30}$$

Esta expansão está se a dar sobre o ponto *x+1*. Como apenas interessa aplicar a segunda derivada sobre o ponto *x*, subtrai-se 1 a cada argumento e fica-se com:

$$\frac{\partial^2 f}{\partial^2 x} = f''(x) = f(x+1) - 2f(x) + f(x-1) \tag{2.31}$$

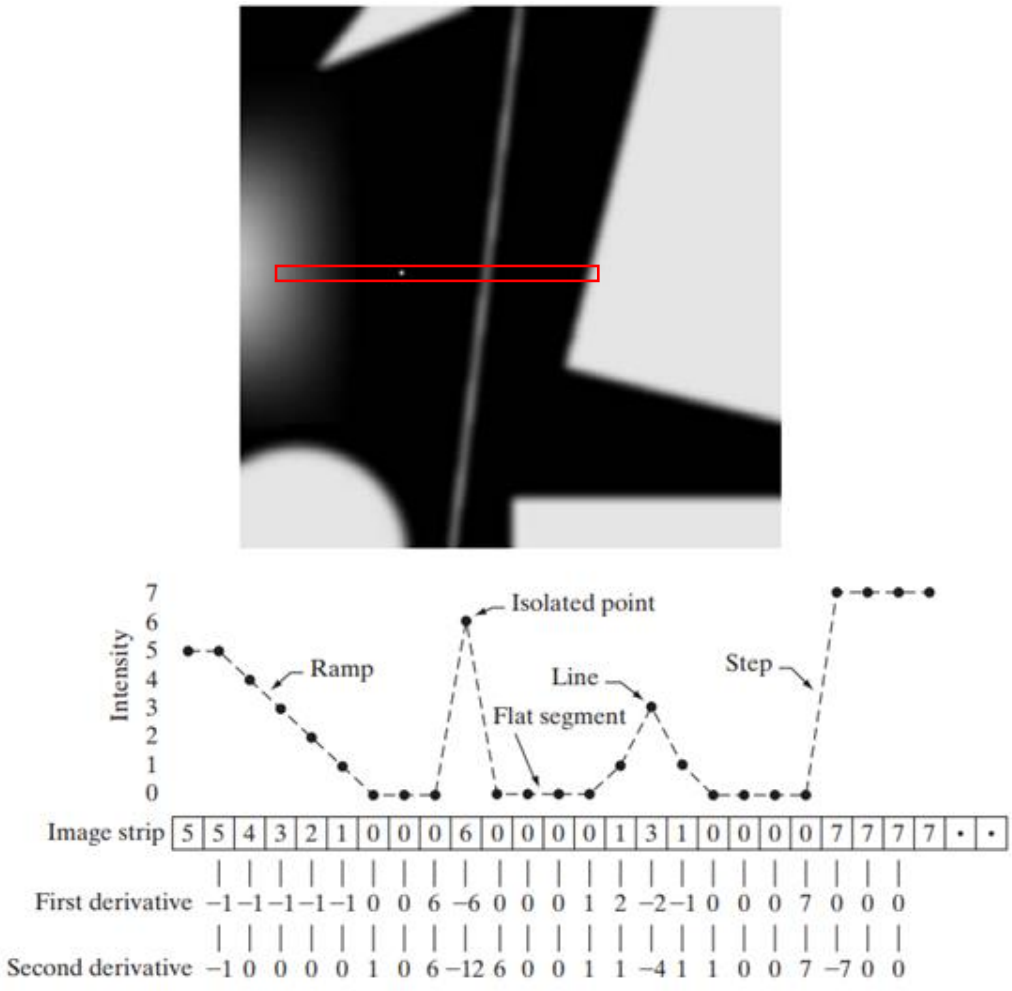


Figura 33 – Imagem com várias formas e uma representação do perfil (simplificada) [28].

A Figura 33 apresenta uma imagem de 3 *bits* (8 níveis de intensidade) que contém vários objetos, uma linha e um ponto isolado. Adicionalmente também apresenta um perfil de intensidade aproximadamente no centro da imagem, tal como as derivadas de primeira e segunda ordem. Posto isto, observa-se as propriedades da derivadas da esquerda para a direita. Inicialmente a primeira derivada é diferente de zero no início e ao longo da rampa, enquanto a segunda derivada só é diferente de zero no início e fim da rampa. De seguida encontra-se um ponto isolado e verifica-se que a resposta à segunda derivada é muito maior que na primeira. O mesmo pode-se verificar no degrau.

Apesar das derivadas de segunda ordem terem respostas mais fortes na deteção de contornos, pode-se somente usar derivadas de primeira ordem. Mais uma vez, em processamento de imagem é usual a utilização de filtros. Existem máscaras que utilizam derivadas de primeira ordem e outras que além de derivada aplicam um operador gaussiano.

Um dos operadores mais comuns para deteção de contornos é o operador de *Sobel*. Este operador usa derivadas de primeira ordem (*gradiente*) para obter um filtro de convolução. O filtro é aplicado em duas direções para calcular a derivada na respetiva direção [13], [28], [29]:

$$\nabla f = gradiente(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.32}$$

As duas derivadas são então combinadas para produzir a magnitude (ou módulo) do gradiente, que representa a intensidade do contorno:

$$magnitude(\nabla f) = \sqrt{g_x^2 + g_y^2} \tag{2.33}$$

A direção do gradiente é dada pelo ângulo:

$$\alpha(x, y) = tg^{-1}\left(\frac{g_y}{g_x}\right) \tag{2.34}$$

O operador *Sobel* usa uma máscara *3×3*, as derivadas parciais em ordem a *x* e *y* são dadas por:

$$g_x = \frac{\partial f}{\partial x} = (p_7 + 2p_8 + p_9) - (p_1 + 2p_2 + p_3) \quad (2.35)$$

$$g_y = \frac{\partial f}{\partial y} = (p_3 + 2p_6 + p_9) - (p_1 + 2p_4 + p_7) \quad (2.36)$$

Onde *p* é a amplitude do *pixel* (consultar equação (*2*.26) e Figura 29).

−1	−2	−1
0	0	0
1	2	1

(a)

−1	0	1
−2	0	2
−1	0	1

(b)

Figura 34 – Máscara Sobel em ordem a x (a) e ordem a y (b).

Assim sendo, a máscara resultante para o operador *Sobel* é dada pela Figura 34.

Técnica do Limiar

A técnica do limiar é usada para separar uma imagem em duas regiões distintas com base num valor limite ou *threshold* (*T*). A técnica do limiar é frequentemente utilizada para separar o objeto de interesse do fundo da imagem, eliminando o ruído e as imperfeições. O processo de limiarização envolve comparar cada pixel com um valor limite e atribuir um valor binário (0 - preto ou 1 - branco) com base nessa comparação [13], [28].

Matematicamente, a imagem segmentada *g(x,y)* é dada por:

$$g(x,y) = \begin{cases} 1 & se\ f(x,y) > T \\ 0 & se\ f(x,y) \leq T \end{cases} \quad (2.37)$$

Quando o valor limite é uma constante aplicada a toda imagem, o processo refere-se como *global thresholding*, limiarização global ou binarização. Se a intensidade dos pixéis dos objetos e do fundo for suficientemente distinta, é possível usar apenas um valor limite, como apresentado em (*2*.37). Para estimar o valor limite usa-se o seguinte algoritmo [25]:

1. Recorrendo ao histograma da imagem (Figura 35), escolher um valor para *T* (por exemplo a média da intensidade dos *pixéis* da imagem);
2. Segmentar a imagem de acordo com (*2*.37). Desta operação surgem dois grupos G_1 para $> T$ e G_2 para $\leq T$;
3. Calcular as intensidades médias de cada grupo (m_1 e m_2);
4. Calcular o novo valor limite a partir de:

$$T = \frac{1}{2}(m_1 + m_2) \qquad (2.38)$$

5. Repetir os passos 2 a 4 até a diferença entre os valores de *T* em iterações sucessivas for menor que o valor ΔT predefinido (por exemplo $\Delta T = 0$). Este intervalo server para reduzir o tempo de computação.

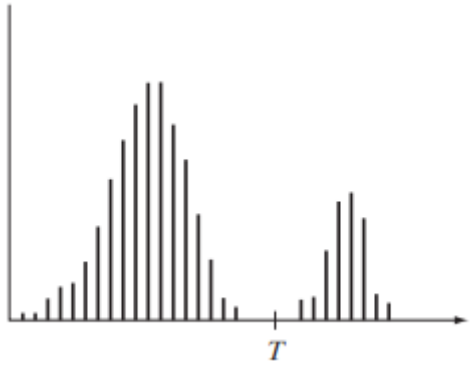


Figura 35 – Histograma de uma imagem com o valor limite T.

Regiões

A segmentação com base em regiões é outra técnica importante que divide uma imagem em regiões tendo em conta características comuns, como cor, textura ou intensidade. Os algoritmos de segmentação utilizam diferentes métodos, como o crescimento de regiões, e *clustering*, para separar a imagem em regiões significativas.

O crescimento de região tem como princípio agrupar um grupo de pixéis, ou sub-regiões, em regiões maiores com base num critério predefinido. A abordagem básica é começar com um conjunto de pontos de referência (*seeds*) e a partir deles, anexar pontos vizinhos que tenham características semelhantes à referência. A seleção de um *critério de*

semelhança (ou inclusão) depende não só do problema, mas também da informação que consta na imagem.

Um algoritmo típico de crescimento de região para imagens pode ser declarado da seguinte forma [13], [28]:

1. A partir de imagem de entrada *f(x,y)*, criar uma imagem *S(x,y)*, com as *seeds* onde os pontos de interesse têm intensidade *1* e os restantes *0*;
2. Encontrar todos os pontos conectados em *S(x,y)* e converter cada grupo de pontos conectados (regiões) num único pixel;
3. A partir de um *critério de semelhança* prefinido *Q*, formar uma imagem $f_Q(x,y)$;

$$Q = \begin{cases} f_Q(x,y) = 1 & se\ a\ condição\ for\ verdadeira \\ f_Q(x,y) = 0 & caso\ contrário \end{cases}$$

4. Comparar cada pixel resultante do passo 2 a $f_Q(x,y)$. Atribuir um valor equivalente nos pixéis adjacentes (vizinhos) se ocorrer uma combinação de atributos.

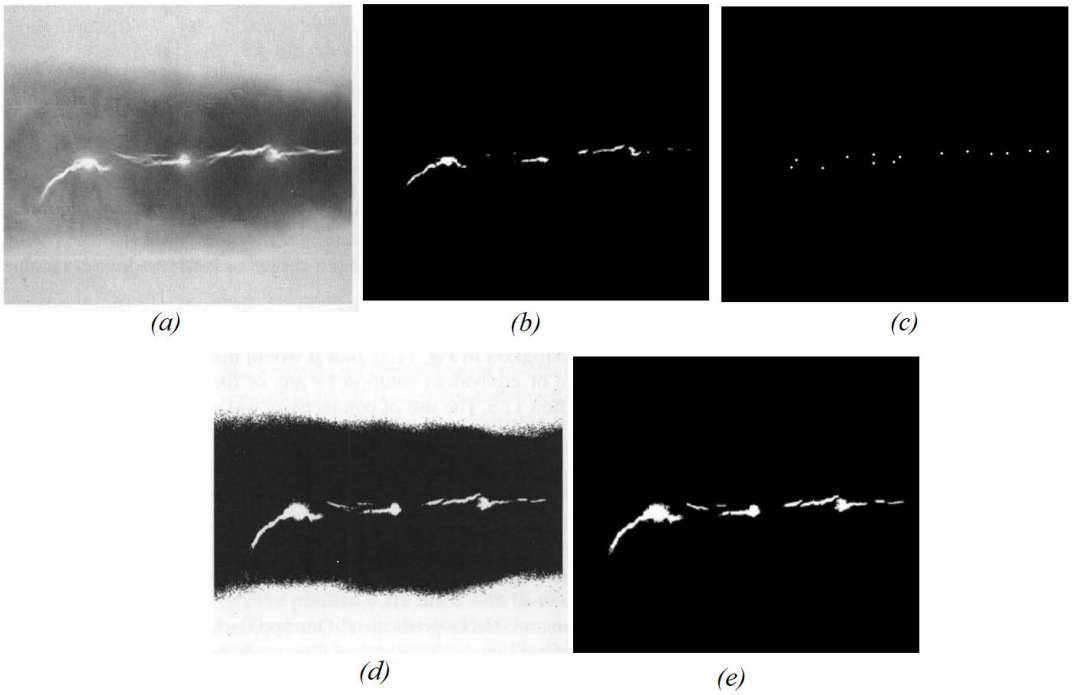


Figura 36 – Raio X de uma soldadura com defeito (a); Imagem com as seeds (b); Conversão de cada conjunto de seeds num único pixel (c); Imagem após aplicar o critério de semelhança (d); Imagem após comparar os pixéis vizinhos (e) [28].

A Figura 36 ilustra o processo de segmentação de um raio X de uma soldadura com pontos defeituosos (região a branco no centro de *(a)*). O primeiro passo foi identificar as *seeds*. Neste caso utilizou-se um *threshold* (*T = 254*) para isolar as regiões *(b)*. De seguida, realizou-se o segundo passo, converter cada região num único pixel *(c)*. A imagem *(d)*, é o resultado da aplicação do critério de semelhança. Neste caso a condição é verdadeira se $|f(x,y) - S(x,y)| \leq 64$. Por fim, ao realizar o quarto passo, obtém-se *(e)*. Note-se que em *(d)*, as extremidades da imagem foram consideradas como candidatos pelo critério de semelhança, o que não é ideal. No entanto, como não existiam conceções às *seeds* através de pontos vizinhos, foram desprezadas no resultado.

2.7 Robótica em Jogos

Diversos trabalhos foram encontrados na literatura que exploram a utilização de robôs em jogos. Foram identificados projetos que abordam a automação de jogos de tabuleiro por meio de robôs, como por exemplo o *Neo* um robô humanoide que joga o *jogo do galo* através de um sistema de visão e efetua a tomada de decisão com base no algoritmo *Minimax* [30]. Alternativamente, também foi encontrado um sistema que usa um manipulador robótico para jogar *damas*. Á semelhança do anterior este também usa técnicas de processamento de imagem para avaliar o jogo, no entanto, para tomar decisão recorre a técnicas de paralelismo em árvores de decisão com base no algoritmo *NegaMax*. Este paralelismo é o processo de dividir problemas maiores em partes menores, independentes e geralmente semelhantes que podem ser executados em simultâneo por vários processadores. Quando aplicado a jogos reduz tempo para chegar à melhor solução [31].

No contexto específico do jogo do Quatro em Linha, foi encontrada uma solução que consistia num atuador linear situado sobre o tabuleiro que deslizada ao longo do mesmo para colocar as peças na respetiva posição. Contrariamente às soluções anteriores, esta aplicação utilizava sensores em cada posição para saber se a peça foi colocada em vez de um sistema de visão. Para o algoritmo de decisão foi utilizado o *NegaMax* [32].

3 Desenvolvimento do Trabalho

Neste Capítulo, serão apresentados os procedimentos adotados para a realização desta dissertação, dos quais incluem a escolha do manipulador robótico, o estudo da cinemática direta e inversa desse manipulador, a aplicação do processamento de imagem para o jogo "Quatro em Linha" e finalmente, o algoritmo *Minimax*, que é utilizado para a teoria de jogos e será aqui aplicado.

A escolha de um manipulador robótico é uma etapa crucial no desenvolvimento de um projeto. É necessário garantir que o mesmo se adequa às tarefas propostas e garante um desempenho eficiente. Além disso, o estudo da cinemática direta e inversa do manipulador é essencial para entender como o robô se movimenta e como pode ser controlado de forma precisa.

De forma a dar inteligência artificial ao manipulador utilizou-se processamento de imagem para ajudar a tomar decisões. Numa primeira fase é necessário fazer uma observação e interpretação do ambiente. Através de algumas técnicas de processamento de imagem é possível dar ao manipulador as instruções para realizar a tarefa proposta.

Com base na informação extraída através das imagens será utilizado o algoritmo *Minimax*, que é amplamente utilizado em jogos para determinar a melhor jogada a ser realizada. Este algoritmo será aplicado ao jogo "Quatro em Linha".

3.1 Escolha do robô

Ao escolher um manipulador robótico para um projeto, é importante considerar as características específicas de cada modelo. Neste caso, poderiam ser consideradas duas opções: o *Dobot Magician* e o *Niryo Ned2* (Figura 37). Ambos disponíveis no ISEL, são modelos que oferecem recursos interessantes, mas ao analisar com mais rigor, o *Niryo Ned2* destaca-se por algumas características específicas que o tornam a melhor opção.

Uma das principais diferenças entre os dois manipuladores é o número de graus de liberdade que possuem. Enquanto o *Dobot Magician* tem apenas 4, o *Niryo Ned2* possui 6. Isso significa que o *Niryo Ned2* pode realizar tarefas mais complexas e precisas, incluindo tarefas que exigem maior alcance e movimentos mais delicados. Outra característica importante é a força de tração que o manipulador é capaz de gerar. Enquanto o *Dobot Magician* tem uma força de tração de 500g, o *Niryo Ned2* pode gerar uma força de tração de até 300g. Isso é especialmente importante em projetos que envolvem a manipulação de objetos pesados ou resistentes, como em processos industriais ou montagem de equipamentos. No entanto, esta característica não é um fator determinante, pois as peças utilizadas pesavam menos de 2g cada.

Além disso, o *Niryo Ned2* está equipado com uma câmara *RGB*, permitindo o reconhecimento de objetos e o processamento de imagens, tornando-o mais versátil em aplicações que exigem a interação com o ambiente externo. Possui ainda, uma interface de programação intuitiva, que torna a programação e o controlo do robô mais fácil [33], [34].

Embora o *Dobot Magician* seja uma escolha viável, as suas limitações em relação ao número de graus de liberdade e falta de câmara tornam-no menos adequado para projetos mais complexos. Além disso, as experiências desenvolvidas quer com o *MATLAB*, quer com o *Python*, não permitiam ter um controlo perfeito do manipulador. O *Niryo Ned2*, por outro lado, é capaz de lidar com tarefas mais desafiantes, o protocolo é mais transparente e adequado para trabalhar com a linguagem *Python* e oferece um maior potencial para a realização de projetos de maior escala e relevância. Desta forma, foi escolhido o *Niryo Ned2*.

(a) *(b)*

Figura 37 – Dobot Magician (a). Niryo Ned2 (b).

Especificações do *Niryo Ned2*

A partir do *Manual de Utilizador Ned2* é possível retirar as características mais importantes [34]:

Tabela 1 – Características do Manipulador.

Graus de liberdade	6 juntas rotacionais
Alcance de juntas [rad]	$-2{,}95 \leq \text{Junta } 1 \leq 2{,}95$ $-2{,}09 \leq \text{Junta } 2 \leq 0{,}61$ $-1{,}34 \leq \text{Junta } 3 \leq 1{,}57$ $-2{,}09 \leq \text{Junta } 4 \leq 2{,}09$ $-1{,}92 \leq \text{Junta } 5 \leq 1{,}92$ $-2{,}53 \leq \text{Junta } 6 \leq +2{,}53$
Alcance [mm]	440
Carga útil [g]	300
Repetibilidade [mm]	0,5
Precisão [mm]	0,5
Comunicação	Modbus TCP (master); TCP/IP
Ambiente de Programação	Niryo Studio; C++; PyNiryoRos; PyNiryo; ROS

As 6 juntas do *Niryo Ned2* são rotacionais, mas como referido no Capítulo 2.3.2, podem ser de diferentes categorias. Neste caso, as juntas J1, J4 e J6 são de torção, enquanto as restantes são rotacionais. A Figura 38 ilustra os eixos e o sentido de movimento positivo de cada uma.

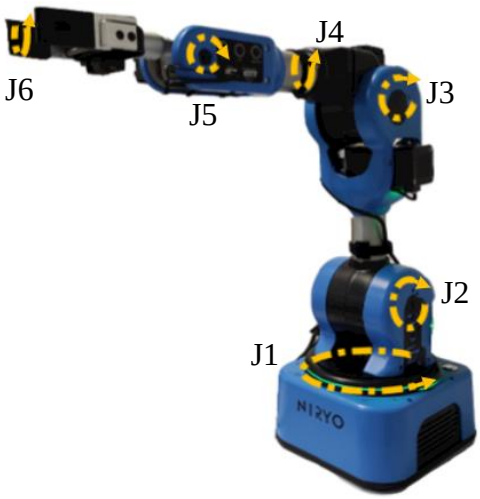


Figura 38 – Representação do movimento das juntas do Niryo Ned2.

A Figura 39 apresenta o esquema do manipulador, onde se pode extrair os comprimentos dos elos, necessários para o estudo da cinemática a ser realizado mais à frente. Ainda assim há duas medidas essenciais para o estudo da cinemática que não estão representadas na documentação. Estas medidas são a base do triângulo na Figura 41 (b) e o comprimento de a_5, que foram obtidas através do modelo *URFD* (Unified Robotics Description Format) do manipulador [35].

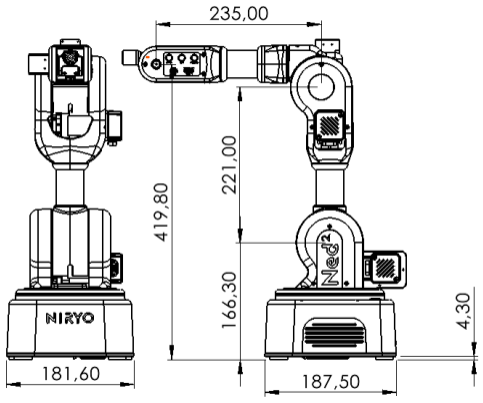


Figura 39 – Esquema com os comprimentos dos elos do manipulador [34].

Por fim, a comunicação com o manipulador será feita através do modo *Hotspot* (o robô fornece a própria rede *Wi-Fi*), enquanto o ambiente de programação a utilizar, para executar comandos, será o *python*, com recurso à *API* do fabricante *PyNiryo* [36].

3.2 Estudo da cinemática

Com a escolha do robô determinada é possível agora efetuar o estudo da cinemática direta e inversa. Nesta etapa, irão ser apresentadas as fórmulas e algoritmos utilizados para calcular a posição e orientação do efetor terminal com base nos ângulos das juntas, assim como determinar os ângulos necessários para alcançar uma determinada posição desejada.

Apesar de na prática os comandos para o manipulador serem executados através da *API* fornecida pelo fabricante, é importante realizar este estudo para entender o comportamento do robô. Adicionalmente, se não houvesse esta *API*, estes cálculos seriam fundamentais para a comunicação série com o manipulador.

