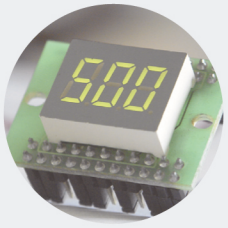
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Braço Robótico Controlado Por Reinforcement Learning

AFONSO MIGUEL ALCOBIA TIMÓTEO
outubro de 2024

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Braço Robótico Controlado Por *Reinforcement Learning*

Afonso Timóteo

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Outubro, 2024

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Automação e Sistemas.

Candidato: Afonso Timóteo, Nº 1190328, 1190328@isep.ipp.pt

Orientação Científica: Ramiro de Sousa Barbosa, rsb@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Outubro, 2024

Dedico este projeto ambicioso à minha família que me incentivou nos momentos mais difíceis e que foi determinante para o seu sucesso.

Agradecimentos

Expresso o meu agradecimento ao professor Ramiro de Sousa Barbosa pela colaboração e orientação prestadas ao longo de todo o processo de desenvolvimento deste projeto.

Não menos importante é o agradecimento à minha família mais próxima, pela paciência e incentivo dados ao longo destes meses de trabalho, com muitos altos e baixos.

Um sincero e afável agradecimento a todos os que de uma maneira ou de outra contribuíram para o sucesso deste projeto. Obrigado!

Resumo

Dado o crescimento promissor da área de Inteligência Artificial, ficaram disponíveis nos vários setores da nossa sociedade novas possibilidades. São inúmeras as empresas e entidades que utilizam esta área em aplicações do quotidiano, a tal ponto que impossibilita a sua deteção e reconhecimento.

Também no domínio da robótica, se assiste a um incremento das tecnologias de Inteligência Artificial que tornaram robôs, outrora limitados a tarefas repetitivas em linhas de montagem, capazes de realizar tarefas complexas e dinâmicas.

Na sua essência, este projeto procura explorar um pouco deste contexto atual, de modo a aplicar os conceitos investigados no desenvolvimento e implementação de um braço robótico controlado por algoritmos de inteligência artificial, para realizar tarefas interativas.

Assim, apresenta-se a conceção e a implementação de um sistema robótico integrado capaz de detetar objetos através de visão computacional, tomar decisões com base em estratégias e lógica e executar tarefas físicas, utilizando um braço robótico. O foco principal é o desenvolvimento de um módulo inteligente capaz de realizar uma tarefa simples interativa como o jogo do galo.

O sistema pode ser controlado através de uma *interface* gráfica que possibilita o acompanhamento em tempo real de todo o processo, facilitando a interação entre o utilizador e o braço robótico. Desde o controlo dos ângulos dos servo-motores ou da posição final do atuador-final, passando pela visualização das deteções nas imagens até à apresentação da melhor jogada para o estado do tabuleiro de jogo, esta *interface* proporciona um ambiente claro e compreensivo.

No final, apresentam-se fundamentalmente os resultados obtidos, evidenciando o controlo do braço robótico pelas equações de cinemática, o treino de um modelo de *YOLOv5* robusto e a capacidade de aprendizagem de um algoritmo de *Q-Learning*.

Palavras-Chave: Inteligência Artificial, Robótica, Braço Robótico, Cinemática Direta, Cinemática Inversa, Convenção de Denavit-Hartenberg, Visão Computacional, *YOLO*, *Q-Learning*, *MiniMax*, Jogo do Galo.

Abstract

Due to the promising growth in the field of Artificial Intelligence, new possibilities have become available in various sectors. Numerous companies and organizations are using this area in everyday applications, to such an extent that it is impossible to detect and recognize.

Also, in the field of robotics, there has been an increase in Artificial Intelligence technologies that have made robots, once limited to repetitive tasks on assembly lines, capable of performing complex and dynamic tasks.

In essence, this project seeks to explore some of this current context in order to apply the concepts investigated in the development and implementation of a robotic arm controlled by artificial intelligence algorithms to perform interactive tasks.

Thus, the design and implementation of an integrated robotic system capable of detecting objects through computer vision, making decisions based on strategies and logic and performing physical tasks, using a robotic arm, is presented. The main focus is the development of an intelligent module capable of performing a simple interactive task such as the Tic-Tac-Toe game.

The system can be controlled using a graphical interface that enables real-time monitoring of the entire process, facilitating interaction between the user and the robotic arm. From controlling the angles of the servomotors or the final position of the end-effector, to visualizing the detections in the images, to displaying the best move for the state of the game board, this interface provides a clear and comprehensive environment.

At the end, the results obtained are basically presented, highlighting the control of the robotic arm by the kinematics equations, the training of a robust YOLOv5 model and the learning capacity of a Q-Learning algorithm.

Keywords: Artificial Intelligence, Robotics, Robotic Arm, Direct Kinematics, Inverse Kinematics, Denavit-Hartenberg Convention, Computer Vision, YOLO, Q-Learning, MiniMax, Tic-Tac-Toe Game.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Lista de Acrónimos

3D	3 Dimensões
CNC	*Computer Numerical Control*
CNN	*Convolution Neural Network*
DC	Corrente Contínua
DH	Denavit-Hartenberg
DL	*Deep Learning*
DNN	*Deep Neural Network*
DOF	*Degree of Freedom* ou Grau de Liberdade
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
GPIO	*General-Purpose Input/Output*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
I2C	*Inter-Integrated Circuit*
IA	Inteligência Artificial
IBM	International Business Machines Corporation
IC	Circuito Integrado
ID3	Iterative Dichotomiser 3
IMU	Unidade de Medição de Inércia
IoU	*Intersection over Union*
KNN	*K-Nearest Neighbor*
LiDAR	*Light Detection and Ranging*

mAP	*Mean Average Precision*
ML	*Machine Learning*
NLG	*Natural Language Generation*
NLP	*Natural Language Processing*
NLU	*Natural Language Understanding*
OpenCV	Open-Source Computer Vision
PID	Proporcional, Integral e Derivativo
PPO	*Proximal Policy Optimization*
PWM	*Phase Width Modulation*
RAM	*Random Acess Memory*
RGB	*Red, Green, Blue*
RGB-D	*Red, Green, Blue - Depth*
RL	*Reinforcement Learning*
RNA	Redes Neuronais Artificiais
RNN	*Recurrent Neural Network*
RUR	*Rossum's Universal Robots*
SAE	Sociedade de Engenheiros Automóveis
SCARA	*Selective Compliance Assembly Robot Arm*
SPI	*Serial Peripheral Interface*
SRAM	*Static Random-Access Memory*
SSD	*Single Shot Multibox Detector*
SVM	*Support Vector Machines*
TTL	*Transistor-Transistor Logic*
UART	*Universal Asynchronous Receiver / Transmitter*
USART	*Universal Synchronous and Asynchronous Receiver-Transmitter*
USB	*Universal Serial Bus*
VS Code	Visual Studio Code
YOLO	*You Only Look Once*

Capítulo 1

Introdução

O avanço da tecnologia impulsionado pela área da Inteligência Artificial (IA), tem sido imparável, fator que tem revolucionado a sociedade e exigido uma constante adaptação do ser humano. A integração da IA nos mais diversos domínios tem permitido alcançar patamares até hoje impensáveis. São diversas as áreas que atestam a inovação desde a área da saúde até à rede de transportes.

Também no domínio da robótica, a IA está a permitir que robôs e derivados, outrora limitados a tarefas repetitivas em linhas de montagem, evoluam drasticamente, sendo capazes de realizar operações complexas, como intervenções cirúrgicas delicadas, processos de fabrico precisos e até missões de exploração espacial.

1.1 Contextualização

Em meados da década de 1950, começaram a surgir os primeiros conceitos de IA. Estas primeiras definições não eram idênticas às conhecidas atualmente.