Cinemática Direta

O primeiro passo para calcular a cinemática direta é estabelecer um sistema de coordenadas para cada um dos elos. Para isto, colocou-se o manipulador com todas as juntas recuadas, ou seja, todas com ângulo zero. Assim sendo, seguindo os espaços enunciados no Capítulo 2.5.4 obtém-se o sistema de coordenadas para o manipulador robótico.

Inicialmente, define-se o referencial da base R_0, onde o seu eixo Z_0 (azul) coincide com o eixo de movimento da primeira junta. O eixo X_0 (vermelho) aponta para a frente e o eixo Y_0 (verde) completa o sistema de acordo com a regra da mão direita. De seguida, atribui-se os restantes eixos Z_i que coincidem com o eixo de movimento das juntas $i+1$ (Figura 40).

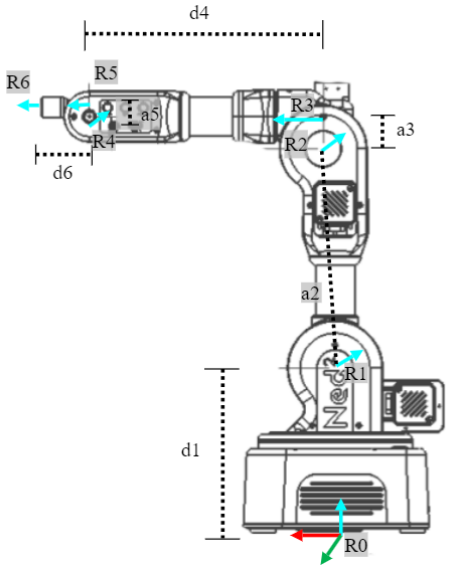
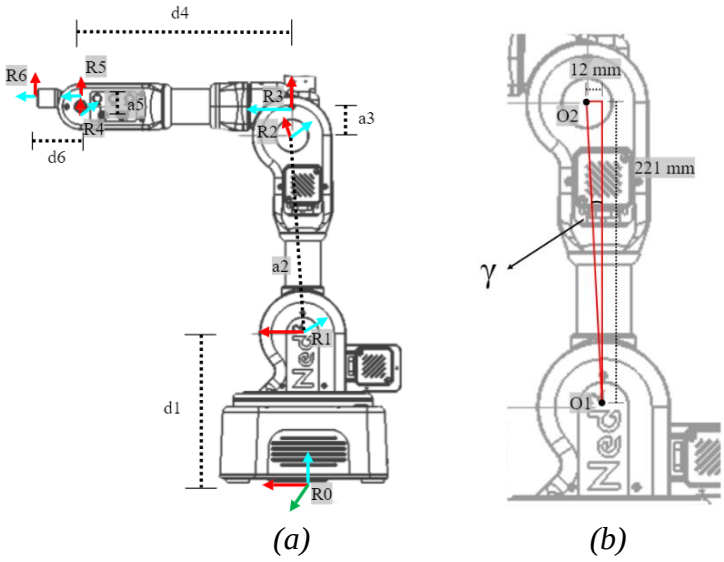


Figura 40 – Referencial base e eixos Z_i.

Na Figura 41 (a), os eixos X_i são atribuídos de forma a que sejam paralelos ao eixo X_{i-1}. No entanto existem casos em que isso não é possível, como se pode ver em R_2 (junta 3), que o eixo X_2 tem de apontar ao longo da normal comum que interseta Z_1 e Z_2. Isto significa que haverá um desfasamento φ_2 entre X_1 e X_2 e um desfasamento φ_3 entre X_2 e X_3. Tanto o desfasamento como o comprimento de a_2 são obtidos através da Figura 41 (b), resultando nas seguintes equações:

$$\gamma = \tan^{-1}\frac{12}{221} = 3{,}1°$$
$$\varphi_2 = 90 - \gamma = 90 - 3{,}1 = 86{,}9°$$
$$a_2 = \sqrt{221^2 + 12^2} = 221{,}3\ mm$$
$$\varphi_3 = \gamma = 3{,}1° \qquad (3.1)$$



(a) *(b)*

Figura 41 – Atribuição dos eixos X_i (a). Obtenção dos desfasamentos φ_2, φ_3 e o comprimento de a_2 (b).

Por fim, são atribuídos os eixos Y_i de acordo com a regra da mão direita. O último referencial coloca-se na ponta da garra, com o eixo Z_6 com sentido da aproximação, Y_6 coincide com o movimento de abertura/fecho da garra e X_6 completa o sistema de eixos de acordo com a regra da mão direita.

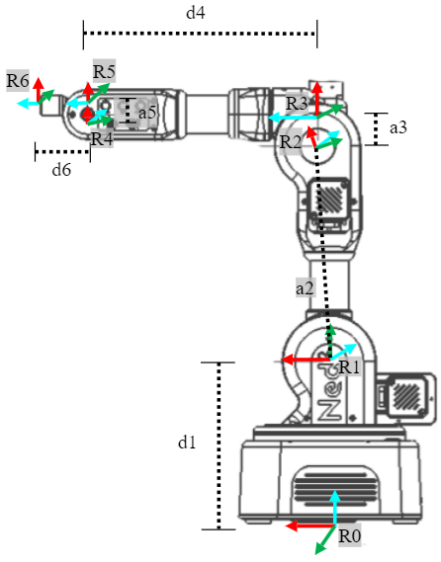


Figura 42 - Atribuição dos eixos Y_i.

Após a determinação dos referenciais consegue-se obter a tabela com os parâmetros *Denavit-Hartenberg*:

Tabela 2 – Tabela Denavit-Hartenberg.

Junta i	θ_i	d_i [mm]	a_i [mm]	α_i [rad]
1	θ_1	166,3	0,0	+ π/2
2	$\theta_2 + \varphi_2$	0,0	221,3	0
3	$\theta_3 + \varphi_3$	0,0	32,5	+ π/2
4	θ_4	235,0	0,0	- π/2
5	θ_5	0,0	9,2	+ π/2
6	θ_6	132,0	0,0	0

Depois de obter a tabela de parâmetros DH, com recurso à equação (*2*.13), pode-se obter a sequência de matrizes de transformação. Ao multiplicar as matrizes de transformação homogênea ao longo da cadeia cinemática do robô, podemos obter a matriz de transformação homogênea final, que representa a posição e a orientação do efetor terminal em relação ao sistema de coordenadas da base ($^0A_6$).

O seguinte conjunto da matrizes representa o referencial da junta *i* em relação ao referencial da base (^0A_i).

$$^0A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 166{,}3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^0A_2 = \begin{bmatrix} 0{,}05 & -0{,}99 & 0 & 11{,}9 \\ 0 & 0 & 0 & 0 \\ 0{,}99 & 0{,}05 & 0 & 386{,}9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^0A_3 = \begin{bmatrix} 0 & 0 & 1 & 11{,}9 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 419{,}4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^0A_4 = \begin{bmatrix} 0 & -1 & 0 & 246{,}9 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 419{,}4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^0A_5 = \begin{bmatrix} 0 & 0 & 1 & 246.9 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 428{,}6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^0A_6 = \begin{bmatrix} 0 & 0 & 1 & 378{,}9 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 428{,}6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Para facilitar o processo de cálculo, foi realizado um algoritmo em *MATLAB* com recurso à *Robotics Toolbox* desenvolvida por *Peter Corke* [37]. Desta forma pode-se calcular com

facilidade todas as matrizes de transformação. Além disso, esta ferramenta tem capacidade de realizar uma ilustração gráfica do manipulador robótico (Figura 43).

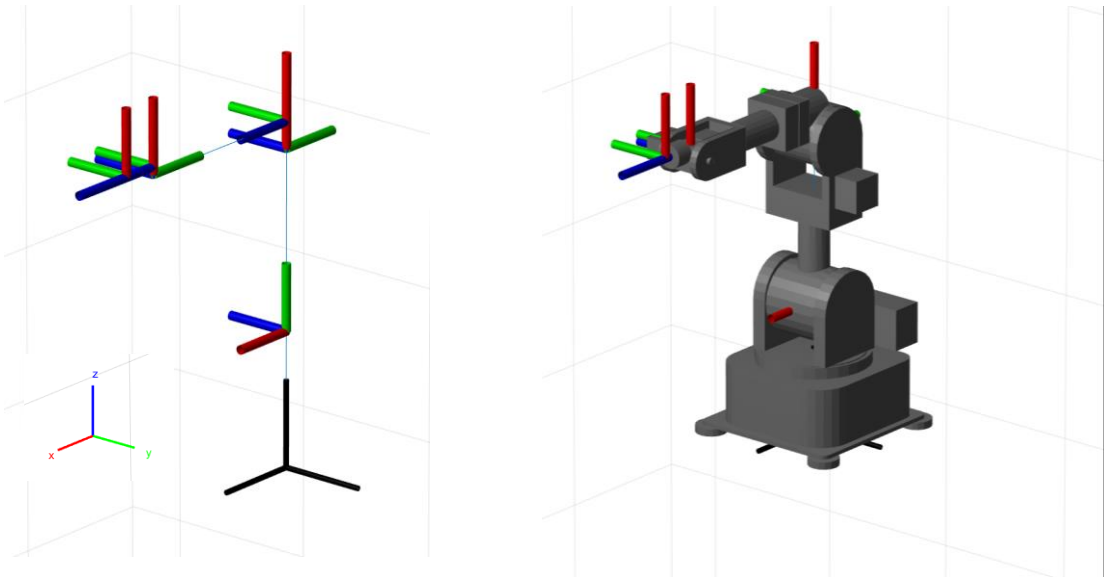


Figura 43 – Simulação do manipulador na posição de repouso.

Desta forma, com base na posição das juntas, podemos determinar a localização espacial da garra do manipulador. Isto é crucial para controlar o movimento do robô e garantir que ele executa as tarefas desejadas com precisão e eficiência.

Cinemática Inversa

Na robótica, a cinemática inversa é necessária para o planeamento da trajetória do robô e é um dos problemas mais complicados da robótica. As duas principais maneiras de resolver o problema da cinemática inversa são através de métodos analíticos (como o apresentado no Capítulo 2.5.5) ou através de métodos numéricos. As soluções analíticas existem apenas para alguns tipos de manipuladores, nomeadamente aqueles que possuem três juntas cujos eixos intersetam num ponto comum. Manipuladores com um punho esférico, isto é, os eixos das últimas três juntas intersetam num ponto comum, são muito utilizados precisamente para garantir a existência de uma solução analítica. No entanto, algumas aplicações industriais requerem especificações, como amplitudes de movimento do punho que os manipuladores anteriores não conseguem cumprir. De forma a cumprir os requisitos são utilizados manipuladores com *offsets* nos punhos. Contudo, a cinemática inversa torna-se bastante mais complicada de obter por métodos analíticos, podendo até

mesmo ser impossível. Na ausência de uma solução analítica, os modelos interativos numéricos são formas de obter a cinemática inversa de manipuladores sem punho esférico. Dentro desta categoria existem dois principais tipos de técnicas. A primeira faz uso do método *Newton-Raphson* para estimar a raiz da função de erro, e baseia-se no Jacobiano inverso. O segundo utiliza diferentes algoritmos baseados no gradiente de descida para encontrar o mínimo da função [38], [39].

Através do estudo da cinemática direta, verifica-se que as três últimas juntas não intersetam num ponto comum, porque existe um desvio de comprimento a_5 entre as juntas 5 e 6. Isto significa que o punho não é esférico e o método analítico é extremamente complexo de obter. Contudo, será apresentada uma solução analítica para uma situação onde este manipulador apresenta-se um punho esférico, ou seja, $a_5 = 0$.

Em primeiro lugar, pode-se tabular os vários parâmetros, onde d_i e a_i são as constantes diferentes de zero (da tabela anterior) e t_i são as variáveis de junta a determinar.

Tabela 3 – Tabela Denavit-Hartenberg com incógnitas.

Junta i	θ_i	d_i	a_i	α_i [rad]
1	t_1	d_1	0	+ π/2
2	t_2	0	a_2	0
3	t_3	0	a_3	+ π/2
4	t_4	d_4	0	- π/2
5	t_5	0	$a_5 = 0$	+ π/2
6	t_6	d_6	0	0

Em segundo lugar, aplica-se novamente a equação (*2*.13) aos parâmetros da tabela anterior. O resultado será um conjunto de matrizes de transformações em função de t_i. Para simplificar a apresentação das funções, abreviou-se os senos e cossenos do angulo t_i tal que:

$$\begin{aligned} sin(t_i) &= s_i \\ cos(t_i) &= c_i \\ sin(t_i + t_j) &= s_{ij} \\ cos(t_i + t_j) &= c_{ij} \end{aligned} \qquad (3.2)$$

Tendo por base as equações (*2*.21) a (*2*.23), com *n = 6*, é possível obter o valor da primeira junta dado que:

$$
{}^{1}A_6 = \begin{bmatrix} -c_6(s_{23}s_5 - c_{23}c_4c_5) - c_{23}s_4s_6 & s_6(s_{23}s_5 - c_{23}c_4c_5) - c_{23}c_6s_4 \\ c_6(c_{23}s_5 + s_{23}c_4c_5) - s_{23}s_4s_6 & -s_6(c_{23}s_5 + s_{23}c_4c_5) - s_{23}c_6s_4 \\ c_4s_6 + c_5c_6s_4 & c_4c_6 - c_5s_4s_6 \\ 0 & 0 \end{bmatrix} \tag{3.3}
$$

$$
\begin{matrix} s_{23}c_5 + c_{23}c_4c_5 & a_3c_{23} + d_4s_{23} + a_2c_2 + d_6s_{23}c_5 + c_4s_5c_{23} \\ s_{23}c_4s_5 - c_{23}c_5 & a_3s_{23} - d_4c_{23} + a_2s_2 - d_6c_{23}c_5 - c_4s_5s_{23} \\ s_4s_5 & d_6s_4s_5 \\ 0 & 1 \end{matrix} \Bigg]
$$

E que a multiplicação entre ${}^{1}A_0$ e ${}^{0}T_6$, resulta em:

$$
{}^{1}T_6 = \begin{bmatrix} n_xc_1 + n_ys_1 & o_xc_1 + o_ys_1 & a_xc_1 + a_ys_1 & p_xc_1 + p_ys_1 \\ n_z & o_z & a_z & p_z - d_1 \\ n_xs_1 - n_yc_1 & o_xs_1 - o_yc_1 & a_xs_1 - a_yc_1 & p_xs_1 - p_yc_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}
$$

$$
= \begin{bmatrix} A & B & C & D \\ n_z & o_z & a_z & E \\ F & G & H & I \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

A partir desta primeira iteração, pode-se extrair, através da igualdade dos elementos (3,3) e (3,4) de cada matriz o valor para a primeira junta,

$$
\begin{cases} H = s_4s_5 \\ I = d_6s_4s_5 \end{cases} \tag{3.5}
$$

Substituindo na segunda equação s_4s_5 por *H*:

$$
p_xs_1 - p_yc_1 = d_6a_xs_1 - d_6a_yc_1 \tag{3.6}
$$

Dividindo os dois lados por $c_1(p_x\text{-}d_6a_x)$:

$$
\frac{s_1(p_x - d_6a_x)}{c_1(p_x - d_6a_x)} = \frac{c_1(p_Y - d_6a_y)}{c_1(p_x - d_6a_x)} \tag{3.7}
$$

Ao aplicar a inversa da tangente consegue-se obter o valor para a primeira variável de junta:

$$t_1 = \tan^{-1}\left(\frac{p_y - d_6 a_y}{p_x - d_6 a_x}\right) \tag{3.8}$$

Em vez da expressão anterior, utiliza-se a função atan2, que calcula o arco tangente e define o valor do ângulo entre -π e π, e permite maior precisão. Neste caso, a primeira junta tem duas soluções posições, com uma diferença de 180º.

$$\begin{cases} t_1 = atan2(p_y - d_6 a_y, p_x - d_6 a_x) \\ t_1' = t_1 \pm \pi \end{cases} \tag{3.9}$$

Com primeira junta determinada, efetua-se a segunda iteração, multiplicando à esquerda em ambos os termos, pela matriz de transformação seguinte:

$${}^2T_6 = {}^2A_1\,{}^1T_6 = {}^2A_3\,{}^3A_4\,{}^4A_5\,{}^5A_6 \tag{3.10}$$

Assim sendo, as matrizes de transformação são dadas por:

$${}^2A_6 = \begin{bmatrix} -c_6(s_3 s_5 - c_3 c_4 c_5) - c_3 s_4 s_6 & s_6(s_3 s_5 - c_3 c_4 c_5) - c_3 c_6 s_4 \\ c_6(c_3 s_5 - s_3 c_4 c_5) - s_3 s_4 s_6 & -s_6(c_3 s_5 - s_3 c_4 c_5) - c_6 s_3 s_4 \\ c_4 s_6 + c_5 c_6 s_4 & c_4 c_6 - c_5 s_4 s_6 \\ 0 & 0 \end{bmatrix} \tag{3.11}$$

$$\begin{matrix} c_5 s_3 + c_3 c_4 s_5 & d_6(c_5 s_3 + c_3 c_4 s_5) + a_3 c_3 + d_4 s_3 \\ c_4 s_3 s_5 - c_3 c_5 & a_3 s_3 - d_4 c_3 - d_6(c_3 c_5 - c_4 s_3 s_5) \\ s_4 s_5 & d_6 s_4 s_5 \\ 0 & 1 \end{matrix}\Bigg]$$

$${}^2T_6 = \begin{bmatrix} n_z s_2 + c_2 A & o_z s_2 + c_2 B & a_z s_2 + c_2 C & c_2 D - s_2 E - a_2 \\ n_z c_2 - s_2 A & o_z c_2 - s_2 B & a_z c_2 - s_2 C & -s_2 D - c_2 E \\ F & G & H & I \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

Da igualdade dos elementos (1,3), (1,4), (2,3) e (2,4) de cada matriz resulta em:

$$\begin{cases} c_5 s_3 + c_3 c_4 s_5 = a_z s_2 + c_2 C \\ d_6(c_5 s_3 + c_3 c_4 s_5) + a_3 c_3 + d_4 s_3 = c_2 D - s_2 E - a_2 \\ c_4 s_3 s_5 - c_3 c_5 = a_z c_2 - s_2 C \\ a_3 s_3 - d_4 c_3 - d_6(c_3 c_5 - c_4 s_3 s_5) = -s_2 D - c_2 E \end{cases} \tag{3.13}$$

$$\begin{cases} d_6 a_z s_2 + d_6 c_2 C + a_3 c_3 + d_4 s_3 = c_2 D - s_2 E - a_2 \\ a_3 s_3 - d_4 c_3 + d_6 a_z c_2 - d_6 s_2 C = -s_2 D - c_2 E \end{cases} \quad (3.14)$$

$$\begin{cases} a_3 c_3 + d_4 s_3 = -c_2 Y - s_2 X - a_2 \\ a_3 s_3 - d_4 c_3 = s_2 Y - c_2 X \end{cases} \quad (3.15)$$

Onde:

$$\begin{aligned} X &= E + d_6 a_z \\ Y &= d_6 C - D \end{aligned} \quad (3.16)$$

Elevando cada uma das equações ao quadrado e de seguida efetuar uma igualdade obtém-se:

$$a_3 + d_4 - Y^2 - X^2 - a_2^2 = 2Y a_z c_2 + 2X a_2 s_2 \quad (3.17)$$

Para simplificar a expressão,

$$\begin{aligned} K_1 &= 2Y a_2 \\ K_2 &= 2X a_2 \\ K_3 &= a_3 + d_4 - Y^2 - X^2 - a_2^2 \end{aligned} \quad (3.18)$$

Substituindo em *(3.17)* e isolando o termo associado ao cosseno,

$$K_1 c_2 = K_3 - K_2 s_2 \quad (3.19)$$

Elevando ambos os termos ao quadrado:

$$K_1^2 c_2^2 = K_1^2 (1 - s_2^2) = K_3^2 - 2K_2 K_3 s_2 + K_2^2 s_2^2 \quad (3.20)$$

Ao colocar todos os termos do mesmo lada da equação,

$$(K_3^2 - K_1^2) - 2K_2 K_3 s_2 + (K_1^2 + K_2^2) s_2^2 = 0 \quad (3.21)$$

Com a equação nesta forma, pode-se aplicar a fórmula resolvente a uma equação de segundo grau, que resulta em:

$$s_2 = \frac{K_2 K_3 \pm K_1 \sqrt{K_1^2 + K_2^2 - K_3^2}}{K_1^2 + K_2^2} \quad (3.22)$$

Efetuando o mesmo passo que se realizou em *(3.19)*, mas isolando o termo com seno, obtém-se:

$$c_2 = \frac{K_1 K_3 \pm K_2\sqrt{K_1^2 + K_2^2 - K_3^2}}{K_1^2 + K_2^2} \quad (3.23)$$

$$t_2 = atan2(s_2, c_2) \quad (3.24)$$

Ao analisar as equações *(3.22)* e *(3.23)*, verifica-se que podem existir diferentes soluções tendo em conta o sinal utilizado. A escolha do sinal depende da opção escolhida para a junta anterior.

Para obter o valor a terceira junta, recorre-se a uma equação do tipo 3 [10], a partir de *(3.15)*:

$$\begin{cases} a_3 c_3 + d_4 s_3 = W_1 \\ a_3 s_3 - d_4 c_3 = W_2 \end{cases} \quad (3.25)$$

Multiplicando a primeira equação por a_3 e a segunda por d_4, resulta em:

$$c_3 = \frac{d_4 W_2 - W_1 a_3}{-a_3^2 - d_4^2} \quad (3.26)$$

Novamente, em *(3.25)* multiplicando a primeira equação por d_4 e a segunda por a_3, resulta em:

$$s_3 = \frac{a_3 W_2 + W_1 d_4}{a_3^2 + d_4^2} \quad (3.27)$$

$$t_3 = atan2(s_3, c_3) \quad (3.28)$$

Para obter o ângulo da quarta junta, efetua-se a terceira iteração que resulta em:

$${}^3A_6 = \begin{bmatrix} c_4c_5c_6 - s_4s_6 & c_6s_4 - c_4c_5c_6 & c_4s_5 & d_6c_4s_5 \\ c_4s_6 + s_4c_5c_6 & c_4c_6 - c_5s_4s_6 & s_4s_5 & d_6s_4s_5 \\ -c_6s_5 & s_5s_6 & c_5 & d_4 + d_6c_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.29)$$

Igualando os elementos (1,3) e (2,3) de cada matriz, consegue-se extrapolar o valor para a quarta junta.

$$\begin{cases} c_4 s_5 = c_3(a_z s_2 + c_2 C) + s_3(a_Z c_2 - s_2 C) \\ s_4 s_5 = H \end{cases} \quad (3.30)$$

Dividindo a segunda equação pela primeira:

$$\frac{s_4 s_5}{c_4 s_5} = \frac{H}{c_3(a_z s_2 + c_2 C) + s_3(a_Z c_2 - s_2 C)} \tag{3.31}$$

Desta forma, consegue-se obter duas soluções para a junta 4:

$$\begin{cases} t_4 = atan2(H, c_3(a_z s_2 + c_2 C) + s_3(a_Z c_2 - s_2 C)) \\ t_4' = t_4 + \pi \end{cases} \tag{3.32}$$

Note-se que se poderia retirar o valor para a quinta junta a partir dos elementos (3,3) através da função arco cosseno, mas esta solução não seria tão precisa, para além do sinal do ângulo ser desconhecido. Para além disso, os elementos (3,1) e (3,2) também dariam uma possível solução para a última junta, no entanto, não têm em conta o valor da quarta junta.

Apesar de não serem utilizados na solução final, o processo de cálculo é descrito nas seguintes equações:

$$\begin{cases} c_5 = s_3(a_z s_2 + c_2 C) - c_3(a_Z c_2 - s_2 C) \\ s_5 = \pm\sqrt{1 - c_5^2} \end{cases} \tag{3.33}$$

$$\frac{s_5 s_6}{c_6 s_5} = \frac{c_3(s_2 B - c_2 o_z) + s_3(s_2 o_z - c_2 B)}{-c_3(s_2 A - c_2 n_z) - s_3(s_2 n_z + c_2 A)} \tag{3.34}$$

Portanto, como é preferível utilizar a função atan2, efetua-se a quarta iteração e obtém-se:

$${}^4A_6 = \begin{bmatrix} c_5 c_6 & -c_5 s_6 & s_5 & d_6 s_5 \\ c_6 s_5 & c_4 c_6 - c_5 s_4 s_6 & -c_5 & -d_6 c_5 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.35}$$

$${}^4T_6 = \begin{bmatrix} c_4(c_3(n_z s_2 + c_2 A) - s_3(s_2 A - c_2 n_z) + s_4 F \\ -c_3(s_2 A - c_2 n_z) - s_3(s_2 n_z + c_2 A) \\ -s_4(c_3(s_2 n_z + c_2 A) - s_3(s_2 A - c_2 n_z)) + c_4 F \\ 0 \end{bmatrix}$$

$$\begin{matrix} c_4(c_3(o_z s_2 + c_2 B) - s_3(s_2 B - o_z c_2)) + s_4 G \\ -c_3(s_2 B - c_2 o_z) - s_3(s_2 o_z + c_2 B) \\ -s_4(c_3(s_2 o_z + c_2 B) - s_3(s_2 B - c_2 o_z) + c_4 G \\ 0 \end{matrix} \quad (3.36)$$

$$\begin{matrix} c_4(c_3(a_z s_2 + c_2 C) + s_3(a_Z c_2 - s_2 C)) + s_4 H \\ -s_3(a_z s_2 + c_2 C) + c_3(a_Z c_2 - s_2 C) \\ c_4 H - s_4(c_3(a_z s_2 + c_2 C) + s_3(a_Z c_2 - s_2 C)) \\ 0 \end{matrix}$$

$$\begin{matrix} s_4 I - c_4(a_3 + s_3(s_2 D + c_2 E) - c_3(a_2 + s_2 E - c_2 D)) \\ d_4 - c_3(s_2 D + c_2 E) + s_3(a_2 + s_2 E - c_2 D) \\ s_4\big(a_3 + s_3(s_2 D + c_2 E) + c_3(a_2 + s_2 E - c_2 D)\big) + c_4 I \\ 1 \end{matrix}$$

Com isto, consegue-se obter os valores para a quinta junta através da igualdade dos elementos (1,3) e (2,3).

$$\begin{cases} c_5 = s_3(a_z s_2 + c_2 C) - c_3(a_Z c_2 - s_2 C) \\ s_5 = c_4(c_3(a_z s_2 + c_2 C) + s_3(a_Z c_2 - s_2 C)) + s_4 H \end{cases} \quad (3.37)$$

$$t_5 = atan2(s_5, c_5) \quad (3.38)$$

Por fim, a sexta e última junta é obtida através dos elementos (3,1) e (3,2).

$$\begin{cases} c_6 = -s_4(c_3(s_2 o_z + c_2 B) - s_3(s_2 B - c_2 o_z) + c_4 G \\ s_6 = -s_4(c_3(s_2 n_z + c_2 A) - s_3(s_2 A - c_2 n_z)) + c_4 F \end{cases} \quad (3.39)$$

$$t_6 = atan2(s_6, c_6) \quad (3.40)$$

Uma vez determinadas as equações para obter os ângulos das juntas, pode-se arbitrar uma posição desejada para o efetor terminal, e criar uma solução. Sendo assim, considera-se que:

$${}^0T_6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & -120{,}0 \\ 1 & 0 & 0 & 196{,}0 \\ 0 & 0 & -1 & 250{,}0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.41)$$

O versor ***n*** indica que o eixo X_6 tem direção e sentido igual a Y_0, por sua vez o versor ***o*** indica que Y_6 tem direção e sentido igual a X_0. Por último, o versor ***a***, indica que o eixo Z_6 tem a mesma direção que Z_0, mas sentido oposto. Numa representação *Roll-Pitch-Yaw* a orientação da órgão terminal seria dada por *RPY(1,57;0;-3,14)*, com os valores dos

ângulos em radianos. Esta orientação irá corresponder à forma como a garra apanha e larga as peças.

Quanto a p_x, p_y, e p_z, estes valores representam as coordenadas cartesianas (em milímetros) *x*, *y* e *z*, respetivamente.

Ao substituir os valores nas equações determinadas anteriormente, obtêm-se os valores apresentados na Tabela 4.

Como se pode verificar, existem várias soluções possíveis para colocar o efetor terminal na posição e orientação desejada. Contudo, estes valores não têm em conta as restrições físicas do manipulador, assim, após analisar a Tabela 1 verifica-se que apenas a *solução A* se encontra dentro dos limites.

Tabela 4 – Conjunto de soluções possíveis de cinemática inversa (a_5 = 0).

Solução	θ_1 [rad]	θ_2 [rad]	θ_3 [rad]	θ_4 [rad]	θ_5 [rad]	θ_6 [rad]
A	2,12	0,08	-0,25	0,00	-1,41	0,55
B	2,12	0,08	-0,25	3,14	1,41	-2,59
C	2,12	-1,61	3,01	0,00	-2,96	0,55
D	2,12	-1,61	3,01	3,14	2,96	-2,59
E	-1,02	1,72	-0,25	0,00	-3,04	-2,59
F	-1,02	1,72	-0,25	-3,14	3,04	0,55
G	-1,02	0,02	3,01	0,00	1,68	-2,59
H	-1,02	0,02	3,01	-3,14	-1,68	0,55

Uma vez que o controlo do manipulador foi efetuado através da *API* dedicada para o *Niryo Ned2*, é interessante efetuar uma comparação entre os resultados de cada um dos métodos.

Tabela 5 – Comparação entre simulação e solução real.

Solução	θ_1 [rad]	θ_2 [rad]	θ_3 [rad]	θ_4 [rad]	θ_5 [rad]	θ_6 [rad]
A	2,12	0,08	-0,24	0,00	-1,41	0,55
Niryo Ned2	2,12	0,13	-0,28	0,00	-1,42	0,52

Ao analisar os resultados da tabela, pode-se observar uma discrepância nos ângulos das juntas 2, 3 e 6. Isto deve-se ao facto de não ter sido considerado o comprimento a_5 para esta solução analítica. A colocação dos valores da solução A num caso prático resultaria numa posição com coordenadas:

Tabela 6 – Posição obtida através do método analítico ($a_5 = 0$).

Coordenadas	Posição Desejada [mm]	Posição obtida com a solução A [mm]
X	-120,0	-124,8
Y	196,0	203,8
Z	250,0	249,9

Mesmo não tendo considerado o comprimento a_5, na Tabela 6 verifica-se que os resultados são próximos. Isto porque a_5 apenas mede 9,2 mm, quanto maior for este comprimento, maior a discrepância entre as coordenadas desejadas e as obtidas.

Tendo isto em conta, efetuou-se a cinemática inversa através de um método numérico. A *Robotics System Toolbox* integrada no MATLAB tem disponível alguns numéricos para o cálculo da cinemática inversa de manipuladores. Estes algoritmos são métodos iterativos de otimização baseados em gradientes que partem de uma estimativa inicial da solução, isto é, atribuem uma configuração de juntas aleatória e depois procuram minimizar uma função de custo específica. Se qualquer um dos algoritmos convergir para uma configuração onde o custo é próximo de zero dentro de uma tolerância especificada, significa que foi encontrada uma solução para o problema da cinemática inversa. Neste caso o método utilizado foi o algoritmo de projeção do gradiente *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) [40].

Impondo as restrições físicas das juntas do manipulador apresentadas na Tabela 1, obtém-se a seguinte solução para a cinemática inversa utilizando o método BFGS.

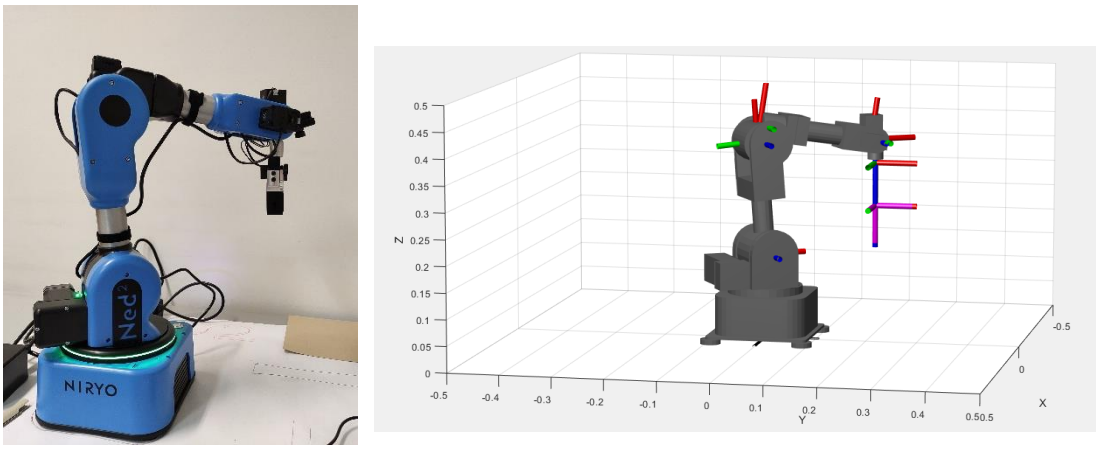
Tabela 7 - Comparação entre as diferentes soluções obtidas.

Solução	θ_1 [rad]	θ_2 [rad]	θ_3 [rad]	θ_4 [rad]	θ_5 [rad]	θ_6 [rad]
A	2,12	0,08	-0,25	0,00	-1,41	0,55
BFGS	2,12	0,12	-0,28	0,00	-1,41	0,55
Niryo Ned2	2,12	0,13	-0,28	0,00	-1,42	0,52

Analisando a Tabela 7, verifica-se que o método numérico alcançou a solução desejada.

O estudo teórico da cinemática inversa e direta ajuda a perceber o manipulador e consegue-se prever trajetórias mesmo antes de aplicar numa situação real.

Novamente, com recurso às propriedades do *MATLAB* e da *toolbox*, pode-se representar graficamente a posição do manipulador e efetuar uma comparação à solução real (Figura 44).



(a) *(b)*

Figura 44 – Posição do manipulador para as juntas da Tabela 7. Niryo Ned2 (a) e da simulação com o método BFGS (b).

Posto isto pode-se enunciar algumas diferenças entre os métodos numéricos e analíticos. Para alcançar uma solução, os métodos analíticos são bastantes exaustivos e difíceis, além disso, têm de ser adaptados para manipuladores com estruturas diferentes. No entanto, quando alcançada uma expressão, a computação de uma solução é extremamente rápida, além de ser possível obter todas as posições possíveis, o que não acontece nos métodos numéricos, onde só é apresentada uma solução assim que o erro atinge zero ou um valor

próximo. Além disso, nos métodos numéricos o número de iterações depende da estimativa inicial das juntas, quanto mais perto da solução for a estimativa, menos iterações são feitas.

Movimentação do Manipulador

Para realizar os objetivos propostos, o manipulador tem de executar uma série de movimentos que garantem uma operação eficiente e precisa. Assim, é crucial especificar as coordenadas e orientação corretas da garra do manipulador ao apanhar e largar as peças. Uma especificação imprecisa destes parâmetros pode levar a problemas significativos durante a realização da tarefa, tais como, colisões com a peça ou com outros objetos próximos, ou no caso de uma orientação inadequada da garra, pode levar a um desalinhamento durante o processo de apanhar e largar a peça, comprometendo a precisão e a estabilidade do processo.

Por estas razões, estabeleceu-se uma sequência para os movimentos a serem realizados por parte do manipulador:

1. Ao iniciar o jogo, o manipulador é colocado na posição *start*, onde todas as juntas estão recuadas ($\theta_i = 0^o$) e a garra encontra-se aberta;
2. Quando for a vez do robô jogar, este movimenta-se para uma posição onde fica com a garra alinhada com a peça, mas com uma certa altura para evitar colisões com objetos. A esta posição chamemos de *pick_pose_high*;
3. De seguida, a garra pode baixar para a posição *pick_pose* e apanhar a peça (fecha a garra);
4. Retorna novamente para a posição *pick_pose_high*;
5. Desloca-se para uma posição *drop_pose_high*, alinhada com o tabuleiro, mas ligeiramente acima de modo para evitar colisões;
6. A garra baixa para a posição *drop_pose* e larga a peça (abre a garra);
7. Por fim, retorna à posição *drop_pose_high*, e aguarda. Quando voltar a ser a sua jogada, o processo repete-se a partir do segundo passo.

3.3 Tratamento de Imagem

De forma a determinar a em que posição o robô vai largar a peça é necessário desenvolver um algoritmo, através do tratamento de imagem, para determinar o estado do tabuleiro e ser possível calcular a jogada a realizar.

Para isto, foi utilizado o *OpenCV*, que é uma biblioteca *open source* com várias funções e algoritmos otimizados com técnicas de processamento de imagem. Esta ferramenta está disponível para várias linguagens de programação, mas para este caso será usado o *Python*.

Para obter as imagens a serem processadas utilizou-se uma *webcam Trust Trino HD* (Figura 45). Esta câmara estará ligada a um computador via *USB* e é capaz de transmitir 30 imagens a cores por segundo, com uma resolução de *1280x720* pixéis e ângulo de visão de 52º [41]. Optou-se por utilizar esta câmara face à que vinha com o robô, assim garante-se que a tomada de decisão será mais precisa, pois, a câmara está sempre a apontar para o tabuleiro, o que não aconteceria caso fosse utilizada a câmara do manipulador.

Figura 45 – Câmara Trust Trino HD.

Deteção do Tabuleiro

O tabuleiro consiste numa caixa em madeira com seis linhas e sete colunas. Inicialmente o tabuleiro encontra-se vazio, visto que nenhuma jogada foi realizada. É nesta fase que se retira a informação útil do tabuleiro, ou seja, as possíveis posições onde podem existir peças. Portanto, o primeiro passo é identificar as coordenadas centrais de cada uma dessa

posições na imagem. Existem várias formas de realizar esta tarefa, a solução escolhida usa a técnica do limiar explicada no Capítulo 2.6.4.

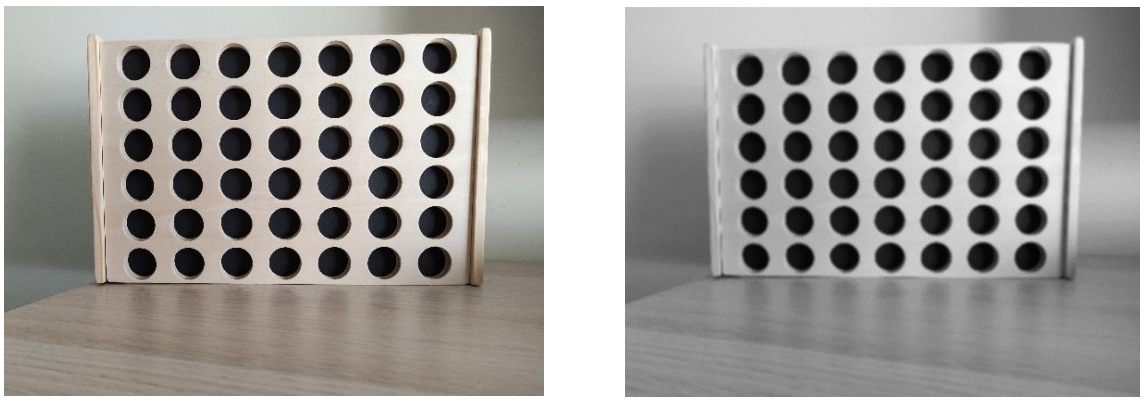
Antes de se definir um *threshold* converteu-se a imagem *RGB* numa imagem em tons de cinza (*grayscale*), na biblioteca *OpenCV*, esta conversão é dada por:

$$grayscale = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \qquad (3.42)$$

De seguida, aplica-se um filtro gaussiano de 5x5 (Figura 46), para alisar a imagem e remover ruído [42]. Na Figura 47 (a) e (b), é possível ver a imagem original em RGB e a imagem resultante do tabuleiro em tons de cinza e com o filtro.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

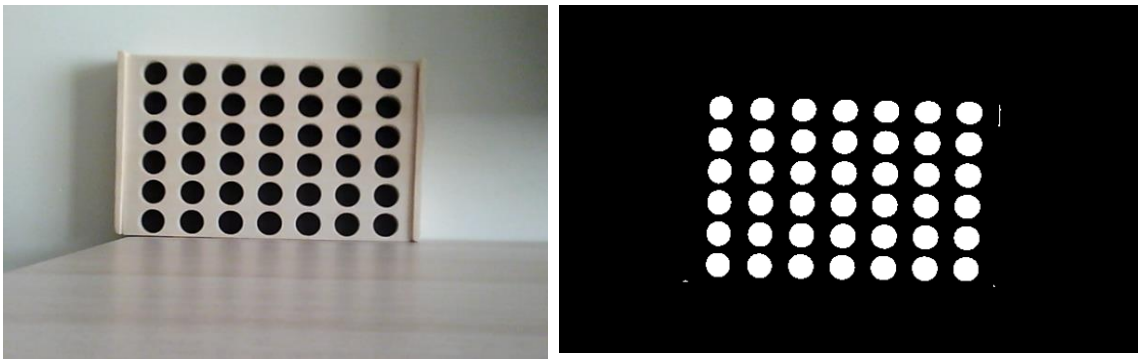
Figura 46 – Máscara com filtro gaussiano 5x5.



(a) (b)

Figura 47 – Imagem original (a). Imagem após conversão para tons de cinza e aplicação do filtro gaussiano (b).

Finalmente, aplicou-se um *threshold* inverso, isto significa que os valores com intensidade abaixo do valor limite, *T*, terão intensidade 1, enquanto os valores superiores terão valor 0. Após testar várias hipóteses definiu-se *T = 40*.



(a) (b)

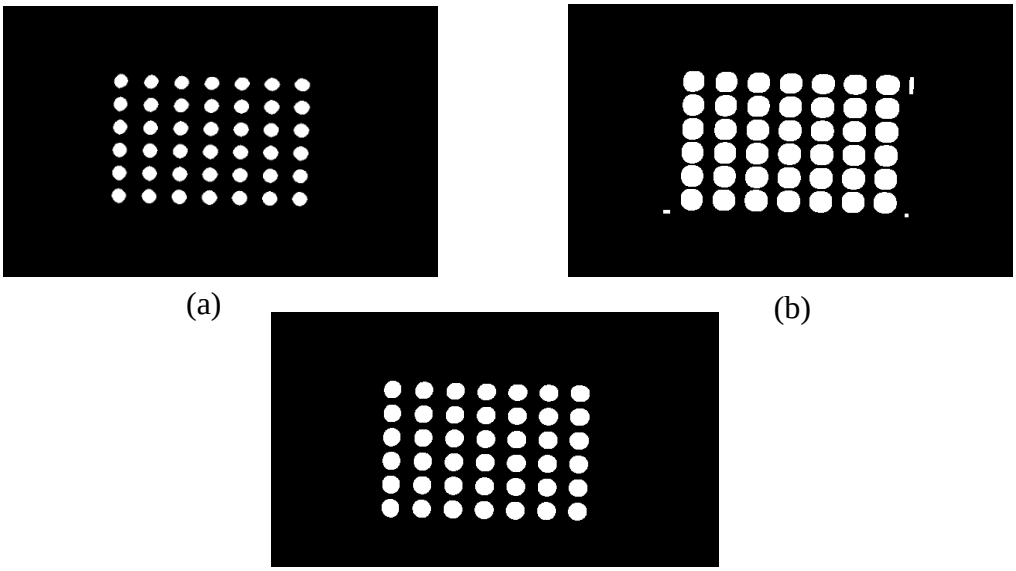
Figura 48 – Imagem original (a). Imagem após conversão para tons de cinza, aplicação do filtro gaussiano e threshold inverso (b).

Na Figura 48, os pontos em branco representam as posições onde estarão as peças do jogo. Como se pode verificar, ainda existem alguns pontos isolados. Para garantir que apenas são detetadas as posições do tabuleiro, foi imposta uma restrição onde apenas podiam ser detetados 43 objetos (42 círculos correspondentes aos espaços para as peças, 1 para o fundo preto). Assim sendo aplicou-se uma *abertura*, que consiste numa *erosão* seguida de uma *dilatação*. Estas operações são consideradas transformações morfológicas e são aplicadas em imagens binárias.

A erosão (Figura 49 (a)) consiste em reduzir a espessura do objeto, ou seja, erode os seus limites. Este processo funciona de modo semelhante a uma convolução. Existe um filtro que percorre a imagem binária, e se todos os pixéis dentro da vizinhança do filtro tiverem intensidade 1, na nova imagem o valor desse mesmo pixel será também 1, caso contrário será 0. Isto também significa que consegue remover pontos isolados na forma de ruído.

Já a dilatação é o oposto da erosão. Se dentro da vizinhança existir pelo menos um pixel com intensidade 1, na nova imagem esse mesmo pixel também terá valor 1, caso contrário será 0. Assim, esta operação expande os limites do objeto.