IA é a capacidade de pensar, raciocinar, identificar padrões, por parte de máquinas criadas pelo Homem de uma maneira similar ao que um cérebro humano consegue atingir. Nesta linha desenvolveu-se a área de *Machine Learning* (ML), uma aplicação da IA que utiliza modelos de dados, de modo a ensinar os computadores a observar e encontrar padrões nas diferentes interações.

Nos finais da década de 80, introduziu-se outro conceito ao domínio de ML que iria catapultar a área da IA: *Deep Learning* (DL), embora somente no início do século XXI, é que a DL teve um incremento devido à implementação de Redes Neuronais

Artificiais (RNA), mais especificamente, as Redes Neuronais Profundas. Estas redes deram o protagonismo à área de DL devido aos bons resultados, comparativamente aos restantes membros da área de IA e, em certos campos, ultrapassarem os obtidos por seres humanos, "[...] deep neural networks reached an accuracy of classification of 90% compared to 50% for human experts." [1].

A criação destes conceitos de IA, conseguiu transformar máquinas simples e pré-programadas em sistemas inteligentes capazes de ações complexas e de se adaptarem ao seu contexto.

Atualmente, algoritmos de IA conseguem processar grandes quantidades de dados e aprender padrões complexos, o que os leva a serem aplicados às mais diversas áreas, como a saúde, a auxiliar nos diagnósticos e a criar tratamentos individualizados; a área automóvel, com a condução autónoma; a área financeira, a prever preços de mercado e tendências; entre outras áreas.

Uma área em que não se pode descurar o papel da IA é a área da robótica. A integração de algoritmos de IA é fundamental para que robôs possam interpretar e compreender os sinais dos seus sensores e interagir com o ambiente envolvente de uma forma segura e inteligente.

O conceito de braço robótico como é, normalmente conhecido hoje em dia, foi introduzido em meados da década de 50 e, desde esse ponto, a sua evolução é notória.

O Unimate, desenvolvido por George Devol em 1954, o primeiro braço robótico digitalmente operável e programável, desempenhou um papel fundamental na revolução dos processos de fabrico, permitindo automatizar tarefas repetitivas, simplificando processos e reduzindo o esforço humano. Este braço industrial foi utilizado na linha de fundição automatizada da General Motors em New Jersey, a primeira indústria a receber um braço robótico e abriu as portas ao mercado de robótica industrial [2].

Com o avançar da eletrónica ao longo dos anos, a introdução de microcomponentes tornou estes primeiros robôs aptos a tarefas mais complexas, em vez de estarem limitados a somente três graus de liberdade. Agora, era possível estes braços robóticos apresentarem 6 graus de liberdade e movimentos controlados eletricamente, em substituição dos antigos sistemas hidráulicos.

1.2 Definição do Problema

O domínio da área de robótica com controlo através de algoritmos de IA tem evoluído rapidamente e constantemente nos últimos anos. Investigadores nacionais e internacionais exploram continuamente novas formas de integrar sistemas de controlo inteligentes em pequenas máquinas, de modo a melhorarem pequenos aspetos da vida quotidiana.

No entanto, este domínio interessante nem sempre é simples ou rápido de compreender e aplicar. Geralmente, modelos de IA necessitam de longos tempos de treino e implementação de modo a estarem preparados para a maioria das eventualidades com que se podem deparar.

Este projeto procura explorar um pouco do contexto atual do domínio da integração da área de robótica e de IA de modo a aplicar os conceitos investigados no desenvolvimento e implementação de um braço robótico controlado por diversos algoritmos de IA para realizar tarefas interativas.

Assim, esta investigação centra-se na exploração do potencial da integração destas duas áreas num ambiente controlado, visando contribuir para a compreensão do modo como a IA pode ser aplicada para melhorar a interação robótica e as capacidades de tomada de decisões num ambiente interativo.

1.2.1 Objetivos

Com este propósito em mente, podem-se definir alguns marcos para enquadramento do projeto:

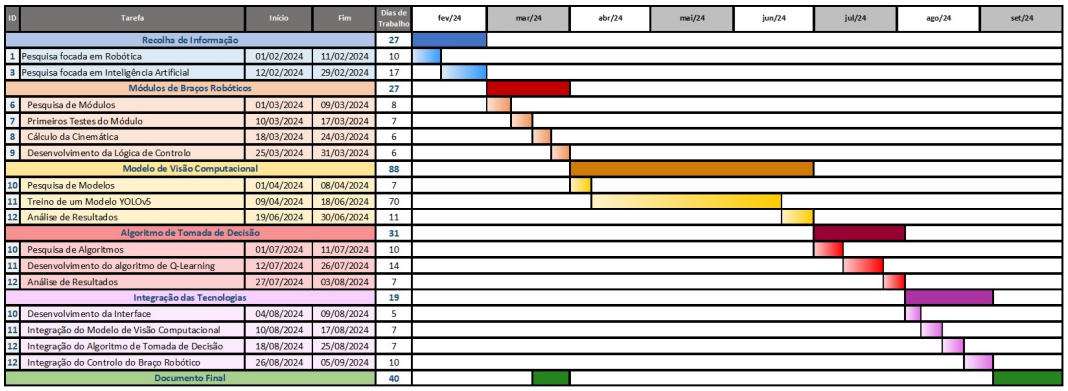
1. **Investigação**
    - **Área de Robótica** – Exploração do mundo da robótica procurando entender os princípios e componentes fundamentais, focando a estrutura, características e os diversos métodos de controlo por detrás dos braços robóticos.
    - **Área de IA** – Aprofundamento do conhecimento de alguns conceitos de IA procurando expor a sua definição e as suas características.
    - **Algoritmos de IA** – Pesquisa de diversos algoritmos existentes, procurando entender a sua implementação e as suas vantagens e desvantagens.
2. **Implementação**
    - **Controlo de um Braço Robótico** – Desenvolvimento do controlo do Braço Robótico, tendo em conta a pesquisa prévia, e desenvolvimento de uma interface para controlo interativo em tempo real que faça a interligação entre o utilizador e a máquina.
    - **Algoritmo de Visão Computacional** – Desenvolvimento de um algoritmo de visão computacional eficaz de modo a permitir decisões informadas por parte do controlador.
    - **Algoritmo de Tomada de Decisão** – Desenvolvimento de um algoritmo que permita ao controlador realizar decisões informadas para a tarefa designada.

3. **Análise de Resultados** – Análise da taxa de sucesso do controlador e dos seus algoritmos para a tarefa do jogo do galo.

1.3 Plano de Trabalho

Este projeto foi desenvolvido ao longo de sete meses e repartido por seis fases principais que se encontram representadas na Tabela 1.1.

Tabela 1.1: Calendarização do projeto



O projeto apresenta uma fase inicial de recolha de conhecimento relativo à área de robótica e de tecnologias de IA.

Posteriormente, a fase seguinte é caracterizada pela pesquisa e escolha de um módulo de um braço robótico que sirva como base do sistema. Nesta fase, realizaram-se ainda os primeiros testes de funcionamento do braço robótico, os cálculos das equações de cinemática que descrevem o robô e o desenvolvimento da lógica de controlo do mesmo.

Na terceira fase, deu-se início ao treino do modelo de visão computacional e análise de resultados.

A quarta fase é descrita pelo desenvolvimento do algoritmo responsável pela tomada de decisões e sua implementação no projeto.

A quinta etapa deste projeto inclui a integração das tecnologias anteriores e implementação do sistema.

Por fim, elaborou-se o relatório que descreve os procedimentos e resultados da implementação do sistema desenvolvido.

1.4 Organização da Dissertação

Este documento encontra-se divido em oito capítulos que permitem ao leitor a compreensão e o acompanhamento de todas as etapas de desenvolvimento do projeto.

O primeiro capítulo expõe o problema que este projeto pretende resolver, os objetivos delineados e a calendarização de cada fase de implementação do trabalho desenvolvido.