Desta forma, aplica-se a abertura que é a combinação destes dois processos (Figura 49 (c)). A erosão remove ruído, mas como consequência também erode os limites do objeto, por isso, usa-se a dilatação para voltar a expandir à sua forma original. É de notar, que não se deve aplicar uma dilatação antes de eliminar o ruído, caso contrário esta operação apenas aumenta o ruído, como se pode observar na Figura 49 (b).



(a) (b)

(c)

Figura 49 – Aplicação das operações morfológicas: erosão (a), dilatação (b) e abertura (c), à Figura 48 (b).

Após eliminar ruído, pode-se determinar o número total de objetos. Neste caso, um objeto será um conjunto de pixéis numa imagem que estão todos conectados entre si. Como referido anteriormente, apenas quando forem detetados 43 objetos é que se pode calcular os centroides de cada um. Esta operação é realizada através da função "*ConnectedComponentsWithStats*", integrada no *OpenCV*, que devolve uma lista com as coordenadas dos centroides de cada objeto. Em seguida, as coordenadas serão colocadas na forma matricial, de modo que o primeiro elemento corresponda à posição no canto superior esquerdo e o último elemento corresponda à posição no canto inferior direito (Figura 50). Para garantir isto as coordenadas terão de ser ordenadas primeiramente em *x* e de seguida em *y*, com grupos de 7 (correspondentes ao número de colunas).

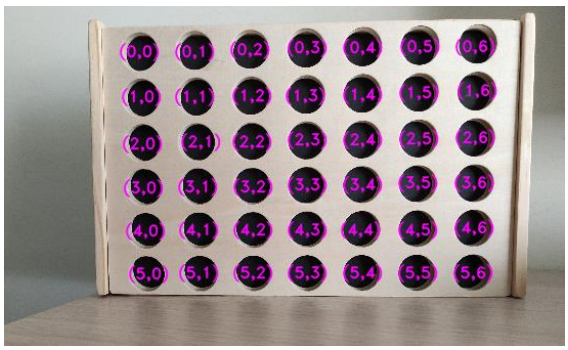


Figura 50 – Centro de cada posição na forma matricial.

Deteção das Peças

Uma vez identificadas as coordenadas de cada possível posição, pode-se avaliar a intensidade dos pixéis nesses pontos para identificar a sua cor. Através da identificação de cada cor em cada posição, reconhece-se como está o jogo.

Antes demais, para representar o estado do jogo ir-se-á usar uma matriz de tamanho *6x7*. As posições vazias correspondem ao valor 0, as posições com peça vermelha ao valor 1 e as azuis ao valor 2.

O algoritmo vai procurar na imagem tons de vermelho e tons de azul, que correspondem às cores da peça. Uma vez que a imagem original fornecida pela câmara é *RGB*, pode-se definir um intervalo para cada uma das matrizes que englobe os vários tons de vermelho e azul.

No entanto, este processo torna-se mais fácil quando se usa o sistema de cores *HSV* (*hue* - matiz, *saturarion* - satauração, *value* - valor). A matiz é a componente que descreve a cor em si, medida em graus. A saturação corresponde à quantidade de branco presente na cor. Por fim, o valor corresponde ao brilho da caro, quando o valor é igual a 0, a cor é preta.

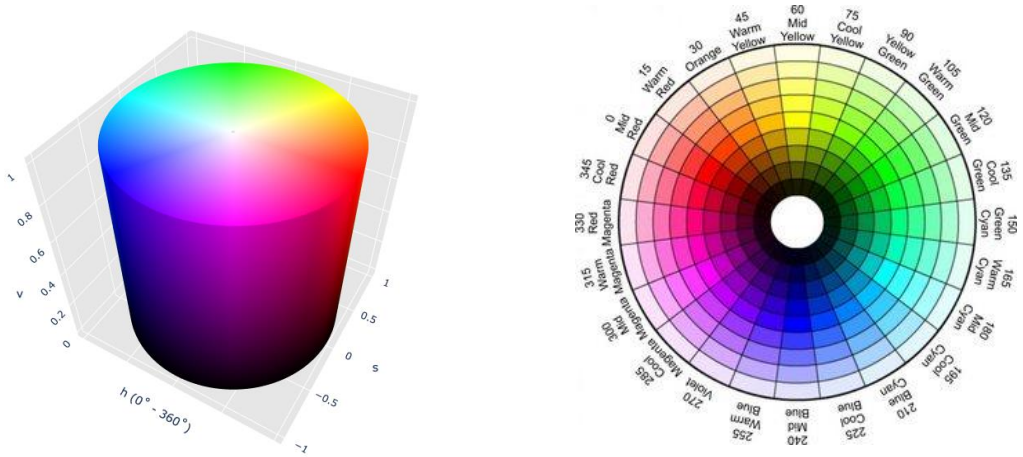
O método de conversão de uma imagem *RGB* para *HSV* é dado por:

$$V = Max\,(R,G,B)$$

$$S = \begin{cases} \dfrac{V - Min\,(R,G,B)}{V}, & se\ V > 0 \\ 0, & se\ V = 0 \end{cases}$$

$$H = \begin{cases} 60 \times \dfrac{G-B}{V - Min(R,G,B)} + 0, & se\ V = R\ e\ G \geq B \\ 60 \times \dfrac{G-B}{V - Min(R,G,B)} + 360, & se\ V = R\ e\ G < B \\ 60 \times \dfrac{B-R}{V - Min(R,G,B)} + 120, & se\ V = G \\ 60 \times \dfrac{R-G}{V - Min(R,G,B)} + 240, & se\ V = B \end{cases} \qquad (3.43)$$

Deste modo, é agora necessário definir os intervalos para as cores de cada peça. Numa representação *HSV* (Figura 51), a matiz pode variar entre 0 e 360 graus [43]. Neste sistema de cores, os vários tons de vermelho encontram-se entre os 345º (ou -15º) e os 15º, enquanto os tons de azul situam-se entre 180º e os 255º [44].



(a) (b)

Figura 51 – Representação HSV (a) [43]. Intervalo de ângulos da matiz entre 0º e 360º (b) [44].

Contudo, na biblioteca do *OpenCV*, a componente matiz varia entre 0 e 180 graus [45]. Quanto à saturação e ao valor estes variam entre 0 e 255. Portanto é necessário adaptar os valores da matiz (Figura 52).

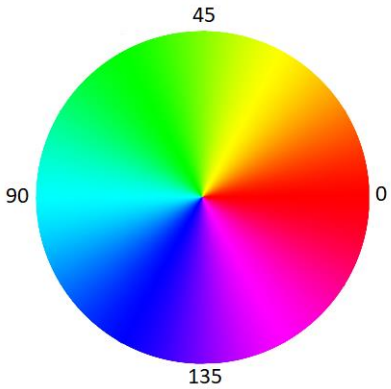


Figura 52 - Intervalo de ângulos da matiz entre 0º e 180º [45].

Posto isto, foram estabelecidos os seguintes intervalos para cada uma das cores:

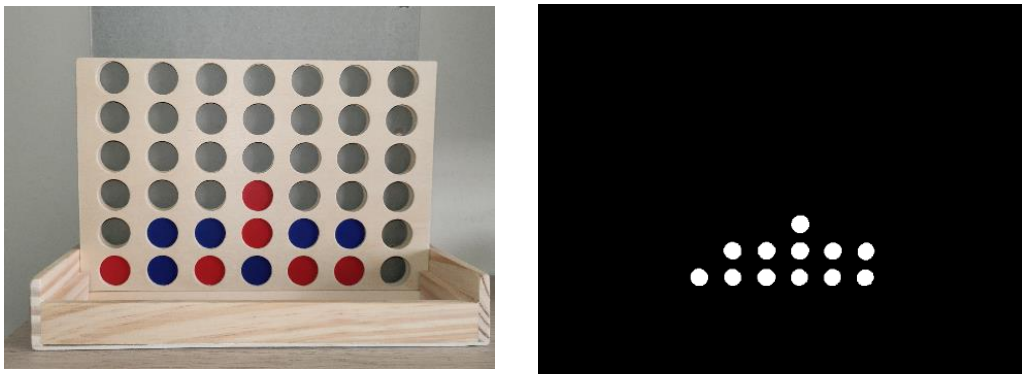
Tabela 8 – Intervalo dos parâmetros HSV para as peças Vermelha e Azul.

Peça	H (Matiz)	S (Saturação)	V (Valor)
Vermelha	[0º, 10º] e [160º, 180º]	[100, 255]	[100, 255]
Azul	[90º,130º]	[50, 255]	[50, 255]

Tal como no processo de deteção do tabuleiro, foram aplicados alguns processos de melhoramento de imagens para remover ruído, nomeadamente operações de erosão e dilatação.

Uma vez parametrizados os intervalos das cores para cada peça, é então possível ir aos centros obtidos anteriormente e verificar qual a cor correspondente a essa posição. Se os valores de intensidade registados se encontrarem dentro do intervalo estabelecido, será registado no elemento da matriz o número atribuído para a cor da peça.

Na Figura 53 é possível observar a imagem do tabuleiro, assim como a imagem onde se deteta as peças e por fim a matriz que corresponde ao estado atual do tabuleiro e que será fundamental para o algoritmo *Minimax*.



(a) (b)

```
Tabuleiro:
 [[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0.]
 [0. 2. 2. 1. 2. 2. 0.]
 [1. 2. 1. 2. 1. 1. 0.]]
```

(c)

Figura 53 – Imagem orignal (a). Imagem após filtrar as cores e aplicar operação de tratamento de imagem (b). Representação do tabuleiro na forma matricial (c).

3.4 Algoritmo Minimax

O algoritmo *Minimax* é um dos principais algoritmos utilizados em jogos de estratégia de dois jogadores com informação perfeita, como *xadrez*, *damas* e *go*. Foi proposto pela primeira vez na década de 1920 por John von Neumann, um renomeado matemático e economista.

É considerado como uma técnica de inteligência artificial (IA) utilizada para desenvolver sistemas computacionais capazes de realizar tarefas que normalmente exigem inteligência humana, como tomar decisões e resolver problemas complexos. Por esta razão, é uma abordagem clássica para a construção de um jogador artificial em jogos de estratégia, para determinar a melhor jogada, assumindo que o oponente também executa a jogada perfeita.

O nome "*Minimax*" deriva das duas principais etapas do algoritmo: um dos jogadores tenta maximizar os seus ganhos (*max*), enquanto o adversário tenta minimizar (*mini*) esses ganhos. Neste caso, o robô é o jogador que maximiza e o humano o que minimiza.

Este algoritmo é recursivo, ou seja, faz chamadas a si próprio. Desta forma, examina todas as possíveis jogadas a partir do estado atual do jogo, e gera uma árvore de jogadas possíveis. Cada nó da árvore representa um estado do jogo, e as arestas são as jogadas que levam a outros estados. A árvore é percorrida, alternando entre os jogadores.

Após alcançar um nó terminal da árvore, ou seja, um estado do jogo onde não existem mais jogadas possíveis, é atribuído um valor de utilidade ao estado terminal. Esse valor pode ser determinado através de uma função heurística que avalia o estado do jogo no nó terminal.

De seguida, os valores são propagados de volta pelos nós da árvore até ao nó inicial. Se o jogador que maximiza estiver a jogar, vai escolher a jogada com o valor maior. Caso contrário, significa que é a vez do jogador que minimiza, evidentemente, este escolhe o menor valor.

O algoritmo continua a expandir e a explorar os nós da árvore, escolhendo as jogadas com base nas avaliações mínimas e máximas, até alcançar a raiz e obter a melhor jogada possível [46], [47].

Quatro em Linha

Para o "Quatro em Linha" são necessários 2 jogadores. Ambos têm no total 21 peças. O jogo é jogado num tabuleiro, com 6 linhas (horizontal) e 7 colunas (vertical). Se uma peça for jogada numa coluna, irá cair até atingir a posição mais baixa que estiver desocupada. Cada um dos jogadores tenta alinhar quatro das suas peças na horizontal, vertical ou diagonal. O primeiro a obter quatro das suas peças em linha ganha. No entanto, se os jogadores usarem todas as suas peças e não houver nenhum quatro em linha, o jogo resulta num empate.

Geralmente, em jogos tradicionais o primeiro a jogar é o que tem as peças brancas, mas como neste jogo apenas existem peças azuis (utilizadas pelo robô) e peças vermelhas (utilizadas pelo humano), o primeiro a começar é escolhido aleatoriamente.

Função de Avaliação

O uso de funções de avaliação é uma maneira heurística de resolver o problema, ou seja, é uma forma de estimar a melhor solução [46]. Em jogos de tabuleiro, atribui-se um valor heurístico a cada estado terminal a partir da função de avaliação. Estas funções, vão gerar uma avaliação com base no estado do jogo.

Assim sendo, considera-se o estado de jogo apresentado na Figura 54:

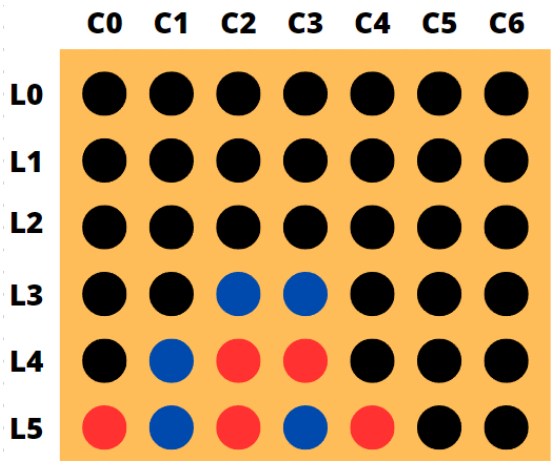


Figura 54 – Representação de um estado de jogo.

Da perspetiva do jogador com as peças azuis, efetua-se uma avaliação do estado atual do jogo. O valor avaliado pode ser obtido através da observação do número de peças azuis que estão em linha. Quanto maior o número de peças consecutivas em linha, maior o valor atribuído. Sendo assim, percorre-se cada coluna e verifica-se numa janela de quatro posições a quantidade de peças azuis que podem gerar um grupo de quatro em linha. Esta janela percorre o tabuleiro na horizontal, vertical e nas duas diagonais.

A função de avaliação será a seguinte:

- Se num espaço de quatro posições existirem 2 peças azuis e 2 espaços vazios: +2 pontos;
- Se num espaço de quatro posições existirem 3 peças azuis e 1 espaço vazio: +5 pontos;
- Se num espaço de quatro posições existirem 3 peças vermelhas e 1 espaço vazio: -10 pontos;
- Por cada peça colocada na coluna do meio (C3): +1 ponto;
- Se existirem 4 peças azuis em linha (azul vence): +1000 pontos;
- Se existirem 4 peças vermelhas em linha (vermelho vence): -1000 pontos.

Ao percorrer cada uma das colunas e aplicando a função descrita anteriormente, obtém-se a seguinte avaliação para cada coluna:

- C0 tem +2 pontos na horizontal em L3[C0:C3];
- C1 tem +2 pontos na vertical [L2:L5], +2 pontos na horizontal em L3[C1:C4] e +2 pontos na diagonal [L4,C1:L1,C4];
- C2 tem +2 pontos na horizontal [C2:C5];
- As restantes colunas têm 0 pontos.

Isto significa que este estado de jogo tem uma avaliação total de +12 pontos.

Árvore de Decisão

A partir do estado inicial do jogo gera-se um conjunto de ramificações com todas as jogadas possíveis, originando assim uma árvore. A tamanho da árvore irá depender do nível de profundidade. Um nível de profundidade explora todas as opções para a jogada

seguinte, sem ter em consideração a jogada posterior do adversário. Enquanto uma árvore com dois níveis de profundidade já tem em consideração a jogada do adversário. À medida que se vai adicionando mais níveis de profundidade, mais cenários futuros se consegue observar, logo maior o nível de dificuldade da IA.

Uma vez que o algoritmo inicializa sempre que se vai iniciar uma jogada do robô, o primeiro nível de profundidade tenta sempre maximizar o valor da avaliação. No nível seguinte de profundidade, passa a ser a vez do adversário que tenta minimizar o resultado. Esta situação vai-se alternando até chegar à profundidade final.

Supondo, que a partir do estado da figura anterior se pretende efetuar o *Minimax* para dois níveis de profundidade. Visto que é a vez do azul efetuar a jogada, no primeiro nível são verificadas todas as possíveis posições, onde cada uma dessa posições gera um estado de jogo que equivale a um nó. Para cada um desses nós, observam-se todas as possíveis posições, o que corresponde ao segundo nível de profundidade. Uma vez que se atingiu o nível de profundidade máximo, aplica-se a função de avaliação atrás descrita a cada um dos estados terminais (Figura 55).

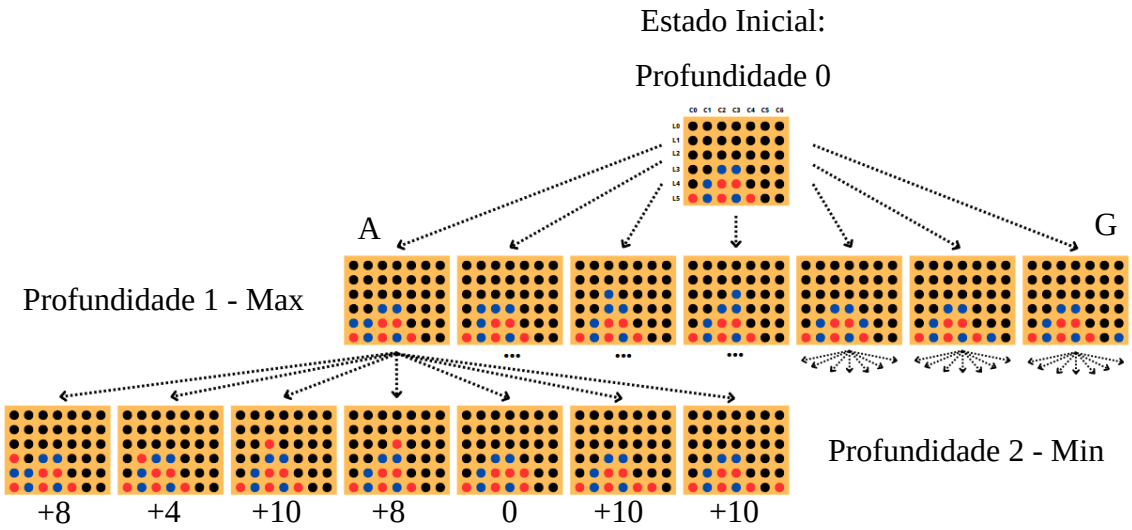


Figura 55 – Geração de uma árvore Minimax, com profundidade 2.

Como o último nível de profundidade corresponde a uma jogada do adversário, o valor a propagar para o nó anterior será o valor mínimo. Logo, o nó *A* recebe o valor mais baixo de um dos seus descendentes, que neste caso corresponde a 0. O mesmo processo repete-se para os restantes nós. Quando todos os nós do primeiro nível tiverem recebido o valor

mínimo dos seus descendentes, retira-se o valor máximo, que corresponde à melhor jogada que o azul pode efetuar, dado a jogada seguinte do seu adversário.

Alfa-Beta Pruning

Apesar do algoritmo funcionar bastante bem, a adição de níveis de profundidade aumenta exponencialmente a quantidade de cenários a observar. Isto significa que o algoritmo fica bastante mais pesado em termos computacionais, ou seja, torna-o mais lento.

Portanto, desenvolveu-se um mecanismo que permite reduzir significativamente o número de ramificações que são criadas, isto significa que se vai desprezar as jogadas que não são promissoras [47]. Este mecanismo introduz dois novos parâmetros ao algoritmo para realizar o *pruning* (desbaste).

O parâmetro *Alfa* é o valor máximo que se pode obter no nível atual ou superior, enquanto *Beta* é o valor mínimo que pode ser obtido no nível atual ou superior [48].

A Figura 56 seguinte demonstra o funcionamento desta nova adição. Nesta situação o jogo já está quase a terminar, com apenas duas colunas com espaço para jogar. Isto significa que a quantidade de ramificações já é reduzida. Adicionalmente, verifica-se que o algoritmo tem quatro níveis de profundidade.

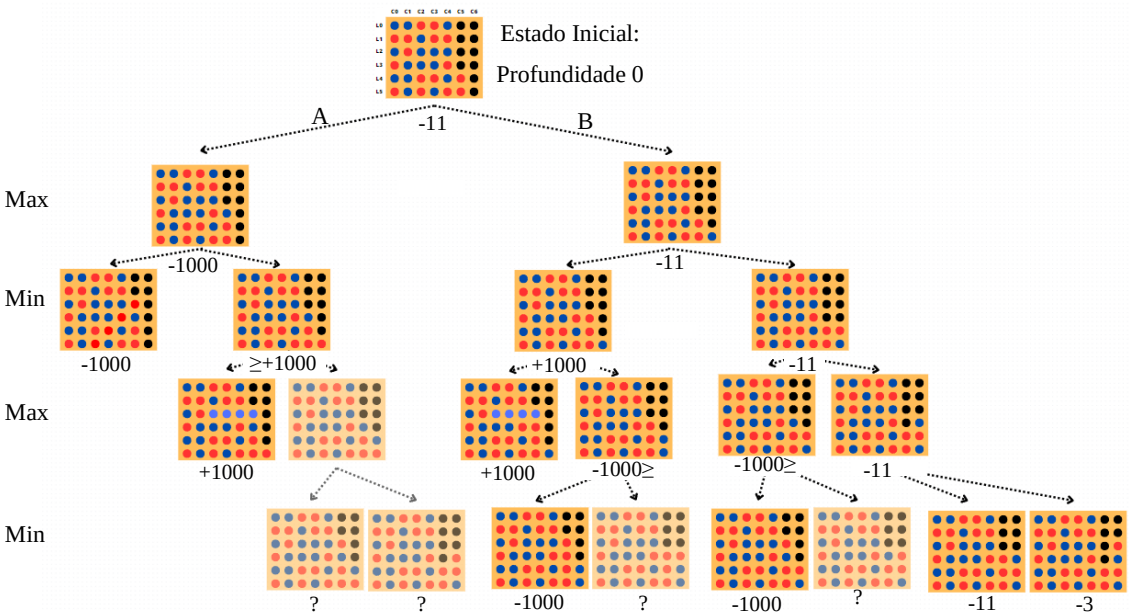


Figura 56 – Árvore Minimax com profundidade 4 e Alfa-Beta pruning.

Note-se que houve duas situações onde o jogo terminou antecipadamente por vitória de um dos jogadores. Uma no lado *A* da árvore com vitória do jogador vermelho a profundidade dois, e uma vitória por parte do azul no lado *B* com profundidade igual a três. Nestes casos a avaliação é feita nesses instantes e o nó não cria mais descendentes. Posto isto, os restantes atingem os estados terminais e efetua-se a avaliação do tabuleiro.