O segundo, foca-se na descrição e apresentação do mundo da robótica, evidenciando a evolução da área, as características e componentes principais dos sistemas robóticos e as suas categorias.

No terceiro capítulo, explora-se a área da IA dando relevância às bases da tecnologia e a alguns algoritmos.

No quarto capítulo enumeram-se e analisam-se diversos módulos de braços robóticos que apresentam os requisitos identificados para a execução deste projeto. Também neste capítulo, apresentam-se os primeiros testes e cálculos realizados ao módulo selecionado para a base do projeto.

O quinto capítulo apresenta a arquitetura principal do sistema, fundamentando as escolhas de cada componente e ferramenta.

No sexto capítulo apresenta-se, ao leitor, alguns detalhes do sistema desenvolvido, focando o controlo do braço robótico, o treino do modelo de visão computacional, o desenvolvimento do algoritmo de tomada de decisão e o desenho e implementação da aplicação.

O sétimo capítulo foca-se na apresentação e análise dos resultados.

Por fim, no último capítulo apresentam-se as conclusões e melhorias possíveis de realizar.

Capítulo 2

Robótica

A robótica, segundo a sua definição, é um subdomínio da engenharia focado na conceção, construção, funcionamento e aplicação de robôs.

Os avanços tecnológicos das últimas décadas têm dotado os robôs de capacidades, outrora exclusivas do ser humano, tornando-os cada vez mais sofisticados, fatores que provocaram um crescimento exponencial da área.

Cada sistema robótico ideal deve seguir a lógica básica que rege os seres vivos: sentir, pensar e agir. Desta forma, em primeiro lugar o robô deve sentir o ambiente envolvente com os seus sensores, tentando obter o máximo de informação do estado em que se encontra. Em seguida, o controlador principal deve decidir uma resposta à altura da tarefa a realizar. Por último, baseado na decisão, os atuadores do sistema executam a ação escolhida.

Olhando criticamente para este *status quo*, pode-se constatar que a robótica é uma peça fundamental em várias indústrias e sectores e revoluciona a forma como estas operam. Desde a indústria transformadora, passando pelos cuidados de saúde até à exploração e ao entretenimento, diversas categorias de robôs especializados desempenham um papel fundamental na vida humana.

2.1 Evolução da Robótica

O conceito primordial de robótica remonta às civilizações antigas, onde não só mitos e lendas descreviam seres mecânicos com capacidades semelhantes às humanas, como também pequenas invenções hidráulicas e mecânicas demonstravam o princípio de

um conceito que só séculos mais tarde seria desenvolvido [3, 4]. Na Figura 2.1 apresentam-se alguns exemplos de invenções anteriores à introdução do conceito de robótica.

Figura 2.1: Vaso antigo apresentando Talos (esquerda) e o cavaleiro robótico de Leonardo da Vinci (direita) [5]

Foi no início do século XX que o dramaturgo checo Karel Čapek apresentou, pela primeira vez, o termo "robô" na sua peça de 1921 *Rossum's Universal Robots* (RUR). O termo deriva das palavras checas *robota* que significa trabalho forçado e *robotnik* que se pode traduzir para servo ou trabalhador. A peça procurava protestar contra o rápido avanço da tecnologia moderna e, assim, retratou um futuro em que robôs criados para servir os seres humanos se revoltavam contra os seus criadores [6, 7].

No entanto, a popularização do conceito de robótica aconteceu na década de 1940, pelas mãos do escritor Isaac Asimov. Este apresentou-o nas suas histórias em que retratava um futuro mais otimista que o dramaturgo checo. Estes seres mecânicos inteligentes cumpriam as suas funções de serviço à raça humana. Estas histórias apresentavam também as três regras fundamentais para o comportamento dos robôs [8, 9]:

1. Um robô não pode ferir um ser humano ou, por inação, permitir que um ser humano seja ferido.

2. Um robô deve obedecer às ordens que lhe são dadas por humanos, exceto se isso entrar em conflito com a primeira lei.

3. Um robô deve proteger a sua própria existência, desde que isso não entre em conflito com a primeira ou a segunda lei.

O real desenvolvimento de sistemas robóticos começou em meados do século XX, impulsionado pelos avanços na eletrónica, na teoria do controlo e na computação digital. E assim a era dos robôs começou com o pedido de patente do primeiro manipulador por George Devol em 1954. O Unimate era capaz de efetuar movimentos repetidos "ponto a ponto" e combinava as características de reprodução das máquinas de controlo numérico com a tecnologia de servo-controlo e os mecanismos

articulados controlados à distância, presente na Figura 2.2. Em 1962 Joe Engleberger, que tinha anos antes adquirido os direitos da patente, conseguiu um contrato com a General Motors para instalar este primeiro robô numa das suas linhas de montagem, uma linha de fundição sob pressão. O Unimate melhorou de tal forma a produção que motivou outras empresas e centros de investigação a dedicar ativamente recursos à área da robótica [2].

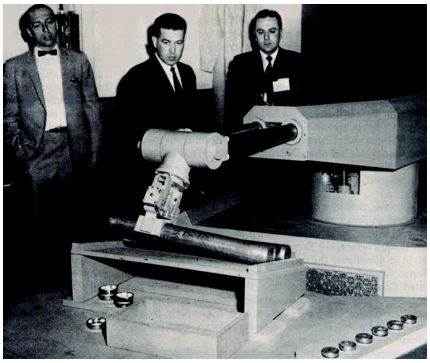


Figura 2.2: George Devol e o Unimate [10]

As décadas seguintes trouxeram grandes avanços e a introdução e desenvolvimento de robôs mais sofisticados, capazes de executar tarefas complexas com elevada precisão. Durante este período, salienta-se, por exemplo, a integração de sensores que ofereceram a capacidade de reação e resposta aos fatores do ambiente envolvente e a introdução de controladores dedicados, tornando os robôs máquinas reprogramáveis.

Por volta da década de 1990 já se considerava que este campo tinha o potencial de se expandir para além das aplicações industriais, podendo ser introduzido em inúmeras áreas, desde a área da saúde até ao entretenimento [8]. Alguns exemplos icónicos incluem os *rovers* de Marte Spirit e Opportunity, que demonstraram o potencial da robótica na exploração espacial, e o robô cirúrgico Da Vinci (Figura 2.3), que revolucionou a cirurgia minimamente invasiva.

Figura 2.3: Sistema cirúrgico Da Vinci [9]

Desde os primeiros conceitos das civilizações antigas até aos complexos robôs modernos, a área da robótica está marcada pela constante inovação e progresso. Atualmente, esta é um domínio extenso que está focado em robôs inteligentes [5]. A

convergência da engenharia mecânica, das ciências da computação e da inteligência artificial promete sistemas mais sofisticados e altamente versáteis num futuro em que os robôs se apresentam como indispensáveis.

2.2 Características e Componentes de Sistemas Robóticos

O objetivo da robótica é conceber máquinas que possam assistir e auxiliar os seres humanos e, para este efeito, a exemplo do ser humano que segue uma lógica prévia a realizar uma ação, cada sistema robótico deve sentir primeiramente o ambiente envolvente para depois decidir uma resposta à altura. Por último, baseado na decisão o sistema executa a ação escolhida [11].

Assim, cada sistema robótico é constituído por vários componentes que trabalham em conjunto para realizar as suas tarefas de uma forma autónoma. Estes componentes podem variar consoante a categoria do robô e o fim a que se destina, porém, o seu propósito essencial mantém-se o mesmo. Alguns componentes comuns em muitos sistemas incluem uma estrutura mecânica, sensores, sistemas de controlo e, finalmente, os atuadores.

2.2.1 Estruturas Mecânicas

A estrutura de um robô é o primeiro componente e a base sobre a qual todos os outros componentes assentam; define a forma e a disposição do robô, influenciando o seu desempenho e capacidades. Esta tem de ser concebida para suportar todos componentes mecânicos do robô e ao mesmo tempo suportar os fatores ambientais externos e as cargas de operação.