Começando no nível inferior (lado A) da esquerda para a direita, à profundidade três, pretende-se maximizar o valor, logo o valor a propagar será +1000 ou maior que isso. No entanto à profundidade dois já está avaliado um valor -1000, referente à vitória antecipada do jogador com a peça vermelha. Isto significa que independentemente do valor das restantes duas avaliações o valor a ser escolhido será sempre -1000. Desta forma, esses nós não são considerados e passa-se logo para o lado *B* da árvore.

Tal como anteriormente, o estado final mais à esquerda em *B* termina com uma vitória da peça vermelha. Antes de se avaliar o estado seguinte, o valor é temporariamente propagado para o seu antecessor. A partir daí, pretende-se retirar o valor máximo entre +1000 e um valor que será -1000 ou menor ainda. Evidentemente, não é necessário avaliar o estado seguinte, pois qualquer valor que tome nunca vai ser escolhido, portanto não se considera esse ramo. No ramo seguinte acontece o mesmo.

O resto da árvore terá de ser avaliada e conclui-se que a melhor opção é o cenário que leva à avaliação -11. Deste modo, evitou-se avaliar 4 situações onde as jogadas não eram promissoras. A adição destes parâmetros ao algoritmo *Minimax* permite usar mais níveis de profundidade sem sacrificar o tempo de processamento.

Como este sistema tem previamente processamento de imagem, é importante que se evite tempos de processamento computacional desnecessários. Este algoritmo está a ser aplicado ao Quatro em Linha e mesmo assim já se viu o que se pode ganhar em adicionar o *Alfa-Beta pruning*, algo que seria ainda mais vantajoso num jogo mais complexo como por exemplo o xadrez.

4 Apresentação de Resultados

Neste Capítulo são apresentados os resultados obtidos no âmbito desta de dissertação de mestrado. Nesta etapa, serão discutidas as principais características associadas à determinação dos referenciais para apanhar e largar as peças do jogo, que envolveu a análise dos movimentos e das posições ideais para a manipulação das peças durante o jogo. Além disso, será abordada a aplicação desenvolvida, na qual foi possível efetuar, com o auxílio do processamento de imagem, a escolha da jogada a ser realizada pelo robô. Serão apresentados os detalhes desta aplicação, destacando os algoritmos utilizados e os resultados obtidos durante os testes.

A solução final é composta de um manipulador, uma câmara externa, o jogo quatro em linha e a calha onde serão apanhadas as peças, tal como representado na Figura 57.

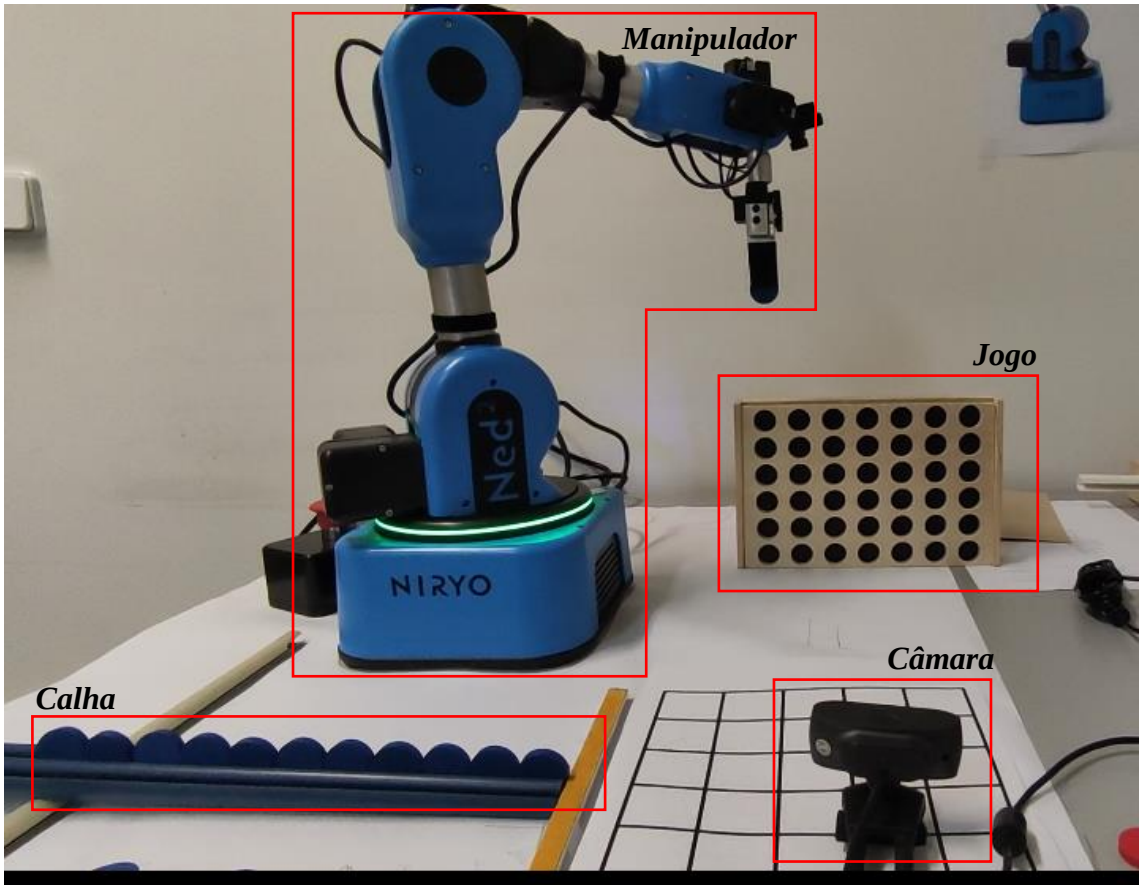


Figura 57 – Composição da solução final.

4.1 Posição e Orientação dos Referenciais

No Capítulo 3.2 foi estabelecida uma sequência que representa o movimento que o manipulador irá executar para realizar uma jogada. As posições *pick_pose_high* e *pick_pose*, dependem da distância entre o manipulador e o local onde a peça irá ser apanhada. As posições *drop_pose_high* e *drop_pose*, dependem da distância entre o manipulador e o tabuleiro. Antes de iniciar o jogo, estas posições têm de ser devidamente registadas, caso contrário o sistema irá apresentar resultados indesejados.

De forma a representar estas posições, que na verdade correspondem a uma matriz de transformação que relaciona a base do manipulador com o local onde se apanha/larga a peça, irão ser especificadas as coordenadas *XYZ* em metros a orientação na forma *RPY* em radianos.

Posição = (X;Y;Z;Roll;Pitch;Yaw)

Apanhar a Peça

Como referido anteriormente, o manipulador irá jogar com as peças azuis. Estas, irão estar colocadas verticalmente sobre uma calha inclinada (Figura 58). É importante que a calha seja inclinada, pois assim que uma das peças seja apanhada na posição, *pick_pose*, a seguinte desloca-se para essa posição por ação da gravidade. Desta forma o manipulador apanha as peças sempre no mesmo local.

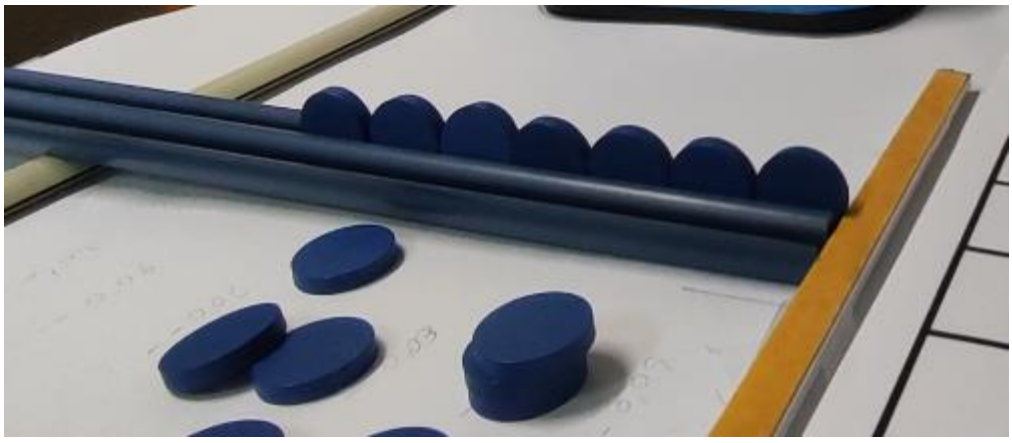


Figura 58 – Calha para as peças.

Relativamente à orientação existem várias maneiras de como se pode apanhar um objeto. Após alguns testes, verificou-se que a melhor maneira de apanhar a peça era com uma aproximação vertical descendente, isto significa que o eixo *Z* da garra aponta para o chão durante esta ação (Figura 59).

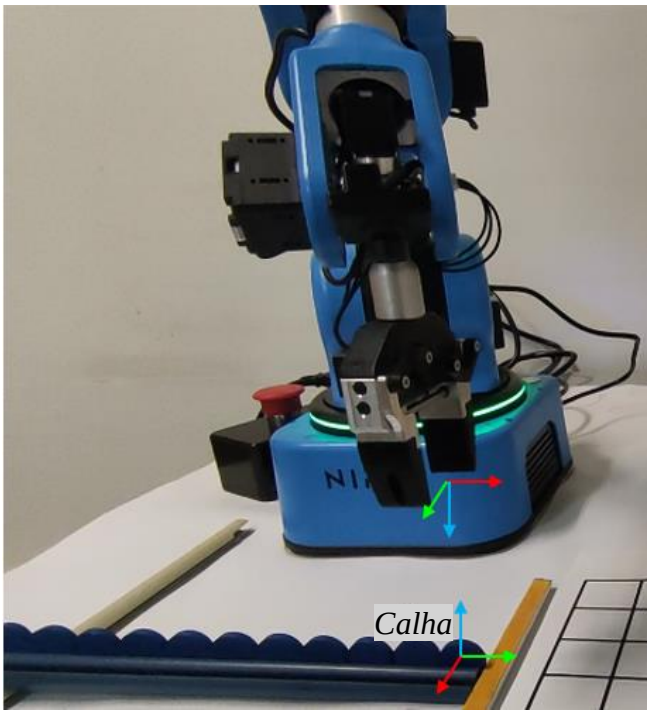


Figura 59 – Orientação do manipulador para apanhar a peça.

Largar a Peça

Contrariamente à situação anterior, onde o manipulador se desloca sempre ao mesmo local para apanhar a peça, aqui já existem sete posições possíveis para a largar. Assim sendo, será estabelecida uma posição de referência associada à primeira coluna a contar da esquerda, que representa a *coluna 0*. Como a distância entre colunas (*dist*) é fixa, é possível calcular a posição para cada uma delas (Figura 60). Contudo, é necessário ter em atenção a posição e orientação do tabuleiro em relação à base do manipulador (Figura 61).

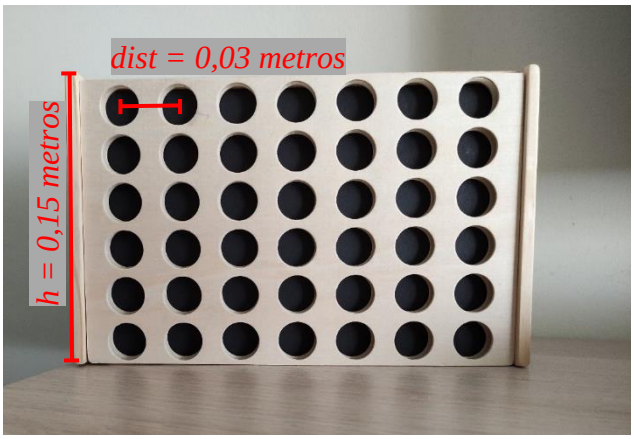


Figura 60 – Especificações do tabuleiro.

A equação que determina as novas coordenada *X'* e *Y'* para todas as colunas, é dada por:

$$\begin{cases} Y' = Y_0 + (nº\ coluna\ \times\ dist) \cdot cos\alpha \\ X' = X_0 - (nº\ coluna\ \times\ dist) \cdot sin\alpha \end{cases} \qquad (4.1)$$

Onde, (X_0,Y_0) são as coordenadas que representam a distância desde a base do manipulador até à posição de largar a peça na *coluna 0*. Enquanto o ângulo α, representa a orientação do referencial do tabuleiro em relação à base do manipulador.

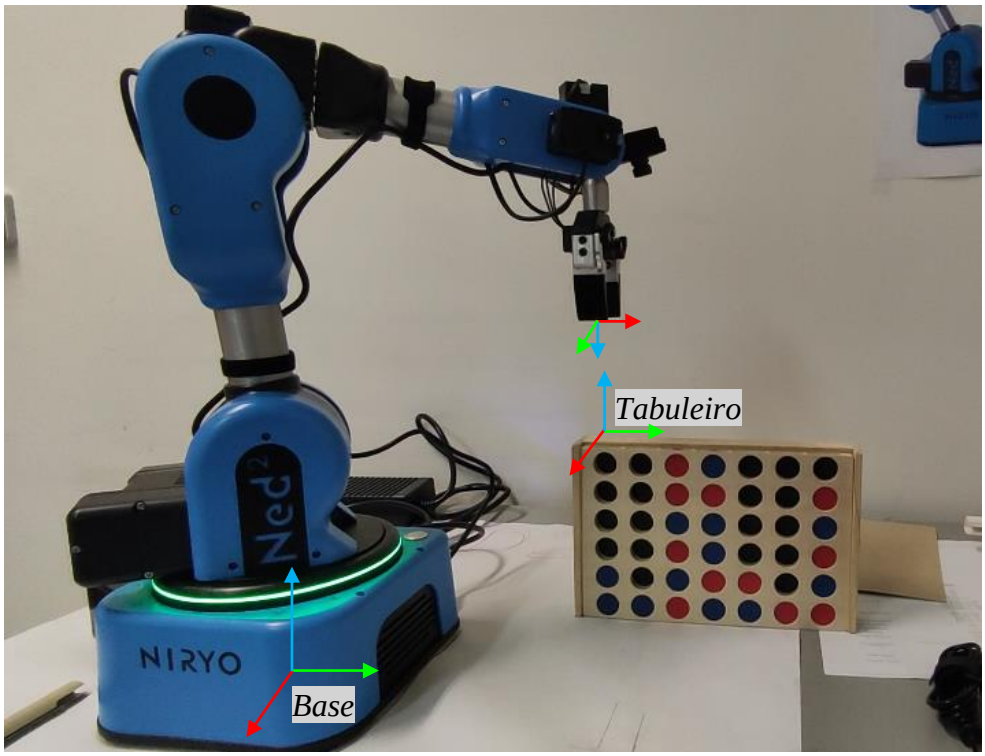


Figura 61 – Orientação para largar a peça.

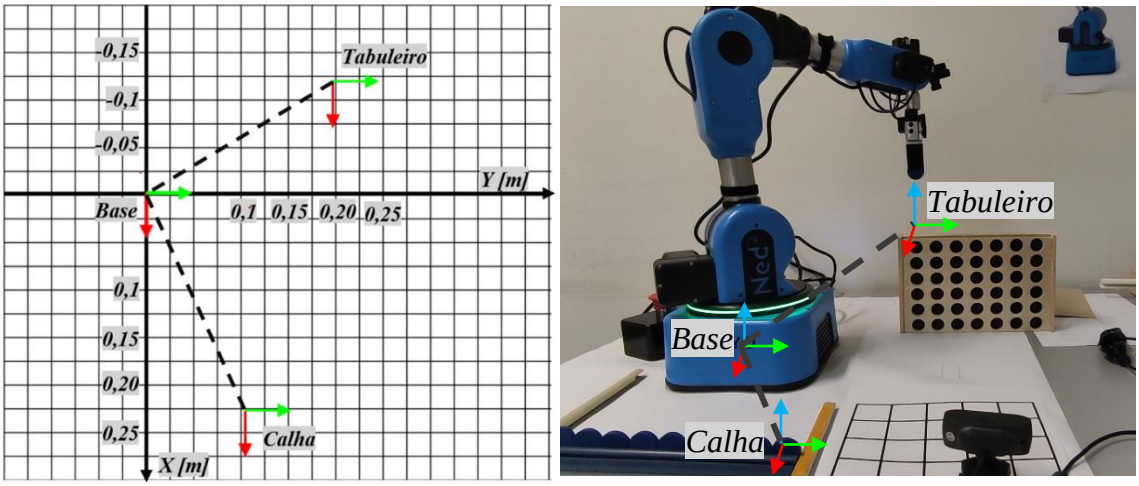
Posto isto, para a demonstração realizada foram consideradas as seguintes posições:

Tabela 9 – Posições e Orientações do Efetor Terminal

Posição	*X [m]*	*Y [m]*	*Z [m]*	*Roll [rad]*	*Pitch [rad]*	*Yaw [rad]*
pick_pose_high	0,230	0,104	0,020	1,57	0,00	-3,14
pick_pose	0,230	0,104	0,250	1,57	0,00	-3,14
drop_pose_high	-0,120	0,196	0,180	1,57	0,00	-3,14
drop_pose (C0)	-0,120	0,196	0,155	1,57	0,00	-3,14

Os valores da última linha da tabela são referentes à *coluna 0*, ou seja, $X_0 = -0{,}120\ m$ e $Y_0 = 0{,}196\ m$. Quanto à orientação do tabuleiro, considerou-se que os seu eixos *X* e *Y* seriam paralelos aos eixos da base do manipulador, logo $\alpha = 0^o$.

Na Figura 62 encontram-se representadas as posições e orientações dos referenciais relativamente à base do manipulador com base nos valores medidos (Tabela 9). O referencial da calha corresponde à *pick_pose*, enquanto a *drop_pose (C0)* corresponde ao referencial do tabuleiro sobre a *coluna 0*. As posições *pick_pose_high* e *drop_pose_high* servem para o manipulador subir a garra (ao longo do eixo *Z*) e evitar colisões com objetos que possam estar na mesa quando este se movimenta.



(a) (b)

Figura 62 – Referenciais Oxy com as posições e orientações dos objetos (a). Referenciais na situação real (b).

Caso se pretenda colocar uma peça na coluna do meio (*coluna 3*) é necessário calcular a nova posição. Assim, utiliza-se a equação *(4.1)* que resulta em:

$$\begin{cases} Y' = 0{,}196 + (3 \times 0{,}03) \cdot \cos(0) = 0{,}289\ m \\ X' = -0{,}120 - (3\ \times\ 0{,}03) \cdot \sin(0) = -0{,}120\ m \end{cases} \qquad (4.2)$$

Dado que neste exemplo o ângulo *α = 0º*, quando se calcula a posição para cada coluna, apenas se altera o eixo *Y*. Assim, a nova posição *drop_pose (C3) = (-0,120;0,289;0,155; 1.57;0,00;-3,14)*. As restantes posições permanecem iguais. Na forma matricial estes referenciais, que representam o local onde se apanha/larga a peça do ponto de vista da base do manipulador, escrevem-se na seguinte forma:

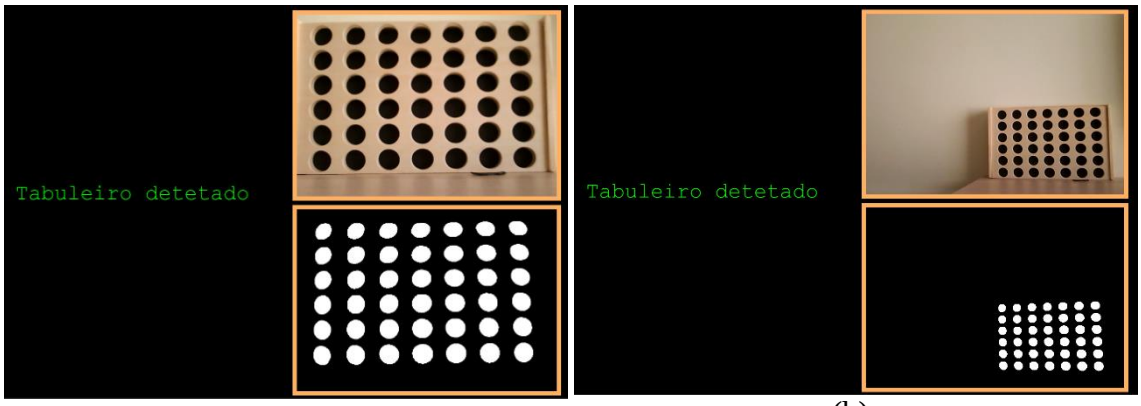
$${}^{Base}A_{Tabu(C0)} = \begin{bmatrix} 0 & 1 & 0 & -0{,}120 \\ 1 & 0 & 0 & 0{,}196 \\ 0 & 0 & -1 & 0{,}155 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{Base}A_{Calha} = \begin{bmatrix} 0 & 1 & 0 & -0{,}230 \\ 1 & 0 & 0 & 0{,}104 \\ 0 & 0 & -1 & 0{,}020 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{Base}A_{Tabu(C3)} = \begin{bmatrix} 0 & 1 & 0 & -0{,}120 \\ 1 & 0 & 0 & 0{,}289 \\ 0 & 0 & -1 & 0{,}155 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.2 Aplicação Prática

Uma vez estabelecidos os referenciais pode-se iniciar o processo. Ao executar o programa o utilizador depara-se com uma janela onde aparecem duas capturas de vídeo. Uma corresponde à informação original transmitida pela câmara e a outra corresponde à deteção do tabuleiro após aplicado o processamento de imagem.

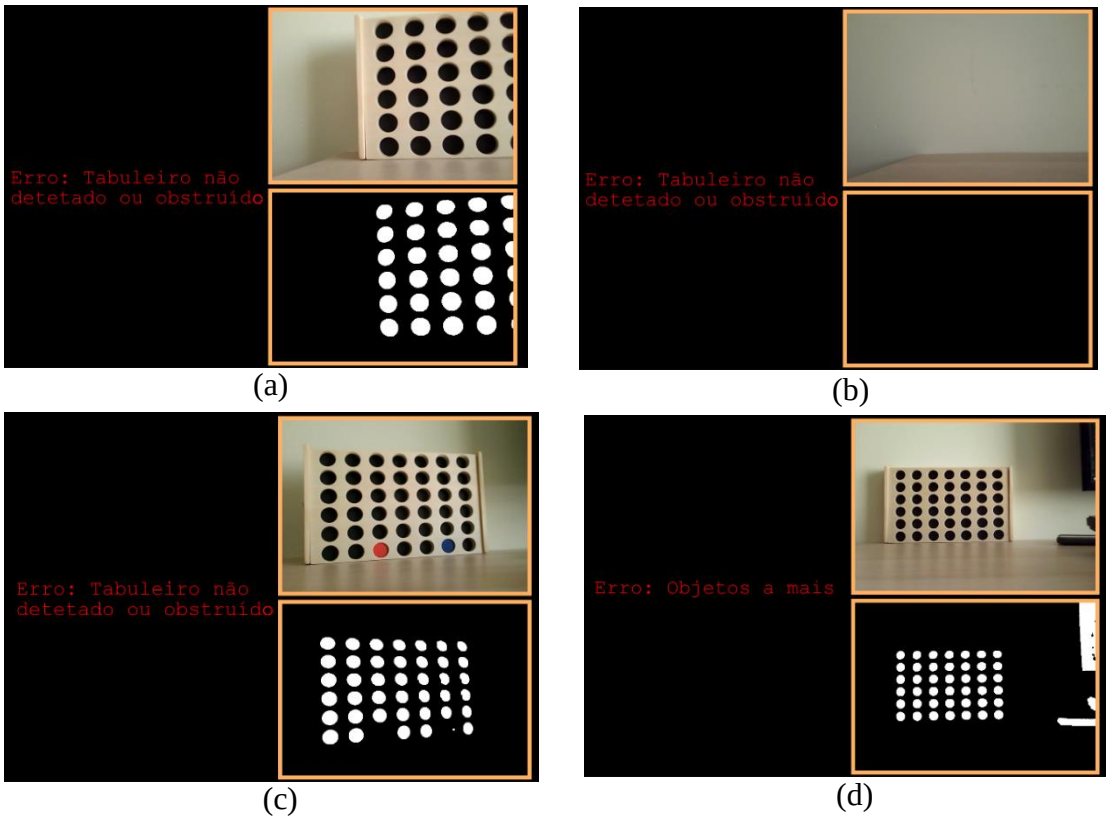
Nesta fase, o utilizador posiciona a câmara de modo a apanhar o tabuleiro na sua totalidade. A distância focal da câmara não interfere com o procedimento, não importa se o tabuleiro está mais à direita/esquerda ou se está mais perto/longe na imagem (Figura 63). Quando a câmara estiver devidamente posicionada, o utilizador pressiona na tecla "*S*" que simboliza o *START* e o programa avança para a fase do jogo.



(a) (b)

Figura 63 – Deteção correta do tabuleiro. Distância próxima (a). Tabuleiro afastado e não centrado (b).

Contudo, se os requisitos não forem cumpridos o programa não avança para a fase seguinte. Isto acontece quando a câmara não deteta o tabuleiro na sua totalidade, ou quando existem objetos que possam ser interpretados como parte do tabuleiro pelo processamento de imagem. Nestas situações, aparece uma mensagem de erro e o programa fica retido nesta secção até que o problema seja resolvido (Figura 64).



(a) (b)

(c) (d)

Figura 64 – Vários tipos de erro na fase inicial. Tabuleiro não detetado totalmente (a). Tabuleiro não se encontra na imagem (b). Peças inicialmente no tabuleiro (c). Objetos a mais (d).

Ao avançar para a segunda fase do programa, a janela é atualizada e desta vez, aparecem três secções (Figura 65). Duas destas secções representam a imagem original captada pela câmara (*A*) e a imagem com apenas as peças que se encontram colocadas no tabuleiro (*B*). Por fim, a terceira secção corresponde à interpretação do computador, ou seja, é uma representação gráfica relativamente ao estado atual do jogo (*C*). Adicionalmente, verifica-se que na secção *A* existe a presença de uns indicadores verdes que representam as posições onde pode ser colocada a próxima peça.

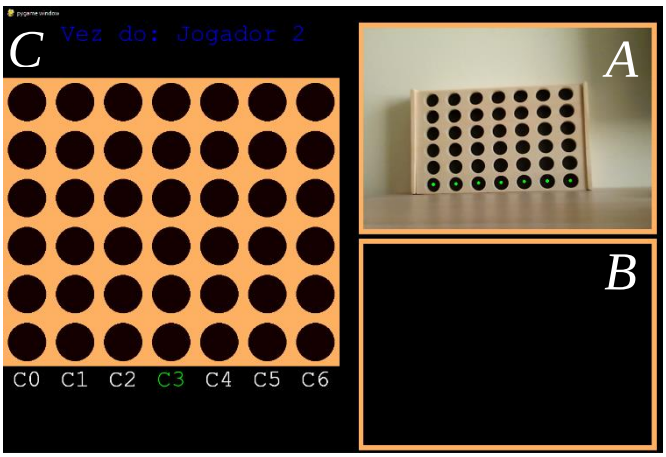


Figura 65 – Segunda fase do programa, onde se realiza o jogo.

Posto isto, define-se aleatoriamente qual será o primeiro jogador e este realiza uma jogada. Assim que existir uma alteração na intensidade dos pixéis numa das posições válidas do tabuleiro, o computador regista a peça e dá a vez ao jogador seguinte.

Quando é a vez do robô, este interpreta o estado de jogo atual, e a partir daí procura a melhor decisão com base no algoritmo *Minimax*, com seis níveis de profundidade. O algoritmo retorna a coluna com a melhor avaliação, e dá ordem ao manipulador para realizar a sequência que irá colocar a peça no tabuleiro. Enquanto o manipulador realiza o movimento, na secção *C*, encontra-se realçada a coluna onde o manipulador vai colocar a peça.

Na Figura 65 o robô é o primeiro a jogar. O algoritmo de decisão definiu a coluna 3 como a melhor jogada (realçada a verde), logo o manipulador executa a sequência e larga a peça na respetiva posição (Figura 66).

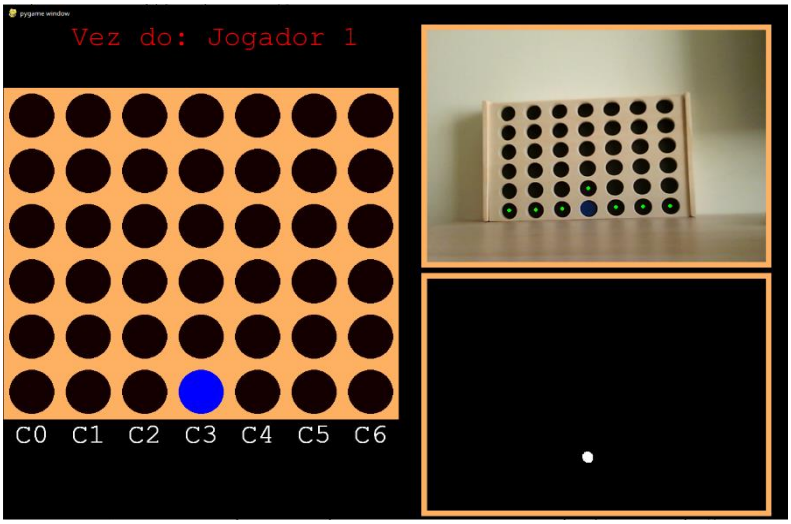


Figura 66 – Estado do jogo após a primeira jogada.

Assim que o robô larga a peça o utilizador pode efetuar a sua jogada. Verifica-se que o indicador da coluna 3 na secção *A* subiu uma linha, representando a nova posição disponível referente a essa coluna. Neste exemplo, o jogador humano escolheu jogar na coluna 3, como representado na Figura 67.

O processo repete-se até o jogo terminar, por vitória de um dos jogadores, ou por empate. Assim que o jogo terminar, o robô retorna à sua posição *start* e o programa termina.

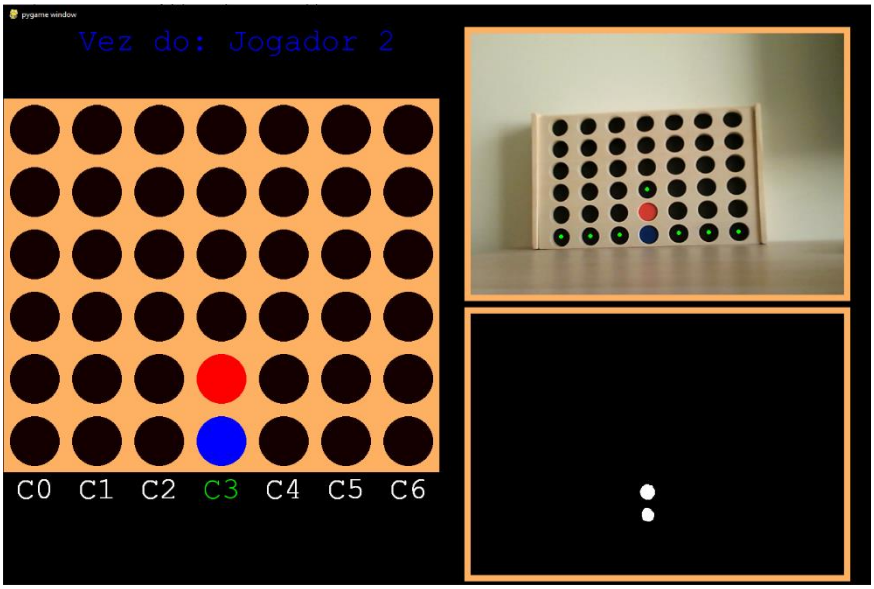


Figura 67 – Estado do jogo após a segunda jogada.

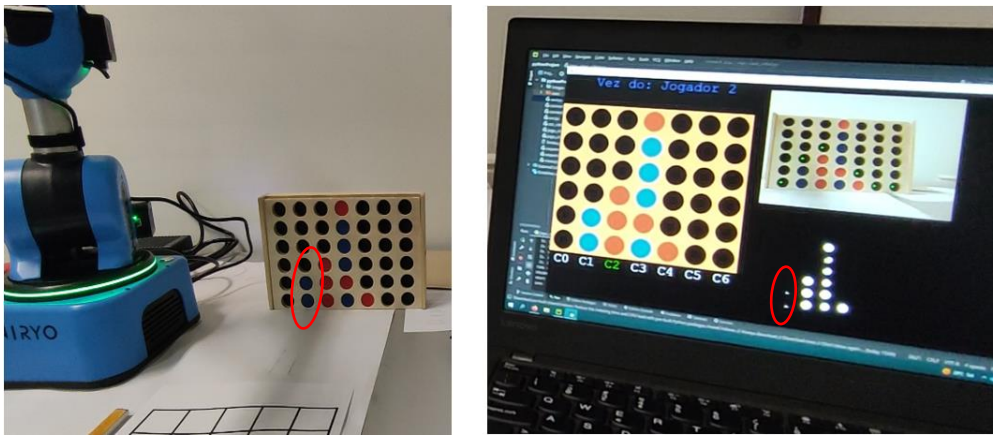
4.3 Desempenho do Sistema

Através da realização de vários ensaios, foi possível avaliar o comportamento do sistema. Em suma, os resultados foram muito positivos. Todo o sistema funcionava como desejado, salvo algumas limitações descritas adiante. Observou-se que os movimentos do manipulador eram corretos e suaves e os tempos de decisão do algoritmo eram curtos. Na maioria das vezes era o humano que demorava mais tempo a pensar onde deveria jogar.

Um grande desafio foi estabelecer como se iria apanhar a peça do jogo e como a colocar no tabuleiro. Desde o início a opção foi sempre trabalhar com o tabuleiro na vertical, tal como no jogo real, sem fazer simplificações. Como a peça é para ser colocada na vertical, a solução de apanhar a peça também é na vertical, através de uma calha com uma certa inclinação.

Quando corretamente estabelecidos os referenciais da calha e do tabuleiro, o manipulador conseguia realizar a tarefa com 100% de eficácia. Desta forma, conseguiu-se efetuar vários jogos seguidos, sem ter de efetuar ajustes ao sistema. No entanto, é importante referir que se um destes referenciais não estiver corretamente dimensionado, irão existir falhas no sistema. Isto é principalmente importante quando um jogo acaba e se pretende iniciar outro. Ao retirar as peças do tabuleiro é necessário garantir que não se altera a posição do mesmo, pois pode afetar o ensaio seguinte. Se, porventura, ocorrer uma falha ao largar a peça no tabuleiro, o jogo terá de ser reiniciado.

O sistema de imagem também funcionou dentro do esperado, mesmo com uma qualidade de iluminação deficiente. Por vezes, quando o manipulador se movimentava, bloqueava a fonte de luz e fazia sombra no tabuleiro (Figura 68), que podia ofuscar alguma informação, mas como se travam de peças que já estavam avaliadas não causou limitações na sua globalidade. Contudo, em situações de processamento de imagem é importante que haja uma boa qualidade de iluminação para evitar resultados indesejados. Todavia foram efetuados testes com diversos tipos de iluminação de modo a verificar a influencia no sistema.



(a) (b)

Figura 68 – Efeito da má iluminação no tratamento de imagem. Tabuleiro sombreado (a). Dificuldade na representação das peças azuis (b).

Quanto ao desempenho do algoritmo *Minimax* os resultados também foram bastante satisfatórios. Dos vários ensaios realizados (mais de 50) com profundidades de 5 e 6, apenas foram registados um empate e uma derrota por parte do algoritmo. No que toca à adição dos parâmetros *Alfa-Beta*, também é interessante avaliar o tempo entre cada decisão.

A tabela seguinte apresenta o tempo médio a que o algoritmo demorava a devolver a coluna a jogar. Uma vez que é no início do jogo onde existem mais possibilidades, ou seja, ramificações, é aqui que o algoritmo demora mais tempo a processar a melhor solução. Logo estas medições foram realizadas para a primeira jogada efetuada.

Tabela 10 – Comparação entre o tempo que demora a efetuar a primeira jogada com e sem Alfa-Beta Pruning.

	Tempo [s]	
Profundidade	Sem *Alfa-Beta Pruning*	Com *Alfa-Beta Pruning*
5	2,69	0,39
6	18,25	1,55
7	-	6,05

Ao adicionar mais um nível de profundidade o tempo de computação aumenta exponencialmente. Uma profundidade de 6 não era viável nesta aplicação sem a adição dos parâmetros *Alfa-Beta*, além disso, à profundidade 7 o programa encravava e já não conseguia realizar a tarefa sem o *pruning*.

5 Conclusões e Trabalho Futuro

Neste Capítulo final apresentam-se as principais conclusões deste trabalho realçando os pontos positivos e algumas limitações, das quais podem servir como linhas de investigação consideradas de interesse para desenvolvimento de trabalhos futuros.

5.1 Conclusão

Em conclusão, esta dissertação de mestrado alcançou com sucesso o objetivo de desenvolver e implementar um sistema que permite jogar o jogo "Quatro em Linha" entre um humano e um manipulador robótico. Para alcançar este feito, foram realizados estudos teóricos sobre manipuladores robóticos e técnicas de processamento de imagem. Além disso, um algoritmo foi desenvolvido para o reconhecimento das posições das peças no tabuleiro e para a tomada de decisão dos movimentos realizados pelo manipulador.

No âmbito dos estudos teóricos, uma comparação entre métodos analíticos e numéricos para determinar a cinemática inversa do manipulador foi realizada. Foi apresentada uma solução analítica para uma situação em que o punho fosse esférico. Para o punho não esférico, a obtenção da cinemática inversa foi efetuada através do modelo numérico BFGS, aproveitando a *toolbox* do MATLAB.

No que diz respeito ao processamento de imagem, várias técnicas foram implementadas para alcançar os objetivos propostos. O uso de um *threshold* foi efetivo para detetar a localização do tabuleiro e do centro de massa das posições, garantindo que apenas se avançava para a próxima fase quando as condições de validação eram cumpridas. A deteção das peças foi realizada através da filtragem das cores (vermelho para o jogador humano e azul para o robô) no espaço de cores HSV.

Para alcançar as melhores jogadas, foi utilizado o algoritmo *Minimax*. Embora tenha produzido bons resultados, observou-se que, em certas profundidades, o tempo para determinar a jogada aumentava significativamente. Para aprimorar os resultados, a técnica de *Alfa-Beta Pruning* foi adicionada para eliminar jogadas menos favoráveis.

Os resultados obtidos no geral foram positivos, demonstrando a viabilidade do sistema proposto. No entanto, algumas situações podem afetar o seu funcionamento. Um dos principais desafios é garantir que o tabuleiro esteja na posição correta, pois, caso contrário, as peças não serão largadas nos lugares adequados, impactando o desempenho do sistema. A calibração adequada da posição do tabuleiro é essencial para evitar esse problema. Além disso, a iluminação pode afetar o reconhecimento das peças azuis, especialmente em condições de pouca luminosidade.

No contexto geral, o trabalho realizado ao longo desta dissertação serve como uma base sólida para desenvolvimentos na robótica interativa e da visão artificial, ao permitir que um robô jogue o jogo "Quatro em Linha" com um ser humano. As limitações encontradas também oferecem oportunidades para futuras melhorias.

Para terminar, acrescento que este sistema foi utilizado com sucesso no dia aberto do ISEL (a 7 de julho de 2023), onde alguns alunos do secundário puderam interagir com o sistema e ficaram positivamente agradados com a prestação do mesmo.

5.2 Trabalho Futuro

Além dos objetivos já alcançados nesta dissertação de mestrado, existem várias oportunidades promissoras para trabalhos futuros que podem aperfeiçoar ainda mais o sistema de jogo "Quatro em Linha" entre um humano e um manipulador robótico. Algumas dessas possibilidades incluem:

- Posição do Tabuleiro: É possível desenvolver técnicas avançadas para calcular a distância entre o tabuleiro e a câmara, isto é crucial para permitir que o manipulador alcance todas as posições do tabuleiro, independentemente da sua posição no espaço de trabalho. Para isto, existem métodos como triangulação estereoscópica (onde são necessárias duas câmaras) ou uso de câmaras com informação relativamente à profundidade da imagem (RGB-D), que podem ser investigados para determinar a localização do tabuleiro em relação à câmara e ao manipulador;

- Deteção de Região de Interesse (ROI): A aplicação de técnicas de deteção de objetos que envolvem ferramentas de aprendizagem automática como redes neurais convolucionais, pode ser explorada para identificar a região de interesse onde apenas o tabuleiro está localizado. Ao limitar a área de busca, o sistema pode economizar recursos computacionais e melhorar a eficiência do processamento de imagem. Para isto, é necessário treinar o sistema com uma base de dados de imagens do tabuleiro em vários cenários e condições diferentes;
- Melhoramento do Algoritmo de Decisão e Adaptação para Jogos Mais Complexos: Investigar estratégias avançadas de *pruning* com o uso de funções heurísticas mais sofisticadas, isto pode ajudar a melhorar a eficiência do algoritmo e levar a um desempenho mais próximo da perfeição no jogo "Quatro em Linha". Adicionalmente, pode-se expandir a aplicação do sistema para outros jogos estratégicos mais complexos. Ao ajustar o algoritmo e as técnicas de processamento de imagem, o manipulador robótico poderia fazer novos desafios, como jogos de tabuleiro mais elaborados.

Em suma, as possibilidades de trabalho futuro são abundantes. Com a contínua pesquisa e desenvolvimento nestas áreas, a integração de sistemas robóticos em interações humanas, como jogos, pode abrir novas fronteiras para a tecnologia e proporcionar experiências enriquecedoras para a sociedade.

Referências

[1] International Organization for Standardization, [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-3:v1:en. [Acedido em 8 junho 2023].

[2] T. Kurfess, Robotics and Automation Handbook (1st ed.), CRC Press, 2005.

[3] K. Sirlantzis, L. Larsen, L. Kanumuru e P. R. Oprea, "Robotics," em *Handbook of Electronic Assistive Technology*, D. N. L. E. Cowan, Ed., London, Elsevier Ltd, 2019, pp. 311-345.

[4] J. Engelberger, Robotics in Practice, London, UK: Kogan Page Limited, 1980.

[5] M. Wilson, Implementation of Robot Systems: An introduction to robotics, automation, and successful systems integration in manufacturing., Elvesier Ldt, 2014.

[6] M. Ben-Ari e F. Mondada, Elements of Robotics, Cham: Springer, 2018.

[7] B. Dynamics, "Stretch," [Online]. Available: https://www.bostondynamics.com/products/stretch. [Acedido em 8 junho 2023].

[8] S. Saha, ROBOT APPLICATIONS, IIT Delhi, 2009.

[9] I. Surgical, "Robotic-Assisted Surgery with da Vinci Systems," [Online]. Available: https://www.intuitive.com/en-us/patients/da-vinci-robotic-surgery. [Acedido em 8 junho 2023].

[10] NASA, "Mars Perseverance Rover," [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/. [Acedido em 8 junho 2023].

[11] W. Khalil e E. Dombre, Modeling Identification & Control of Robots, London: Hermes Penton, 2002.

[12] V. M. F. Santos, Robótica Industria, Universidade de Aveiro, 2004.

[13] Graça Almeida, Automação e Robótica, Sebenta Sistemas Robóticos, Instituto Superior de Engenharia de Lisboa, 2019.

[14] J. Gallardo-Alvarado e J. Gallardo-Razo, Mechanisms Kinematic Analysis and Applications in Robotics, Elvesier Ldt, 2022.

[15] C. V. R. Coutinho, "Robótica Móvel – Sistema de Condução Autónoma," Instituto Superior de Engenharia de Lisboa, 2014.

[16] J. Webster e H. & Eren, Measurement, Instrumentation, and Sensors Handbook, CRC Press, 2014.

[17] B. S. Carlson, "Comparison of modern CCD and CMOS image sensor technologies and systems for low resolution imaging," em *SENSORS, IEE*, USA, 2002.

[18] A. d. Sousa, Smart Cameras as Embedded Systems, Universidade do Minho, 2002.

[19] A. J. M. Sousa, Arquitetura de Sistemas Robóticos e Localização em Tempo Real Através de Visão, Universidade do Porto, 2003.

[20] J. Craig e Addison-Wesley, Introduction to Robotics, Mechanics and Control, Addison-Wesley, 1986.

[21] R. Murray, Z. Li e S. Sastry, A Mathematical Introduction to Robotic Manipulation, CRC Press, 1994.

[22] A. M. Lopes, Modelação Cinemática e Dinâmica de Manipuladores de Estrutura em Série, 2002.

[23] A. C. Reddy, "DIFFERENCE BETWEEN DENAVIT - HARTENBERG (D-H) CLASSICAL AND MODIFIED CONVENTIONS FOR FORWARD KINEMATICS OF ROBOTS WITH CASE STUDY," em *International*

Conference on Advanced Materials and manufacturing Technologies (AMMT), JNTUH College of Engineering Hyderabad, 2014.