A conceção da estrutura afeta a sua durabilidade e precisão de operação, assim, a escolha dos materiais para a composição da estrutura e as capacidades dessa estrutura são cruciais para o desempenho geral do robô.

Relativamente à escolha dos materiais, existem diversos fatores a ter em conta, como as condições do ambiente de trabalho e a natureza da tarefa a realizar, que condicionam a escolha e eficácia dos vários tipos de materiais. Embora os materiais leves aumentem a agilidade do robô, podem não ter a resistência necessária para tarefas pesadas; os metais mais pesados, apesar de oferecem uma maior durabilidade, podem afetar a velocidade e agilidade do sistema [12].

Nesta categoria de materiais incluem-se os metais, que oferecem elevada resistência e durabilidade, como o alumínio, o aço e ligas metálicas; plásticos e derivados, que oferecem opções leves e flexíveis; e materiais como a fibra de carbono e os polímeros de elevado desempenho, que oferecem um equilíbrio entre peso, resistência e flexibilidade [13].

As capacidades de um robô são diretamente influenciadas pelos seus *Degree of Freedom* ou Grau de Liberdade (DOF), que se referem ao número de movimentos independentes que um robô ou uma parte de um robô pode efetuar. Cada grau de liberdade representa uma direção específica, em que cada robô se pode mover ou rodar. Sendo assim, quanto mais graus de liberdade um robô tiver, mais complexos e versáteis podem ser os seus movimentos. No entanto, isto também aumenta a complexidade do controlo e da conceção [14, 15].

Para aumentar os DOF é necessário aumentar o número de juntas/articulações de um robô, uma vez que se pode definir as juntas como as ligações móveis entre elos que permitem o movimento numa nova direção. Estes elos são os componentes rígidos de um robô que transmitem força e movimento entre as articulações e determinam a estrutura geral do robô, condicionando as suas capacidades e habilidades para a execução de tarefas.

2.2.2 Sensores

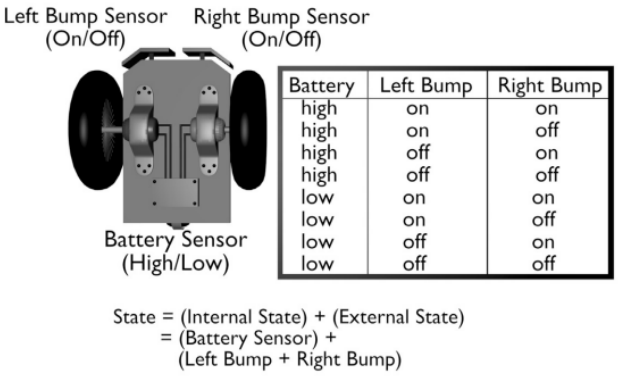
Os sensores são os equipamentos físicos que permitem a um robô recolher informações do ambiente que o rodeia, a exemplo do ser humano podem ser descritos como os órgãos sensoriais de um robô. Identificam-se dois conceitos "deteção" e "perceção" que são tratados como sinónimos em robótica, uma vez que ambos se referem ao processo de receção de informação através de sensores, a diferença é que perceção é o processo de receber informação sobre o mundo exterior, enquanto a deteção remete para o conhecimento do estado do robô [11].

O estado é uma noção geral da física e refere-se à descrição de um sistema. O estado de um robô é, assim, uma descrição de si próprio num determinado momento. Quanto mais detalhada for esta descrição, maior será o estado e, por isso, serão necessários mais bits ou símbolos para o descrever. O termo Espaço de Estados engloba todos os estados possíveis em que um sistema se pode encontrar, de acordo com os seus sensores, como se observa na Figura 2.4, e é essencial para organizar as informações para a atuação do sistema de controlo [11].

Dependendo da tarefa que o robô pretende executar, existem diversas gamas de sensores idealizadas para atingir diferentes objetivos. Alguns exemplos incluem sensores de posição, sensores de proximidade, sensores de força, sistemas de visão, entre outros.

Sensores de Posição

Os sensores de posição focam-se em determinar a localização e orientação exatas de um sistema ou das suas peças. São estes sensores que garantem movimentos precisos e controlados. Existem vários formatos de sensores, cada um com características e vantagens distintas, adequados para diferentes aplicações.



Battery	Left Bump	Right Bump
high	on	on
high	on	off
high	off	on
high	off	off
low	on	on
low	on	off
low	off	on
low	off	off

Figura 2.4: Espaço de estados de um sistema [11]

Começando pelos *Encoders*, estes são dos sensores de posição mais comuns, convertem movimento efetuado num sinal digital ou analógico representativo da posição, movimento este que pode ser a rotação do veio de um motor ou a deslocação de outras peças móveis [16].

Os potenciómetros são outro tipo de sensores de posição, no entanto, estes traduzem o movimento mecânico numa alteração da resistência elétrica. São simples e económicos, mas menos precisos que os *encoders*, sendo adequados para aplicações onde a alta precisão não é crítica, por isso, geralmente encontram-se em circuitos para ajustar a sensibilidade e as propriedades de outros sensores, por exemplo [11].

Relativamente à Unidade de Medição de Inércia (IMU), esta combina giroscópios e acelerómetros (e, opcionalmente, magnetómetros e barómetros, entre outros) para detetar o movimento, a orientação e a posição de um sistema, como se demonstra na Figura 2.5. Estes sensores são essenciais para robôs móveis e *drones*, onde manter o equilíbrio e navegar pelo espaço é crucial [17, 18].

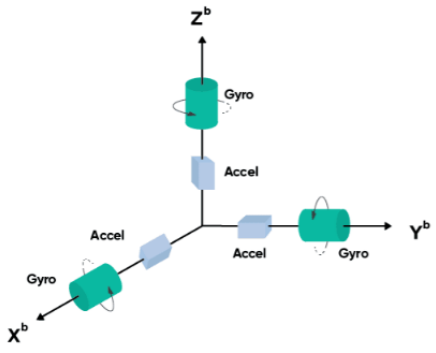


Figura 2.5: Sistema IMU [19]

Uma desvantagem destas unidades diz respeito ao modo de utilização dos campos magnéticos para calcular valores; estes podem ser afetados por campos magnéticos

gerados por equipamentos como motores, estruturas metálicas, entre outros, o que pode resultar em erros indesejados, precisando de calibração adicional.

Sensores de Proximidade

Os sensores de proximidade têm o objetivo de detetar a presença de objetos próximos no ambiente envolvente, tendo um papel fundamental na navegação e na prevenção de obstáculos. Existem vários modelos que procuram atingir este objetivo, desde sensores ultrassónicos, sensores de luz e sensores capacitivos e indutivos, fornecendo informações valiosas e distintas para o sistema de controlo analisar.

Falando um pouco dos sensores ultrassónicos, os sonares enviam sinais acústicos para detetar objetos e medir as distâncias entre o sensor e os obstáculos. São constituídos por duas partes principais, um transmissor e um recetor [18].

Enquanto o transmissor é responsável pelo envio de impulso ultrassónico, o recetor recebe a reflexão do sinal causada por um obstáculo. O sensor calcula posteriormente o tempo decorrido entre a transmissão e a receção do sinal, tendo em conta a velocidade do som no meio. Desta forma, consegue-se calcular a distância a que se encontra o obstáculo do sensor pela equação (2.1).

$$\text{Distância} = \frac{Tempo \times Velocidade_{Ar}}{2} \tag{2.1}$$

A principal desvantagem destes sensores é o facto de serem sensíveis ao ruído do ambiente circundante que pode causar interferências e alterar os valores calculados. No entanto, continuam a ser amplamente aplicados em pequenos robôs autónomos, como se demonstra na Figura 2.6.