[24] R. Hartenberg e J. Denavit, Kinematic Synthesis of Linkages, 1964.

[25] G. Almeida, Sistema Robóticos - Sistema de Coordenadas DH, Instituto Superior de Engenharia de Lisboa, 2021.

[26] C. Wai-Kai, The Electrical Engineering Handbook, Boston: Elvesier Ldt, 2005.

[27] I. Labs, "Bringing Parallelism to the Web with River Trail," [Online]. Available: http://intellabs.github.io/RiverTrail/tutorial/#3D. [Acedido em 16 junho 2023].

[28] R. W. Gonzalez, Digital Image Processing, Boston: Pearson, 2008.

[29] R. Siegwart, I. Nourbakhsh e D. Scaramuzza, Introduction to Autonomous Mobile Robots, Cambridge: MIT Press, 2011.

[30] P. F. Subekti, S. P. Ary e A. S. Rohman, Design and Implementation of Machine Vision for Board Game in Lumen Social Robot System, IEEE, 2014.

[31] A. A. Elnaggar, M. Gadallah e M. A. Aziem, Autonomous Checkers Robot Using Enhanced Massive Parallel Game Tree Search, IEEE, 2014.

[32] J. McCabe, "Connect Four Robot," [Online]. Available: http://patrickmccabemakes.com/hardware/Connect_Four_Robot/. [Acedido em 16 junho 2023].

[33] S. Y. Technology, Dobot Magician User Guide V1.7.0, 2019.

[34] Niryo, "Ned2 User Manual," 2022. [Online]. Available: https://docs.niryo.com/product/ned2/v1.0.0/en/index.html. [Acedido em 31 07 2023].

[35] Niryo, “Simulate Ned/Ned2 with Matlab,” [Online]. Available: https://docs.niryo.com/applications/ned/v1.0.3/en/source/tutorials/SImulate_ned_with_matlab.html . [Acedido em 31 07 2023].

[36] Niryo, “PyNiryo Documentation v1.1.2,” 2022. [Online]. Available: https://docs.niryo.com/dev/pyniryo/v1.1.2/en/index.html. [Acedido em 31 07 2023].

[37] P. Corke, “Robotics Toolbox,” [Online]. Available: https://petercorke.com/toolboxes/robotics-toolbox/. [Acedido em 15 junho 2023].

[38] S. Kucuk e Z. Bingul, “Inverse kinematics solutions for industrial robot manipulators,” em *Applied Mathematical Modelling Volume 38, Issues 7–8*, ScienceDirect, 2014, pp. 1983-1999.

[39] X. Wang, D. Zhang e C. Zhao, “The inverse kinematics of a 7R 6-degree-of-freedom robot with non-spherical wrist,” em *Advances in Mechanical Engineering*, 2017.

[40] MathWorks, “Inverse Kinematics Algorithms,” MathWorks, [Online]. Available: https://www.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html. [Acedido em 21 07 2023].

[41] Trust, Folheto Comercial - Trust Trino Webcam HD.

[42] R. Fisher, S. Perkins, A. Walker e E. Wolfart, “Gaussian Smoothing,” [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm. [Acedido em 16 junho 2023].

[43] I. Muse, “HSV color space in 3D,” [Online]. Available: https://facelessuser.github.io/coloraide/colors/hsv/. [Acedido em 16 junho 2023].

[44] “StackOverflow,” [Online]. Available: https://stackoverflow.com/questions/21737613/image-of-hsv-color-wheel-for-opencv. [Acedido em 16 junho 2023].

[45] “OpenCV Forum,” [Online]. Available: https://answers.opencv.org/question/184711/select-hsv-hue-from-30-to-30-in-python/. [Acedido em 16 junho 2023].

[46] M. Gymrek, J. Li e E. Demaine, The Mathematics In Toys And Games, Massachusetts Institute of Technology, 2011.

[47] G. Strong, The Minimax Algorithm, Trinity College Dublin, 2011.

[48] C. Felstiner, Alpha-Beta Pruning, Whitman College, 2019.

Anexos

Para complementar o documento escrito, da presente dissertação de mestrado, são apresentados os seguintes anexos, que contém informações importantes sobre toda a construção e desenvolvimento do projeto final concluído.

- Anexo 1 – Apresenta o algoritmo da simulação da cinemática direta do manipulador, realizado em MATLAB;
- Anexo 2 – Apresenta o algoritmo da simulação da cinemática inversa do manipulador no caso do punho esférico, realizado em MATLAB;
- Anexo 3 – Apresenta o algoritmo da simulação da cinemática inversa do manipulador para o punho real através do modelo iterativo, realizado em MATLAB;
- Anexo 4 – Apresenta o algoritmo do sistema "Quatro em Linha com o Robot", realizado em python. Este é composto por 5 ficheiros:
    - main.py – ficheiro principal onde é executado o programa;
    - visao.py – ficheiro composto por todas as funções inerentes aos sistema de visão;
    - ecra.py – ficheiro composto por todas as funções que apresentam os resultados no ecrã;
    - jogo.py – ficheiro composto por todas as funções inerentes ao jogo do Quatro em Linha (por exemplo a função *Minimax*);
    - robot.py – ficheiro composto por todas as funções responsáveis por enviar comandos ao manipulador.

Anexo 1

```
% --------------- Instituto Superior de Engenharia de Lisboa -----------
% -------- Departamento de Engenharia Eletrotécnica Energia e Automação ------
% ------------------------------------------------------------------------
% -------------- Dissertação - Quatro em Linha com o Robot -----------------
% ---------------- Algoritmo de Cinemática Direta ----------------------------
% ------------------------------------------------------------------------
% ----------------- Mestrado em Engenharia Eletrotécnica ---------------
% ------------------------ Gonçalo Gomes - A44757 -------------------------


clear all;
clc;

%parametros DH
%       teta_i      d_i          a_i   alfa_i

ParDH =[0           166.3       0       90;
       90-3.108     0           221      0;
        3.108       0           32.5    90;
        0           235         0       -90;
        0           0           9.2     90;
        0           85+47       0       0];


PosDesejada = ParDH;
A06=1;
for i=1:6

      A =
rotz(PosDesejada(i,1)*pi/180)*transl(0,0,PosDesejada(i,2))*transl(PosDesejada(
i,3),0,0)*rotx(PosDesejada(i,4)*pi/180);
      A06=A06*A;


end;


%criacao do modelo do robot
L{1} =link([ParDH(1,4)*pi/180 ParDH(1,3) ParDH(1,1)*pi/180 ParDH(1,2)]);
L{2} =link([ParDH(2,4)*pi/180 ParDH(2,3) ParDH(2,1)*pi/180 ParDH(2,2)]);
L{3} =link([ParDH(3,4)*pi/180 ParDH(3,3) ParDH(3,1)*pi/180 ParDH(3,2)]);
L{4} =link([ParDH(4,4)*pi/180 ParDH(4,3) ParDH(4,1)*pi/180 ParDH(4,2)]);
L{5} =link([ParDH(5,4)*pi/180 ParDH(5,3) ParDH(5,1)*pi/180 ParDH(5,2)]);
L{6} =link([ParDH(6,4)*pi/180 ParDH(6,3) ParDH(6,1)*pi/180 ParDH(6,2)]);


S=robot(L); S.name='TRR-TRT';
q_repouso =
[ParDH(1,1)*pi/180,ParDH(2,1)*pi/180,ParDH(3,1)*pi/180,ParDH(4,1)*pi/180,
ParDH(5,1)*pi/180, ParDH(6,1)*pi/180];
```

```
drivebot(S,ParDH(:,1)'*pi/180);

%obtencao das equacoes
syms t1 d1 a1 alfa1
syms t2 d2 a2 alfa2
syms t3 d3 a3 alfa3
syms t4 d4 a4 alfa4
syms t5 d5 a5 alfa5
syms t6 d6 a6 alfa6

syms nx ox ax px ny oy ay py nz oz az pz
T=sym([nx ox ax px; ny oy ay py; nz oz az pz; 0 0 0 1]);

%matrizes de transformacao com as variáveis (a5=0)
I01= rotz(t1)*transl(0,0,d1)*transl(0,0,0)*rotx((sym(pi)/2));
I12= rotz(t2)*transl(0,0,0)*transl(a2,0,0)*rotx(0);
I23= rotz(t3)*transl(0,0,0)*transl(a3,0,0)*rotx((sym(pi)/2));
I34= rotz(t4)*transl(0,0,d4)*transl(0,0,0)*rotx(-(sym(pi)/2));
I45= rotz(t5)*transl(0,0,0)*transl(0,0,0)*rotx((sym(pi)/2));
I56= rotz(t6)*transl(0,0,d6)*transl(0,0,0)*rotx(0);

I03 =  simplify(I01*I12*I23);
I04 =  simplify(I01*I12*I23*I34);
I05 = simplify(I01*I12*I23*I34*I45);
I06 = simplify(I01*I12*I23*I34*I45*I56);

I10 = simplify(inv(I01));
I21 = simplify(inv(I12));
I32 = simplify(inv(I23));
I43 = simplify(inv(I34));
I54 = simplify(inv(I45));
I65 = simplify(inv(I56));


%primiera iteracao
T16= simplify(I10*T);
%I16=simplify(I12*I23*I34*I45*I56);
I16=I12*I23*I34*I45*I56;

%segunda iteracao
T26 = simplify(I21*T16);
I26=simplify(I23*I34*I45*I56);

%terceira iteracao
T36 = simplify(I32*T26);
I36=simplify(I34*I45*I56);

%quarta iteracao
T46 = simplify(I43*T36);
I46=simplify(I45*I56);

%quinta iteracao
T56 = simplify(I54*T46);
I56=simplify(I56);
```

Anexo 2

```
% ------------------- Instituto Superior de Engenharia de Lisboa -----------
% -------- Departamento de Engenharia Eletrotécnica Energia e Automação ------
% ---------------------------------------------------------------------------
% -------------- Dissertação - Quatro em Linha com o Robot -------------------
% ------------- Algoritmo de Cinemática Inversa para Punho Esférico ----------
% ---------------------------------------------------------------------------
% ----------------------- Mestrado em Engenharia Eletrotécnica ---------------
% ------------------------ Gonçalo Gomes - A44757 -------------------------


clear all
clc

%parametros DH

    %  teta_i       d_i     a_i        alfa_i
ParDH =[0           166.3   0           90;
       (90-3.108)   0       221         0;
       3.108        0       32.5        90;
       0            235     0           -90;
       0            0       9.2       90;
       0            85+47   0           0];


d1= ParDH(1,2);
a2= ParDH(2,3);
a3= ParDH(3,3);
d4= ParDH(4,2);
d6= ParDH(6,2);

%criacao do modelo do robot
L{1} =link([ParDH(1,4)*pi/180 ParDH(1,3) ParDH(1,1)*pi/180 ParDH(1,2)]);
L{2} =link([ParDH(2,4)*pi/180 ParDH(2,3) ParDH(2,1)*pi/180 ParDH(2,2)]);
L{3} =link([ParDH(3,4)*pi/180 ParDH(3,3) ParDH(3,1)*pi/180 ParDH(3,2)]);
L{4} =link([ParDH(4,4)*pi/180 ParDH(4,3) ParDH(4,1)*pi/180 ParDH(4,2)]);
L{5} =link([ParDH(5,4)*pi/180 ParDH(5,3) ParDH(5,1)*pi/180 ParDH(5,2)]);
L{6} =link([ParDH(6,4)*pi/180 ParDH(6,3) ParDH(6,1)*pi/180 ParDH(6,2)]);


%Posicao e orientacao desejada
nx = 0; ox = 1; ax = 0; px = -120;  %RPY = 0 3.14 -1.57
ny = 1; oy = 0; ay = 0; py = 196;   % ou = 3.14 0 1.57
nz = 0; oz = 0; az = -1; pz = 250;



%drop_pose_high (-121 196 250)
%2.12  0.13  -0.283  0.003  -1.41  0.57
```

```
%pick_pose_high (RPY IGUAL AO DROP_POSE // X Y Z = 0.23 0.104 0.25)
%0.43 0.03 -0.18 0 -1.41 -1.10

%start (todas as juntas a zero)
%X Y Z ROLL PITCH YAW
%0.38 0 0.427 -1.57 -1.57 -1.57


%calculo dos angulos
%teta 1
t1=atan2(py-d6*ay,px-d6*ax);

A = nx*cos(t1)+ny*sin(t1);
B = ox*cos(t1)+oy*sin(t1);
C = ax*cos(t1)+ay*sin(t1);
D = px*cos(t1)+py*sin(t1);
E = d1 - pz;
F = nx*sin(t1)-ny*cos(t1);
G = ox*sin(t1)+oy*cos(t1);
H = ax*sin(t1)-ay*cos(t1);

%teta 2
X = E + d6*az;
Y = d6*C-D;

K1 = 2*a2*Y;
K2 = 2*a2*X;
K3 = (a3^2)+(d4^2)-(a2^2)-(X^2)-(Y^2);

S2= ((K2*K3)-K1*sqrt((K2^2)-(K3^2)+(K1^2)))/((K1^2)+(K2^2));
C2= ((K1*K3)+K2*sqrt((K2^2)-(K3^2)+(K1^2)))/((K1^2)+(K2^2));
t2 = atan2(S2,C2);

%teta 3
W1 = (-sin(t2)*X)-(cos(t2)*Y)-a2;
W2 = sin(t2)*Y-cos(t2)*X;
t3 = atan2((W2*a3+W1*d4)/((a3^2)+(d4^2)),(W2*d4-W1*a3)/(-(a3^2)-(d4^2)));

%teta 4
S4 = H;
C4 = cos(t3)*(az*sin(t2)+cos(t2)*C)+sin(t3)*(az*cos(t2)-sin(t2)*C);
t4 = atan2(S4,C4)-pi;

%teta 4
S5 = cos(t4)*(cos(t3)*(az*sin(t2)+cos(t2)*C)+sin(t3)*(az*cos(t2)-sin(t2)*C)) +
(sin(t4)*H);
C5 = sin(t3)*(az*sin(t2)+cos(t2)*C)-cos(t3)*(az*cos(t2)-sin(t2)*C);
t5 = atan2(S5,C5);

%teta 6
S6 = -sin(t4)*(cos(t3)*(sin(t2)*nz+cos(t2)*A)-sin(t3)*(sin(t2)*A-cos(t2)*nz))+
cos(t4)*F;
C6 = -sin(t4)*(cos(t3)*(sin(t2)*oz+cos(t2)*B)-sin(t3)*(sin(t2)*B-oz*cos(t2)))+
cos(t4)*G;
t6 = atan2(S6,C6);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
S=robot(L); S.name='TRRTR-T';

q_rep =
[ParDH(1,1)*pi/180,ParDH(2,1)*pi/180,ParDH(3,1)*pi/180,ParDH(4,1)*pi/180,
ParDH(5,1)*pi/180, ParDH(6,1)*pi/180];
q_des = [t1,t2,t3,t4,t5,t6]
deg = rad2deg(q_des);


drivebot(S,ParDH(:,1)'*pi/180); %pos de repouso

T = [0:.225:15]; % given time vectorA 7th
%order polynomial is used with default zero boundary conditions for
%velocity and acceleration
traj = jtraj(q_rep, q_des, T); %  Returns a joint space trajectory Q
%from state Q0 to Q1.
plot(S, traj)


q_des
q_real =  [t1,  t2-(pi/2)+0.054,  t3-0.054,  t4,  t5,  t6]
```

Anexo 3

```
% ---------------- Instituto Superior de Engenharia de Lisboa -----------
% -------- Departamento de Engenharia Eletrotécnica Energia e Automação ------
% ---------------------------------------------------------------------------
% -------------- Dissertação - Quatro em Linha com o Robot ------------------
% ---------------- Algoritmo de Cinemática Inversa Modelo Iterativo ----------
% ---------------------------------------------------------------------------
% ------------------ Mestrado em Engenharia Eletrotécnica ---------------
% ------------------------ Gonçalo Gomes - A44757 -------------------------
clear all
clc



%parametros DH
%            a          alfa    d               teta
dhparams = [ 0          pi/2    0.1663          0;
             0.22132    0       0              pi/2;
             0.0325     pi/2    0               0;
             0          -pi/2   0.235           0;
             0.0092     pi/2    0               0;
             0          0       (0.047)  0];
```

```
         %limites das juntas do robot
limites = [ -2.949, 2949;
            -2.09, 0.61;
            -1.34, 1.57;
            -2.089, 2.089;
            -1.919, 1.922;
            -2.53, 2.53];


robot = robotics.RigidBodyTree;
bodies = cell(6,1);
joints = cell(6,1);


 %paramtrizacao do modelo do robot
 file_names =
char('shoulder_link.stl','arm_link.stl','elbow_link.stl','forearm_link.stl','w
rist_link.stl', 'hand_link.stl');
 pose = [[0 0 -0.0611];[-0.221 0 0];[0 -0.0325 0];[0 0 -0.1880];[0 -0.0092
0];[0 0 -0.0195-0.0092]];%XYZ
 angulo = [[0 0 -pi/2];[0 0 0];[0 -pi/2 -pi/2];[0 pi/2 pi/2];[0 -pi/2 -
pi/2];[0 0 0]]; %ZYX


 %criacao do modelo do robot
for i = 1:6
    bodies{i} = robotics.RigidBody(['body' num2str(i)]);
    joints{i} = robotics.Joint(['jnt' num2str(i)],'revolute');
    setFixedTransform(joints{i},dhparams(i,:),'dh');
    bodies{i}.Joint = joints{i};
    bodies{i}.Joint.HomePosition = dhparams(i,4);

    T = eul2tform(angulo(i,:), 'ZYX')*trvec2tform(pose(i,:));

    if i == 1 % Primeiro elo para a base
        addVisual(robot.Base, 'Mesh', 'base_link.stl')
        addVisual(bodies{i}, 'Mesh',file_names(i,:) , T)
        addBody(robot,bodies{i},'base');

    else
        addVisual(bodies{i}, 'Mesh',file_names(i,:) , T)
        addBody(robot,bodies{i},bodies{i-1}.Name);
    end

end



for lim = 1:6
    robot.Bodies{1, i}.Joint.PositionLimits = limites(i,:)
end



%criacao do orgao terminal com a garra
```

```
eeoffset = 0.085;
eeBody = robotics.RigidBody('end_effector');
setFixedTransform(eeBody.Joint, trvec2tform([0,0,eeoffset]));
addBody(robot, eeBody, 'body6');
T_M = getTransform(robot, robot.homeConfiguration,'end_effector','base')



ik = robotics.InverseKinematics('RigidBodyTree', robot);

for j = 1:6
    initial_guess{j} = getfield(robot.homeConfiguration, {j},'JointPosition');
end

%estimativa inicial (posicao de repouso)
initial_guess = num2cell(initial_guess);

%Erro da orientacao e posicao
weight = [0.1 0.1 0.1 0.1 0.1 0.1];

%Posicao e orientacao desejada
tform = [0,  1, 0,  -0.120;
         1,  0, 0,  0.196;
         0,  0, -1, 0.250;
         0, 0, 0, 1];


%modelo iterativo
[configSoln,solnInfo] = ik('end_effector', tform, weight,
robot.homeConfiguration);
cell = struct2cell(configSoln);
Joint = cell(2,:);
matrixJoints = cell2mat(Joint)

t1 = matrixJoints(1,1);
t2 = matrixJoints(1,2) - (pi/2)+0.054;
t3 = matrixJoints(1,3) - 0.054;
t4 = matrixJoints(1,4);
t5 = matrixJoints(1,5);
t6 = matrixJoints(1,6);

q_real = [t1, t2,t3,t4,t5,t6]

%mostrar figura
axes=show(robot,configSoln);
axes.XLim=[-0.5,0.5];
axes.YLim=[-0.5,0.5];
axes.ZLim=[0,0.5];
```