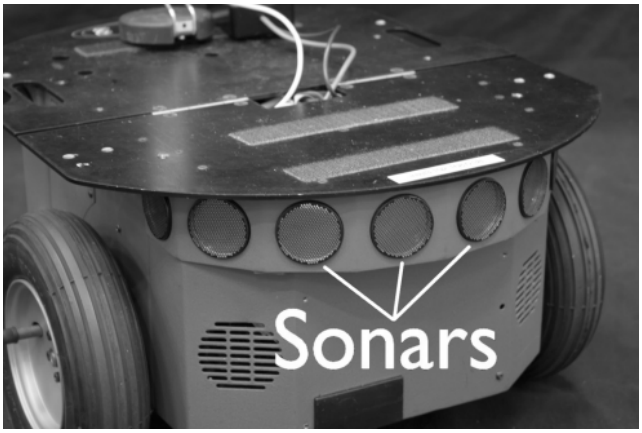


Figura 2.6: Veículo autónomo com sensores ultrassónicos [11]

Relativamente aos sensores de luz, estes fazem uso de energia luminosa para detetar objetos. Podem-se distinguir dois tipos de sensores de luz, tendo em conta o modo de funcionamento: os sensores ativos, a exemplo dos sonares, consistem na transmissão, desta vez de um feixe de luz e na deteção da sua reflexão; por outro

lado, os sensores passivos detetam exclusivamente a intensidade de luz. As foto-resistências, um exemplo de sensores passivos, consoante o aumento da intensidade de luz detetada, diminuem a sua resistência interna [18].

Alguns exemplos de sensores de luz ativos incluem uma grande maioria dos sensores infravermelhos e dos sensores *laser* que envolvem a transmissão de um feixe de luz específica para a deteção de obstáculos, como mostrado na Figura 2.7. Por serem sensores simples e eficazes para a deteção de curto alcance, aparecem, frequentemente, em pequenos robôs e sistemas automatizados [20].

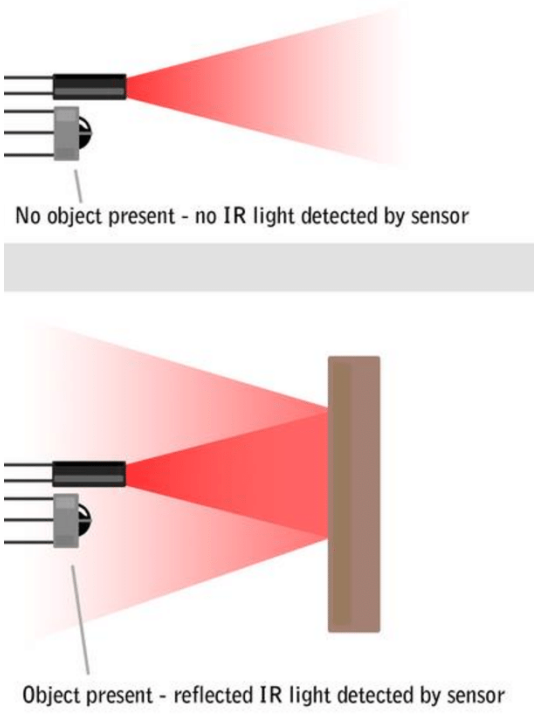


Figura 2.7: Sensor infravermelho [21]

Uma das grandes desvantagens destes sensores é a sua sensibilidade a interferências de luz ambiente, de um modo similar aos sensores ultrassónicos.

Por fim, os sensores capacitivos e indutivos detetam objetos com base nas alterações dos campos de força de cada sensor [16]. Assim, por um lado, os sensores capacitivos detetam objetos que alteram o campo elétrico criado pelo sensor, como plásticos, vidros e líquidos, o que os torna úteis em aplicações de fabrico e embalagem.

Por outro lado, os sensores indutivos detetam objetos metálicos através de alterações no seu campo eletromagnético, o que os leva a ser frequentemente utilizados em ambientes industriais para detetar a presença de peças metálicas em linhas de montagem.

Sensores de Força

Um sensor de força é normalmente utilizado para medir a força exercida num sistema, por exemplo, sob a forma de uma carga, pressão ou tensão.

Quando a força exercida é sobre a forma de uma carga, as células de carga são os sensores mais adequados, convertendo a força exercida sobre elas num sinal elétrico [22]. São frequentemente utilizadas em aplicações de pesagem, mas também podem ser integradas em sistemas robóticos para medir a força exercida durante operações como pressionar, puxar ou levantar [16].

Os medidores de tensão ou os extensómetros são um dos tipos mais comuns de sensores de força. Operam através da medição da deformação (tensão) de um material quando sujeito a uma força. Esta deformação vai provocar uma alteração da resistência elétrica do sensor, que se pode traduzir em dados de força.

Além disso, para pequenas variações de força ou pressão, os sensores táteis são os mais adequados devido à sua elevada sensibilidade. Um exemplo são os sensores piezoelétricos que geram uma carga elétrica em resposta ao esforço mecânico aplicado [22]. Os constituintes deste sensor estão presentes na Figura 2.8.

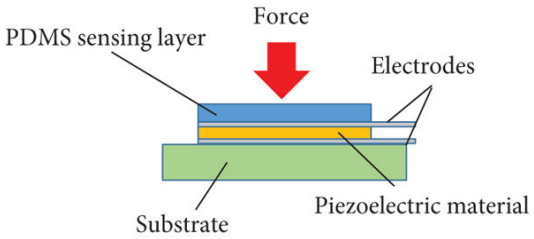


Figura 2.8: Sensor piezoelétrico [23]

Sistemas de Visão

Os sistemas de visão fazem uso de tecnologias e sensores, referidos anteriormente, para permitir aos sistemas reconhecer objetos, navegar em ambientes complexos, entre outras atividades em que simples sensores não teriam a mesma eficácia.

Quando se fala em sistemas de visão, a primeira ideia que fica é de que se tratam de câmaras, no entanto, existem outros tipos de sensores que atingem os mesmos objetivos.

O *Light Detection and Ranging* (LiDAR) é um sistema de visão que tem estado cada vez mais presente na área da robótica, incluindo na deteção de objetos, na prevenção de obstáculos, no mapeamento e na captura de movimentos em 3 dimensões, como se demonstra na Figura 2.9. Este sistema emite impulsos de luz laser e mede o tempo demorado entre a emissão e o regresso da luz refletida [17].

Regra geral, é um sistema com elevada precisão ideal para mapeamento e navegação em 3 dimensões, uma vez que permite a deteção dos raios de luz emitidos em

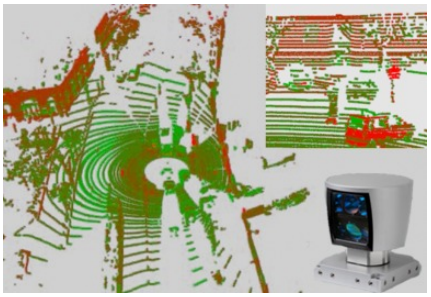


Figura 2.9: Dados de um sistema LiDAR [24]

várias direções simultâneas, como se pode observar na Figura 2.10. Contudo, apresenta alguns pontos negativos: exige uma elevada capacidade computacional para processar dados, o que pode afetar o desempenho em tempo real dos robôs; a radiação pode causar interferências e leituras erradas; é difícil distinguir a luz emitida do próprio sensor ou de raios de outros sensores próximos [18].

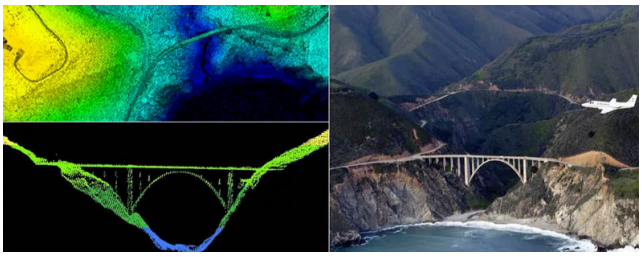


Figura 2.10: Dados de um sistema LiDAR para navegação e mapeamento [25]

É uma tecnologia muito versátil e tem encontrado o seu lugar em áreas muito distintas desde aplicações espaciais e de astronomia até cartografia e arqueologia, não esquecendo os veículos autónomos, onde o seu papel de mapeamento do ambiente envolvente não pode ser descurado.

No que concerne aos sistemas baseados em câmaras, estes podem fornecer informações valiosas sobre o ambiente do robô, uma vez processadas por algoritmos adequados. Conhecem-se diferentes tipos de sistemas, como câmaras monoculares, sistemas *Stereo*, sistemas *Red, Green, Blue - Depth* (RGB-D), entre outros, que se adaptam a todo o tipo de aplicações robóticas [16].

As câmaras monoculares são das mais comuns, são caracterizadas por possuírem uma lente única e captam imagens em duas dimensões, como o exemplo presente na Figura 2.11. Estas câmaras são simples e económicas especialmente adequadas para aplicações em que espaço e peso são fundamentais. Contudo, as câmaras monoculares fornecem informações limitadas sobre a profundidade, o que pode dificultar a perceção 3 Dimensões (3D) [18].

Para colmatar este ponto, existem os sistemas *Stereo* que consistem em duas ou mais câmaras monoculares idênticas espaçadas entre si. Ao comparar as imagens

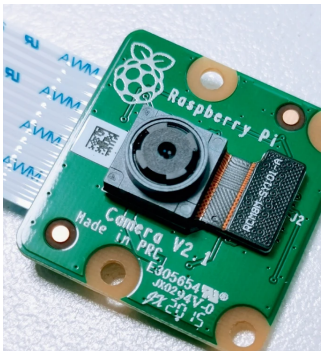


Figura 2.11: Exemplo de uma câmara monocular [18]

captadas por cada lente, as câmaras *Stereo* podem estimar a profundidade dos objetos na cena, fornecendo uma visão 3D do ambiente, segundo o princípio da paralaxe. Isto torna-as úteis para tarefas como a deteção de obstáculos e a navegação [11].

Relativamente aos sistemas RGB-D, estes são compostos por uma câmara *Red, Green, Blue* (RGB) para fornecer imagens coloridas, e sensores de proximidade (em especial os sensores de luz ativos) para fornecer os dados de profundidade, ou seja, o mapa de profundidade [26]. Na Figura 2.12 está presente uma análise do resultado deste sistema.

Figura 2.12: Vista de uma câmara RGD-D (da esquerda para a direita - imagem RGB, imagem de profundidade, imagem infravermelha com o padrão projetado) [18]

Estes sistemas têm sido muito populares na área da robótica para processamento de imagens em tempo real, localização de robôs e prevenção de obstáculos. No entanto, estes sistemas têm um alcance de ação limitado e são sensíveis à radiação exterior.

Resumindo, a panóplia de sensores é imensa com diversas funcionalidades e consequentemente pontos positivos e negativos, assim como aplicações específicas. É importante notar que é a integração responsável de sensores que permite que os robôs operem de forma autónoma e adaptativa, respondendo a ambientes dinâmicos e realizando tarefas complexas com elevada precisão e fiabilidade.

2.2.3 Sistemas de Controlo

Os sistemas de controlo são o "cérebro" dos sistemas robóticos. Estes são responsáveis por processar as entradas dos sensores, determinar as ações apropriadas e

comandar a execução da resposta. Estes sistemas são essenciais para que um robô execute as tarefas com precisão, adaptabilidade e eficiência [18].

Os sistemas de controlo podem apresentar um grau de complexidade consoante o objetivo do robô e do ambiente em que opera, podendo-se distinguir algumas formas de controlo base para a maioria dos sistemas, como o controlo em malha fechada e o controlo em malha aberta.

É importante salientar que sem uma arquitetura base os sistemas de controlo não conseguiriam funcionar com a mesma eficiência e eficácia. A arquitetura é a componente que organiza e define a estrutura geral de controlo, determinando o processo de informação, a tomada de decisões e a execução das ações [11]. As principais arquiteturas dos sistemas de controlo resumem-se à arquitetura reativa, à arquitetura deliberativa e à abordagem híbrida, e estas conseguem englobar a grande maioria dos sistemas robóticos em funcionamento.

Controlo em Malha Aberta

Os sistemas de controlo em malha aberta operam sem a necessidade de *feedback*. Nestes sistemas, os comandos de controlo são enviados para os atuadores, mas não existe uma verificação do estado do sistema, ou seja, desconhece-se se o resultado desejado foi alcançado. São mais fáceis de implementar, contudo, não têm a capacidade de corrigir erros ou ajustar a sua resposta [11].

Este tipo de controlo é normalmente utilizado em situações em que o ambiente é previsível e as ações a executar são relativamente simples, não sendo necessário conhecer o estado final do sistema [27].

Controlo em Malha Fechada

Por outro lado, os sistemas de malha fechada funcionam através da constante avaliação do desempenho do robô e ajuste das suas ações conforme necessário. Estes sistemas são caracterizados, como é visível na Figura 2.13, pela realimentação do *feedback* do estado do sistema à lógica de controlo para comparação com a entrada desejada e correção de quaisquer desvios do objetivo.

Este é um dos métodos de controlo mais implementados e é fundamental em aplicações que requerem precisão, em que todos os movimentos necessitam de ser exatos, sem grande margem para erros, como no fabrico de certos produtos ou na navegação de veículos autónomos.

Um exemplo comum de um controlo em malha fechada é o do controlador Proporcional, Integral e Derivativo (PID). As principais funções deste controlador são o cálculo contínuo do desvio da variável em causa e do valor desejado e, posteriormente, tentar corrigir este desvio [28]. É uma ferramenta essencial em aplicações de controlo industrial devido à sua simplicidade, robustez e eficácia.

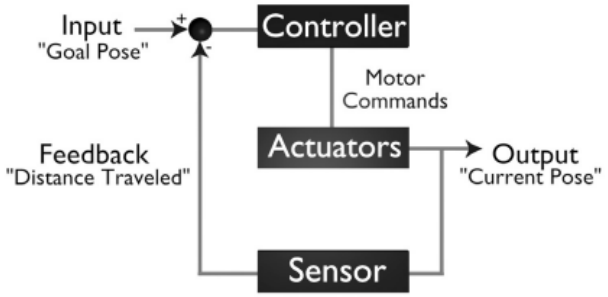


Figura 2.13: Modo de funcionamento de um controlo em malha fechada típico [11]

Este controlador assenta as correções na sua equação característica, representada em (2.2), e em três componentes distintos que lhe dão o nome: a componente Proporcional que altera a saída proporcionalmente ao erro atual; a componente Integral que considera a acumulação de erros passados ao longo do tempo, ou seja, altera a saída do sistema através da soma do erro temporal; e, por fim, a componente Derivativa que prevê o erro futuro com base na reta de declive do erro atual, e soma à saída do sistema uma correção para o minimizar [28].

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt} \tag{2.2}$$

- K_p – Ganho Proporcional;
- K_i – Ganho Integral;
- K_d – Ganho Derivativo.

Arquitetura de Controlo Reativo

A arquitetura de controlo reativo baseia o comportamento do sistema robótico na reação às entradas dos sensores. Assim sendo, nesta arquitetura as ações de um robô são uma resposta direta a uma alteração no estado do meio, não existindo necessidade de processamento ou planeamento complexos [29].

Sistemas que aplicam este controlo caracterizam-se pela sua simplicidade e rapidez, sendo adequados para ambientes dinâmicos e imprevisíveis. No entanto, estes sistemas podem não ter a capacidade de realizar tarefas complexas que exijam previsão ou tomada de decisões estratégicas [27].