Anexo 4

main.py

```
# ---------------- Instituto Superior de Engenharia de Lisboa --------
# ----Departamento de Engenharia Eletrotécnica Energia e Automação ---
# --------------------------------------------------------------------
# -------------- Dissertação - Quatro em Linha com o Robot -----------
# ----------------------- Algoritmo main.py --------- ----------
# --------------------------------------------------------------------
# ------------------ Mestrado em Engenharia Eletrotécnica ------------
#------------------------ Gonçalo Gomes - A44757 ---------------------

import pygame
import sys
import time
from visao import Visao
from ecra import Ecra
from jogo import Jogo
from robot import Robot

game_over = False
pode_avancar = False
empate = False
NLIN = 6
NCOL = 7
NCIRCULOS = (NLIN*NCOL)+1


v = Visao()
e = Ecra(NLIN, NCOL)
j = Jogo(NLIN, NCOL)
r = Robot()

while not game_over:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            v.stop()
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_s  and not pode_avancar and
num_labels == NCIRCULOS:
                v.buscar_centros(centroids)
                v.start(e.screen)
                pode_avancar = True


    if not pode_avancar:
        img, th, num_labels, centroids = v.calibracao()
        v.mostrar_imagem(img,th, e.screen)
        e.draw_erro(num_labels, NCIRCULOS)

    else:
        e.draw_text(j.TURN)
        e.draw_board(j.board)
        if j.TURN == j.AI and not game_over:

            start_time = time.time()
```

```
            value, coluna_escolhida = j.minimax(j.board, j.PROF_INI,
j.ALFA_INI, j.BETA_INI, True)
            print("--- %s seconds ---" % (time.time() - start_time))
            print('coluna: ', coluna_escolhida)

            while not r.stopped:
                print('Robot ainda nao parou')
                time.sleep(2)

            r.start(coluna_escolhida)
            e.draw_text_col(coluna_escolhida)

        j.pos_validas = j.possiveis_jogadas(j.board)
        j.board = v.procurar_cor(j.board, j.pos_validas, j.TURN)


        if j.ver_se_acabou(j.board):
            game_over = True
            v.pos_validas = None

            e.draw_board(j.board)
            if len(j.ver_colunas_livres(j.board)) == 0:
                empate = True

            if not empate:
                e.draw_win(j.board, j.TURN)

            e.draw_text_end(empate, j.TURN)

            while not r.stopped:
                continue
            r.desligar()
            break

        j.TURN = (j.TURN + 1) % 2


v.stop()
time.sleep(5)
sys.exit()
```

visao.py

```
# ---------------- Instituto Superior de Engenharia de Lisboa --------
# ----- Departamento de Engenharia Eletrotécnica Energia e Automação -
# --------------------------------------------------------------------
# -------------- Dissertação - Quatro em Linha com o Robot -----------
# ----------------------- Algoritmo visao.py --------- ----------
# --------------------------------------------------------------------
# ------------------ Mestrado em Engenharia Eletrotécnica ------------
#------------------------ Gonçalo Gomes - A44757 ---------------------


import cv2 as cv
import numpy as np
from threading import Thread, Lock
import pygame
import sys
```

```
class Visao:

    cap = None
    NCOL = 7
    NLIN = 6
    matriz_centros = None
    pos_validas = None


    cap = cv.VideoCapture(1) #1 se pc tiver webcam integrada
    kernel = np.ones((5, 5), np.uint8)

    lower_red1 = np.array([0, 100, 100])
    upper_red1 = np.array([10, 255, 255])

    # upper boundary RED color range values; Hue (160 - 180)
    lower_red2 = np.array([160, 100, 100])
    upper_red2 = np.array([180, 255, 255])

    # boundary BLUE color range values; Hue (90 - 130)
    lower_blue = np.array([90, 50, 50])  # ini 90 50 50
    upper_blue = np.array([140, 255, 255])

    stopped = True
    lock = Lock()

    img = None
    mask_full = None
    mask_blue = None
    mask_red = None

    def capt_imagem(self):
        if not self.stopped:
            _, img = self.cap.read()
            if img is not None:
                img = cv.resize(img, (600, 420))

                hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

                # procurar pelas cores
                mask_red1 = cv.inRange(hsv, self.lower_red1,
self.upper_red1)
                mask_red2 = cv.inRange(hsv, self.lower_red2,
self.upper_red2)
                mask_red = mask_red1 + mask_red2

                mask_red = cv.erode(mask_red, self.kernel,
iterations=2)
                mask_red = cv.dilate(mask_red, self.kernel,
iterations=1)


                mask_blue = cv.inRange(hsv, self.lower_blue,
self.upper_blue)

                mask_blue = cv.erode(mask_blue, self.kernel,
iterations=2)
                mask_blue = cv.dilate(mask_blue, self.kernel,
iterations=1)
```

```
                mask_full = mask_red + mask_blue

                return img, mask_full, mask_red, mask_blue


    def buscar_centros(self,centroids):
        centroids = centroids[1:]  # exluir a label que representa o
fundo
        centroids = sorted(centroids, key=lambda x: x[0])  # ordenar
em x
        centroids = np.uint16(np.reshape(centroids, (self.NCOL,
self.NLIN, 2)))  # para poder ordenar em y
        matriz_centros = np.zeros((self.NLIN, self.NCOL, 2))



        for col in range(0, self.NCOL):
            ord_col = np.array(sorted(centroids[col], key=lambda y:
y[1]))  # ordenar em y
            for lin in range(0, self.NLIN):
                # print(f'{lin},{col}')
                matriz_centros[lin, col] = ord_col[lin]  # criar
matriz com coordenadas
                matriz_centros = np.uint16(matriz_centros)

        self.matriz_centros = matriz_centros


    def calibracao(self):

        _, img = self.cap.read()

        img = cv.resize(img, (600, 420))


        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        blur = cv.GaussianBlur(gray, (5, 5), 0)
        ret, th = cv.threshold(blur, 40, 255, cv.THRESH_BINARY_INV)
        th = cv.morphologyEx(th, cv.MORPH_OPEN, self.kernel)  # open -
erosao seguido de dilatacao

        num_labels, labels_im, stats, centroids =
cv.connectedComponentsWithStats(th)
        # print('Número de etiquetas: ',num_labels)


        return img, th, num_labels, centroids

    def procurar_cor(self, board, pos_validas, AI):
        board_ant = board.copy()
        detetou = False

        self.pos_validas = pos_validas

        while not detetou:
            if self.mask_blue is None or self.mask_red is None:
                continue

            for p in pos_validas:
                if not AI:
```

```
                    if self.mask_red[self.matriz_centros[p[0], p[1],
1], self.matriz_centros[p[0], p[1], 0]]:
                        board[p[0], p[1]] = 1

                else:
                    if self.mask_blue[self.matriz_centros[p[0], p[1],
1], self.matriz_centros[p[0], p[1], 0]]:
                        board[p[0], p[1]] = 2
                if board[p[0], p[1]] != board_ant[p[0], p[1]]:
                    detetou = True

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.stop()
                    sys.exit()

        return board

    def mostrar_imagem(self, img_normal, img_process, screen):

        if self.matriz_centros is not None and self.pos_validas is not
None:
            for p in self.pos_validas:
                cv.circle(img_normal, (self.matriz_centros[p[0], p[1],
0], self.matriz_centros[p[0], p[1], 1]), 2, (0, 255, 0), 2)

        pygame.draw.rect(screen, (254, 177, 99), (740, 5, 620, 440))
        pygame.draw.rect(screen, (254, 177, 99), (740, 455, 620, 440))
        rgb_img = cv.cvtColor(img_normal,
cv.COLOR_BGR2RGB).swapaxes(0, 1)
        pygame_img = pygame.surfarray.make_surface(rgb_img)
        mask_full = img_process.swapaxes(0, 1)
        pygame_mask = pygame.surfarray.make_surface(mask_full)
        screen.blit(pygame_img, (750, 15))
        screen.blit(pygame_mask, (750, 465))
        pygame.display.flip()




    def start(self, screen):
        self.stopped = False
        t = Thread(target=self.run, args=(screen,))
        t.start()

    def stop(self):

        self.stopped = True

    def run(self, screen):
        while not self.stopped:

            self.lock.acquire()
            try:
                img, mask_full, mask_red, mask_blue =
self.capt_imagem()
            except TypeError:
                pass
```

```
            if img is not None:
                self.img = img
                self.mask_full = mask_full
                self.mask_blue = mask_blue
                self.mask_red = mask_red
                self.lock.release()
                self.mostrar_imagem(self.img, self.mask_full, screen)

        self.cap.release()
```

ecra.py

```
# ---------------- Instituto Superior de Engenharia de Lisboa --------
# ----- Departamento de Engenharia Eletrotécnica Energia e Automação -
# --------------------------------------------------------------------
# -------------- Dissertação - Quatro em Linha com o Robot -----------
# ----------------------- Algoritmo ecra.py --------- ----------
# --------------------------------------------------------------------
# ------------------ Mestrado em Engenharia Eletrotécnica ------------
#------------------------ Gonçalo Gomes - A44757 ----------------------

import pygame

class Ecra:


    # janela
    pygame.init()
    size = (1600, 900)
    screen = pygame.display.set_mode(size)
    T_QUADRADO = 100
    RAIO = 40
    font = pygame.font.SysFont("monospace", 50)

    img_normal = None
    img_process = None


    def __init__(self, NLIN, NCOL):
        self.NLIN = NLIN
        self.NCOL = NCOL



    def draw_board(self, board):
        for lin in range(0, self.NLIN):
            for col in range(0, self.NCOL):
                pygame.draw.rect(self.screen, (254, 177, 99), (
                col * self.T_QUADRADO, 120 + (self.T_QUADRADO * lin),
self.T_QUADRADO, self.T_QUADRADO))  # total = 700x600
                if board[lin, col] == 1:
                    pygame.draw.circle(self.screen, (255, 0, 0), (col
* self.T_QUADRADO + 50, (lin * self.T_QUADRADO) + 170), self.RAIO)
                elif board[lin, col] == 2:
                    pygame.draw.circle(self.screen, (0, 0, 255), (col
* self.T_QUADRADO + 50, (lin * self.T_QUADRADO) + 170), self.RAIO)
                else:
```

```
                    pygame.draw.circle(self.screen, (20, 0, 0), (col *
self.T_QUADRADO + 50, (lin * self.T_QUADRADO) + 170), self.RAIO)
                text = self.font.render(f'C{col}', 1, (255, 255, 255))
                self.screen.blit(text, (col * self.T_QUADRADO + 20,
720))
        pygame.display.flip()


    def draw_erro(self, num_labels, NCIRCULOS):
            pygame.draw.rect(self.screen, (0, 0, 0), (100, 400, 630,
100))
            if num_labels > NCIRCULOS:
                text = self.font.render('Erro: Objetos a mais', 1,
(255, 0, 0))
                self.screen.blit(text, (100, 400))
            elif num_labels < NCIRCULOS:
                text1 = self.font.render('Erro: Tabuleiro não ', 1,
(255, 0, 0))
                text2 = self.font.render('detetado ou obstruído', 1,
(255, 0, 0))
                self.screen.blit(text1, (100, 400))
                self.screen.blit(text2, (100, 450))
            elif num_labels == NCIRCULOS:
                text = self.font.render('Tabuleiro detetado', 1, (0,
255, 0))
                self.screen.blit(text, (100, 400))

            pygame.display.flip()

    def draw_text_col(self,coluna):
        pygame.draw.rect(self.screen, (0, 0, 0), (coluna *
self.T_QUADRADO, 720, self.T_QUADRADO, self.T_QUADRADO))
        text = self.font.render(f'C{coluna}', 1, (0, 255, 0))
        self.screen.blit(text, (coluna * self.T_QUADRADO + 20, 720))
        pygame.display.flip()

    def draw_text(self, turn):
        if turn == 0:
            text = self.font.render('Vez do: Jogador 1', 1, (255, 0,
0))
        else:
            text = self.font.render('Vez do: Jogador 2', 1, (0, 0,
255))
        pygame.draw.rect(self.screen, (0, 0, 0), (120, 0, 600, 60))
        self.screen.blit(text, (120, 0))
        pygame.display.flip()

    def draw_win(self, board, turn):
        if turn:
            jogador = 2
            COR = (0, 150, 255)  # azul
        else:
            jogador = 1
            COR = (255, 80, 80)  # vermelho

        for col in range(0, self.NCOL - 3):  # verificar vencedor na
horizontal
            for lin in range(0, self.NLIN):
                if board[lin, col] == jogador and board[lin, col + 1]
== jogador and board[lin, col + 2] == jogador and board[
                    lin, col + 3] == jogador:
```

```
                    for a in range(0, 4):
                        pygame.draw.circle(self.screen, COR, ((a +
col) * self.T_QUADRADO + 50, (lin * self.T_QUADRADO) + 170),
self.RAIO)

        for lin in range(0, self.NLIN - 3):  # verificar vencedor na
vertical
            for col in range(0, self.NCOL):
                if board[lin, col] == jogador and board[lin + 1, col]
== jogador and board[lin + 2, col] == jogador and board[
                    lin + 3, col] == jogador:
                    for a in range(0, 4):
                        pygame.draw.circle(self.screen, COR, (col *
self.T_QUADRADO + 50, ((a + lin) * self.T_QUADRADO) + 170), self.RAIO)

        for lin in range(0, self.NLIN - 3):  # verificar vencedor na
diagonal negativa
            for col in range(0, self.NCOL - 3):
                if board[lin, col] == jogador and board[lin + 1, col +
1] == jogador and board[lin + 2, col + 2] == jogador and \
                        board[lin + 3, col + 3] == jogador:
                    for a in range(0, 4):
                        pygame.draw.circle(self.screen, COR, ((a +
col) * self.T_QUADRADO + 50, ((a + lin) * self.T_QUADRADO) + 170),
                                           self.RAIO)

        for lin in range(3, self.NLIN):  # verificar vencedor na
diagonal positiva
            for col in range(0, self.NCOL - 3):
                if board[lin, col] == jogador and board[lin - 1, col +
1] == jogador and board[lin - 2, col + 2] == jogador and \
                        board[lin - 3, col + 3] == jogador:
                    for a in range(0, 4):
                        pygame.draw.circle(self.screen, COR, ((a +
col) * self.T_QUADRADO + 50, ((lin - a) * self.T_QUADRADO) + 170),
                                           self.RAIO)
        pygame.display.flip()

    def draw_text_end(self, empate, turn):
        pygame.draw.rect(self.screen, (0, 0, 0), (120, 0, 600, 60))
        if empate:
            text = self.font.render('Empate!', 1, (255, 255, 0))
        else:
            text = self.font.render(f'Jogador {turn+1} vence!', 1, (0,
255, 0))
        self.screen.blit(text, (120, 0))
        pygame.display.flip()
```

jogo.py

```
# ---------------- Instituto Superior de Engenharia de Lisboa --------
# ------ Departamento de Engenharia Eletrotécnica Energia e Automação
# ------------------------------------------------------------------
# -------------- Dissertação - Quatro em Linha com o Robot -----------
# ---------------------- Algoritmo jogo.py --------- ----------
# ------------------------------------------------------------------
# ------------------ Mestrado em Engenharia Eletrotécnica ------------
#------------------------ Gonçalo Gomes - A44757 ---------------------

import numpy as np
```

```
import random
import math
from threading import Thread

class Jogo:

  game_over = False
  pos_validas = []
  # parametros jogo
  PLAYER = 0
  AI = 1
  PECA_PLAYER = 1
  PECA_AI = 2
  TURN = random.randint(0, 1)
  ALFA_INI = -math.inf
  BETA_INI = math.inf
  PROF_INI = 5



  T_JANELA = 4  # tamanho da janela para pontuação
  VAZIO = 0  # posiçao vazia ao avaliar pontuação



  def __init__(self, NLIN, NCOL):
    self.NLIN = NLIN
    self.NCOL = NCOL
    self.board = np.zeros((NLIN, NCOL))

  def ver_vitoria(self, board, peca, NLIN, NCOL):
    for col in range(0, NCOL - 3):  # verificar vencedor na horizontal
      for lin in range(0, NLIN):
        if board[lin, col] == peca and board[lin, col + 1] == peca and
board[lin, col + 2] == peca and board[
          lin, col + 3] == peca:
          # print(f'{lin}, {col}horzionta')
          return True

    for lin in range(0, NLIN - 3):  # verificar vencedor na vertical
      for col in range(0, NCOL):
        if board[lin, col] == peca and board[lin + 1, col] == peca and
board[lin + 2, col] == peca and board[
          lin + 3, col] == peca:
          return True

    for lin in range(0, NLIN - 3):  # verificar vencedor na diagonal
negativa
      for col in range(0, NCOL - 3):
        if board[lin, col] == peca and board[lin + 1, col + 1] == peca
and board[lin + 2, col + 2] == peca and \
              board[lin + 3, col + 3] == peca:
          return True

    for lin in range(3, NLIN):  # verificar vencedor na diagonal
positiva
      for col in range(0, NCOL - 3):
        if board[lin, col] == peca and board[lin - 1, col + 1] == peca
and board[lin - 2, col + 2] == peca and \
              board[lin - 3, col + 3] == peca:
          return True
```

```
  def se_coluna_cheia(self, board, coluna):
    return board[0, coluna] == 0  # ver se a coluna nao esta cheia

  def proxima_linha_livre(self, board, coluna):
    for lin in reversed(range(0, self.NLIN)):  # para começar pela
ultima linha)
      if board[lin, coluna] == 0:  # ver qual a linha que esta vazia
        return lin

  def ver_colunas_livres(self, board):  # verificar posiçoes possiveis
(colunas q n estao cheias)
    colunas_livres = []
    for col in range(0, self.NCOL):
      if self.se_coluna_cheia(board, col):
        colunas_livres.append(col)  # anexa colunas livres num vetor
    return colunas_livres

  def possiveis_jogadas(self, board):

    colunas_validas = self.ver_colunas_livres(board)
    linhas_validas = []

    for col in colunas_validas:
      linha = self.proxima_linha_livre(board, col)
      linhas_validas.append(linha)

    pos_validas = np.column_stack((linhas_validas, colunas_validas))

    return pos_validas

  def ver_se_acabou(self, board):  # verificar se nao ha mais espaço
ou se alguem ganhou
    return len(self.ver_colunas_livres(board)) == 0 or
self.ver_vitoria(board, self.PECA_PLAYER, self.NLIN, self.NCOL) or
self.ver_vitoria(board, self.PECA_AI, self.NLIN, self.NCOL)

  def largar_peca(self, board, linha, coluna, player):
    board[linha, coluna] = player

  def verificar_janela(self, janela, peca):
    score = 0
    if peca == self.PECA_AI:
      peca_opp = self.PECA_PLAYER
    else:
      peca_opp = self.PECA_AI

    if janela.count(peca) == 4:
      score += 100
    elif janela.count(peca) == 3 and janela.count(self.VAZIO) == 1:
      score += 5
    elif janela.count(peca) == 2 and janela.count(self.VAZIO) == 2:
      score += 1
    elif janela.count(peca_opp) == 3 and janela.count(self.VAZIO) ==
1:
      score -= 10

    return score
```

```
  def aval_score(self, board, peca, NLIN, NCOL):
    score = 0

    # da prioridade a coluna do meio
    coluna_central = [i for i in board[:, 3]]
    centro_count = coluna_central.count(peca)
    score += centro_count * 2

    # verificar na horizontal
    for lin in range(0, NLIN):
      vetor_linha = [i for i in board[lin, :]]
      for col in range(0, NCOL - 3):
        janela = vetor_linha[col:col + self.T_JANELA]
        score += self.verificar_janela(janela, peca)
    # verificar na vertical
    for col in range(0, NCOL):
      vetor_coluna = [i for i in board[:, col]]
      for lin in range(0, NLIN - 3):
        janela = vetor_coluna[lin:lin + self.T_JANELA]
        score += self.verificar_janela(janela, peca)
    # verificar na diagonal negativa
    for lin in range(0, NLIN - 3):
      for col in range(0, NCOL - 3):
        janela = [board[lin + i, col + i] for i in
range(self.T_JANELA)]
        score += self.verificar_janela(janela, peca)
    # verificar na diagonal positiva
    for lin in range(3, NLIN):
      for col in range(0, NCOL - 3):
        janela = [board[lin - i, col + i] for i in
range(self.T_JANELA)]
        score += self.verificar_janela(janela, peca)

    return score

  def minimax(self, board, depth, alfa, beta, maximazingPlayer):
    pos_val = self.ver_colunas_livres(board)
    is_over = self.ver_se_acabou(board)
    # print('\n', board)
    if depth == 0 or is_over:
      if is_over:
        if self.ver_vitoria(board, self.PECA_PLAYER, self.NLIN,
self.NCOL):
          return (-1000000 - depth, None)
        elif self.ver_vitoria(board, self.PECA_AI, self.NLIN,
self.NCOL):
          return (1000000 + depth, None)
        else:
          return (0, None)
      else:
        return (self.aval_score(board, self.PECA_AI, self.NLIN,
self.NCOL), None)

    if maximazingPlayer:
      maxEval = -math.inf  # menos infinito
      best_col = random.choice(pos_val)  # escolhe uma coluna das
posicoes possiveis
      for col in pos_val:
        linha = self.proxima_linha_livre(board, col)
        tela_copia = board.copy()
        self.largar_peca(tela_copia, linha, col, self.PECA_AI)
```

```
        eval, _ = self.minimax(tela_copia, depth - 1, alfa, beta,
False)
        if eval > maxEval:
          maxEval = eval
          best_col = col
        alfa = max(alfa, eval)
        if beta <= alfa:
          break
      return maxEval, best_col
    else:
      minEval = math.inf  # mais infinito
      best_col = random.choice(pos_val)  # escolhe uma coluna das
posicoes possiveis
      for col in pos_val:
        linha = self.proxima_linha_livre(board, col)
        tela_copia = board.copy()
        self.largar_peca(tela_copia, linha, col, self.PECA_PLAYER)
        eval, _ = self.minimax(tela_copia, depth - 1, alfa, beta,
True)
        if eval < minEval:
          minEval = eval
          best_col = col
        beta = min(beta, eval)
        if beta <= alfa:
          break
      return minEval, best_col
```

robot.py

```
# ---------------- Instituto Superior de Engenharia de Lisboa --------
# --- Departamento de Engenharia Eletrotécnica Energia e Automação ---
# -------------------------------------------------------------------
# -------------- Dissertação - Quatro em Linha com o Robot -----------
# ----------------------- Algoritmo robot.py --------- ----------
# -------------------------------------------------------------------
# ------------------ Mestrado em Engenharia Eletrotécnica ------------
#------------------------ Gonçalo Gomes - A44757 ---------------------

from pyniryo import *
from threading import Thread
import time
import math


class Robot:

    #robot
    ned = NiryoRobot("10.10.10.10")  # 169.254.200.200 para ethernet
    ned.set_arm_max_velocity(100)  # velocidade de junta 40%
    ned.calibrate_auto()
    tcp = [0.085, 0.0, 0.0, -3.14, -1.57, 0]
    ned.set_tcp(tcp)
    ned.enable_tcp(True)

    # coordenadas
     # x, y, z, roll(x), pitch(y), yaw(z)
    pick_pose = [0.23, 0.104, 0.02, -3.14, 0, 1.57]
    pick_pose_high = [0.23, 0.104, 0.25, -3.14, 0, 1.57]
    # -0.025 em Z = 0.0155
```

```
    drop_pose = [-0.122, 0.196, 0.18, -3.14, 0, 1.57]
    drop_pose_high = [-0.122, 0.196, 0.25, -3.14, 0, 1.57]
    start_pose = [0.38, 0.0, 0.427, -3.14, -1.57, 0.0]

    angulo = math.radians(0) #converter graus para radianos

    # nova variavel para largar peca
    new_drop_pose = drop_pose.copy()

    # colocar numa posicao inicial
    ned.move_pose(start_pose)
    ned.open_gripper(max_torque_percentage=5,
hold_torque_percentage=100)


    stopped = True

    def sequencia(self, coluna):
        print('Start:')
        #time.sleep(4)

        self.new_drop_pose[0] =  self.drop_pose[0] - (0.03 *
coluna)*math.sin(self.angulo)  # muda valor em x
        self.new_drop_pose[1] =  self.drop_pose[1] + (0.03 *
coluna)*math.cos(self.angulo)  # muda valor em y

        self.ned.move_pose(self.pick_pose_high)
        self.ned.move_linear_pose(self.pick_pose)  # apanhar
        self.ned.close_gripper(max_torque_percentage=5,
hold_torque_percentage=100)
        self.ned.move_linear_pose(self.pick_pose_high)
        self.ned.move_pose(self.drop_pose_high)
        self.ned.move_linear_pose(self.new_drop_pose)  # largar
        self.ned.set_arm_max_velocity(60)
        self.ned.shift_linear_pose(RobotAxis.Z, -0.025)
        self.ned.open_gripper(max_torque_percentage=5,
hold_torque_percentage=100)
        self.ned.move_linear_pose(self.drop_pose_high)
        self.ned.set_arm_max_velocity(100)
        print('end')
        self.stop()

    def desligar(self):
        print('desligar')

        self.ned.move_pose(self.start_pose)
        print('ok')
        self.ned.close_connection()

        self.stop

    def start(self, coluna):
        self.stopped = False
        t = Thread(target=self.run, args=(coluna,))
        t.start()

    def stop(self):
        self.stopped = True

    def run(self, coluna):
        self.sequencia(coluna)
```