Arquitetura de Controlo Deliberado

O controlo deliberado envolve um planeamento detalhado e a tomada de decisões com base num modelo do ambiente. Esta arquitetura baseia-se numa lógica bem

definida de três etapas: em primeiro receber as informações do meio; em segundo planear e avaliar as possíveis respostas; por fim, executar a ação [30].

Os sistemas deliberados são, por sua vez, mais lentos do que os sistemas reativos, uma vez que requerem recursos computacionais significativos para processar informações e gerar planos. Todavia, são excelentes em ambientes em que as tarefas são complexas e exigem uma análise cuidadosa dos estados futuros [27].

Arquitetura de Controlo Híbrido

Uma arquitetura de controlo híbrido combina os pontos fortes dos sistemas apresentados anteriormente, reativo e deliberado, de modo a poder reagir rapidamente em situações imprevisíveis, não descurando, porém, o planeamento futuro quando a situação o exige [11].

Esta abordagem permite o desenvolvimento de robôs mais versáteis, em que se consegue um equilíbrio entre a resposta imediata e o planeamento a longo prazo.

2.2.4 Atuadores

Os atuadores são componentes fundamentais de um robô. Na lógica de sentir, pensar, agir, os atuadores impulsionam o movimento e executam as tarefas. Ao converter uma forma de energia, como a elétrica, a pneumática ou a hidráulica, num movimento controlado, os atuadores permitem aos robôs manipular objetos, navegar pelo ambiente envolvente e interagir com o mesmo.

Atuadores Elétricos

Os motores elétricos são dos atuadores mais utilizados na robótica devido à sua versatilidade e eficiência. Existem diversos modelos de motores com diferentes tamanhos e capacidades, sendo que cada um apresenta vantagens e desvantagens consoante a sua especialidade. Alguns dos vários motores elétricos mais aplicados incluem os motores Corrente Contínua (DC) preferidos pelo seu controlo simples e fiabilidade, os servo-motores pelo controlo de velocidade e posição e os motores de passo pelo controlo e posicionamento preciso [18].

Os motores DC convertem energia elétrica em energia mecânica, utilizando ímanes e correntes para gerar campos magnéticos que provocam a rotação do veio do motor [31]. Assim, a potência de um motor DC é diretamente proporcional à tensão de alimentação, o que provoca um aumento do binário produzido, consoante o aumento da corrente consumida. A quantidade de potência gerada é proporcional ao produto do binário e da velocidade de rotação.

Quando o motor se encontra sem carga, a sua velocidade de rotação é elevada, mas o binário é nulo. Em oposição, quando o motor está parado, o seu binário é máximo, mas a velocidade de rotação é nula [11].

São estes dois extremos que caracterizam uma grande parte dos motores, uma vez que entre eles encontra-se a zona de trabalho útil.

Em situações em que é necessário o robô mover-se para uma posição específica, os motores DC não são os mais adequados, uma vez que sozinhos não possuem nenhum controlo de posição. Desta forma, apareceram os servo-motores que podem rodar o seu eixo para uma posição específica. Estes são um tipo específico de motores DC com a adição de uma caixa de redução, um sensor de posição e um circuito de controlo, como ilustrado na Figura 2.14 [18, 31].

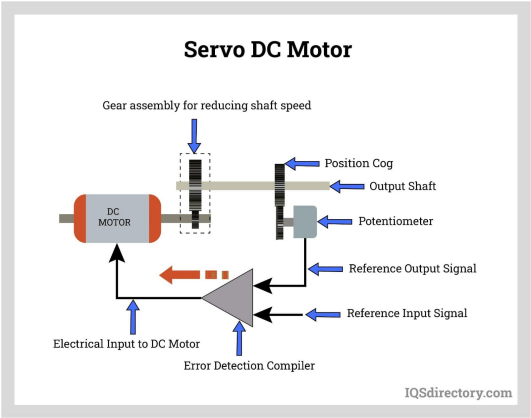


Figura 2.14: Componentes de um servo-motor [31]

Enquanto a caixa de redução permite a custo de velocidade obter mais binário, o circuito de controlo é responsável pelo controlo em malha fechada do motor. Assim, segundo o *feedback* do sensor da posição do eixo, o circuito gera um sinal para corrigir o desvio entre a ordem dada e a posição real [11].

Em relação aos motores de passo, a sua principal característica é o facto do seu eixo poder rodar por passos, ou seja, mover-se uma quantidade fixa de graus. Este facto acontece devido à estrutura interna do motor que permite energizar uma ou mais fases do estator, gerando um campo magnético devido à corrente que flui na bobina, o que causa a que o rotor se alinhe com este campo. Ao alimentar diferentes fases em sequência, o rotor pode rodar uma quantidade específica de passos para atingir a posição final desejada, como demonstrado na Figura 2.15 [32].

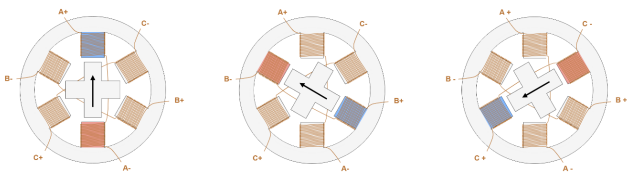


Figura 2.15: Sequência de passos num motor de passo [33]

Assim, um motor de passo está concebido de modo a dividir uma rotação completa num número de rotações parciais mais pequenas, ou seja, o motor de passo pode mover-se através de graus ou ângulos de rotação definidos, conhecendo assim a

posição angular exata do veio através da contagem do número de passos realizados, sem necessidade de um sensor externo de *feedback* [33].

Atuadores Pneumáticos

Os atuadores pneumáticos aproveitam ar comprimido para gerar movimento mecânico, sendo por isso conhecidos pela sua rapidez e segurança.

Estes atuadores apresentam uma conceção mecânica simples e um funcionamento flexível, o que possibilita a sua ampla aplicação em motores a combustível, em aplicações ferroviárias e na aviação [18, 34, 35, 36].

A sua grande vantagem, sobre os outros tipos de atuadores, resume-se à eficácia dos equipamentos, exigindo menos manutenção e longos ciclos de funcionamento.

Atuadores Hidráulicos

Os atuadores hidráulicos, por outro lado, utilizam a pressão de líquidos hidráulicos para criar movimento, fornecendo grandes quantidades de potência e força necessárias para aplicações industriais pesadas [18].

Estes atuadores são vantajosos porque conseguem manter a força e o binário constantes sem que a bomba precise de fornecer mais fluído ou aumentar a pressão. Na mesma linha, os atuadores hidráulicos podem ter as suas bombas e motores a uma distância considerável do equipamento e atuar com uma perda mínima de potência e, comparativamente com os atuadores pneumáticos, os atuadores hidráulicos geram forças superiores [37].

Quanto às suas desvantagens, é comum existir fuga de líquidos, o que provoca uma redução de eficiência e a necessidade de recorrer a várias peças complementares para o seu bom funcionamento.

2.3 Categorias de Sistemas Robóticos

Como já foi referido, a robótica é uma área muito extensa e por isso abrange uma vasta gama de robôs com objetivos distintos e focados em ambientes específicos. Esta gama diversificada pode variar entre simples braços mecânicos para linhas de montagem e sistemas complexos, orientados para a IA, capazes de aprender e de se adaptar a novos ambientes. A classificação dos robôs é essencial para compreender a diversidade deste domínio, uma vez que cada modelo é concebido com características únicas para enfrentar desafios específicos. Nem sempre é simples e fácil categorizar alguns modelos, contudo, estes podem ser agrupados segundo critérios baseados na sua função, conceção e ambientes em que operam [38].

Os robôs podem ser divididos em várias categorias nomeadamente robôs industriais, de serviço, veículos autónomos e robôs colaborativos [39].

2.3.1 Robôs Industriais

A indústria foi uma das primeiras áreas a introduzir a robótica para transformar os processos de fabrico. Desde esse momento, a área da robótica industrial cresceu e continuou a introduzir novas tecnologias para melhorar as suas capacidades e acompanhar as exigências de produção [40, 41].

Os modelos de robôs que se integram nesta categoria definem-se, maioritariamente, pela sua configuração. Esta é uma categoria focada na realização de tarefas automatizadas para a área da indústria, Figura 2.16, que podem incluir a montagem de produtos, a soldadura, o manuseamento de materiais e muito mais; aplicações que devido à sua natureza exigem precisão, rapidez e força [42].

Figura 2.16: Robô industrial [43]

Na categoria de robôs industriais incluem-se, por exemplo, robôs cartesianos, cilíndricos, *Selective Compliance Assembly Robot Arm* (SCARA) e articulados.

Robôs Cartesianos

Os robôs cartesianos, também conhecidos como robôs lineares, movimentam-se somente em linha reta segundo os três eixos ortogonais (X, Y e Z) [44].

Estes robôs são normalmente constituídos por uma estrutura suspensa para o movimento no plano horizontal e um braço robótico para os movimentos verticais, sendo que na extremidade do braço encontra-se uma ferramenta, dependendo da função do modelo [45], como se observa na Figura 2.17.

A vantagem dos robôs cartesianos, em relação a outros sistemas, é a sua simplicidade, não só na sua estrutura funcional, mas também na sua programação; devido a apenas movimentar-se linearmente, um único controlador pode gerir toda a lógica do robô [46].

São aplicados frequentemente em máquinas *Computer Numerical Control* (CNC), impressão 3D e tarefas que envolvam montagem linear ou operações de recolha e

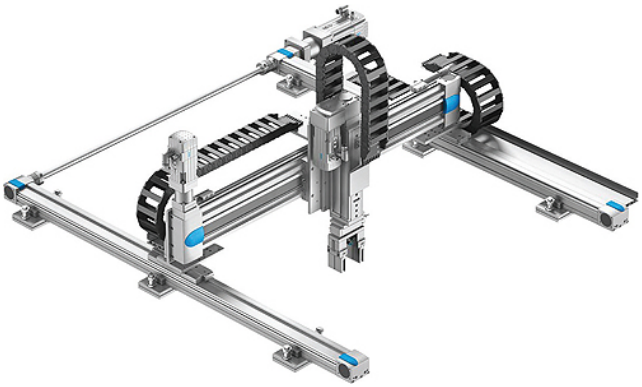


Figura 2.17: Exemplo de um robô cartesiano [45]

colocação, uma vez que podem executar estas tarefas com elevada precisão e repetibilidade [47].

Robôs Cilíndricos

Os robôs cilíndricos, ao contrário dos robôs cartesianos que apenas apresentam articulações lineares, possuem uma articulação rotativa geralmente na sua base, como se observa na Figura 2.18. Assim, com esta configuração o espaço de trabalho apresenta uma forma cilíndrica, o que proporciona mais flexibilidade [48].

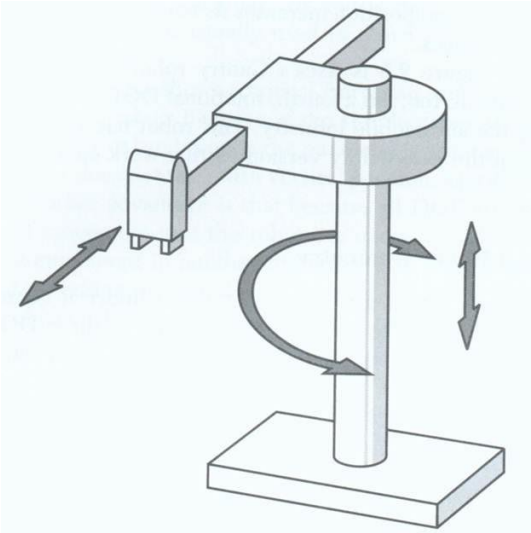


Figura 2.18: Graus de movimento de um robô cilíndrico [49]

A combinação de movimentos rotativos e lineares permite que os robôs cilíndricos executem uma gama mais vasta de tarefas, tais como operações de montagem, soldadura, manuseamento de materiais e embalagem de produtos. O espaço de trabalho cilíndrico é vantajoso e oferece mais versatilidade a este modelo [49].

Robôs SCARA

Os robôs SCARA são definidos por serem compatíveis com os eixos X e Y e rígidos no eixo Z. Estes possuem 4 articulações: uma para a rotação da base, duas para a rotação do braço segundo o eixo Z e, por último, uma articulação linear que se desloca apenas segundo o eixo Z [44]. A Figura 2.19 apresenta um exemplo de um robô SCARA.

Figura 2.19: Robô SCARA [50]

Graças à sua conceção simples, estes robôs podem deslocar-se rapidamente, mantendo sempre a precisão e a exatidão. Por conseguinte, são uma opção popular para pequenas aplicações de montagem, como se observa nas indústrias eletrónicas e de automóveis [51, 52].

Robôs Articulados

Os robôs articulados, ou comummente apelidados braços robóticos, são um dos tipos mais comuns de robôs industriais; são altamente versáteis e podem ser reconhecidos pelas suas articulações rotacionais que proporcionam um elevado grau de flexibilidade e destreza, tornando-os ideais para executar tarefas complexas [53]. Na Figura 2.20 pode-se observar um exemplo destes robôs.

Estes robôs são constituídos por vários segmentos, os chamados elos, que se encontram interligados através de articulações. Estas têm o propósito de permitir o movimento do robô numa nova direção e, desta forma, aumentar o número de parâmetros independentes que determinam a posição e a orientação do robô [44].

As articulações dos braços robóticos podem ser classificadas em dois tipos: rotativas e prismáticas. As primeiras permitem a rotação do robô ao longo de um eixo, enquanto as segundas são articulações lineares que permitem ao robô movimentar-se em linha reta [53].

Figura 2.20: Exemplo de um robô articulado [39]

Normalmente, estes robôs industriais apresentam seis DOF, à semelhança de um braço humano e das suas articulações básicas, o que lhes permite executar tarefas complexas e atingir posições arbitrárias, contornando obstáculos e podendo ser adaptados para operar em diversas situações. Estas características tornam estes robôs ideais para aplicações como a soldadura, o manuseamento de materiais e a montagem.

2.3.2 Robôs de Serviço

Os robôs de serviço pertencem a um segmento da robótica focado em prestar assistência ao ser humano, de modo a melhorar a sua qualidade de vida. A diferença entre esta categoria e a dos robôs industriais é o contexto em que estão inseridos. Enquanto os robôs industriais estão normalmente confinados a ambientes industriais e de fabrico, os robôs de serviço operam em ambientes mais dinâmicos e centrados no ser humano, desde serviço pessoal, em ambientes domésticos ou de lazer, até ao serviço profissional em ambientes comerciais, para executar tarefas particulares de apoio a empresas e organizações [54].

Os robôs de serviço pessoal são concebidos para auxiliar diretamente o ser humano nas tarefas da sua vida diária. O seu espaço de operação é, normalmente, os ambientes domésticos, onde assumem funções como a de limpeza, de prestação de cuidados e de companhia. Neste segmento incluem-se os aspiradores autónomos, que, como o nome indica, tratam autonomamente da limpeza dos espaços do lar; os robôs de assistência, presentes na Figura 2.21, concebidos para funcionarem com pessoas com deficiência/idosos em contexto de mobilidade reduzida ou em atividades diárias básicas e até de supervisão das condições de saúde [55, 56, 57].

Os robôs de serviço profissional focam-se em aumentar a eficiência operacional, melhorar a segurança e prestar serviços em ambientes comerciais, industriais ou públicos. No sector da saúde, estes robôs executam uma vasta gama de tarefas de apoio, desde cirurgias até cuidados e diagnósticos dos pacientes. No setor logístico,

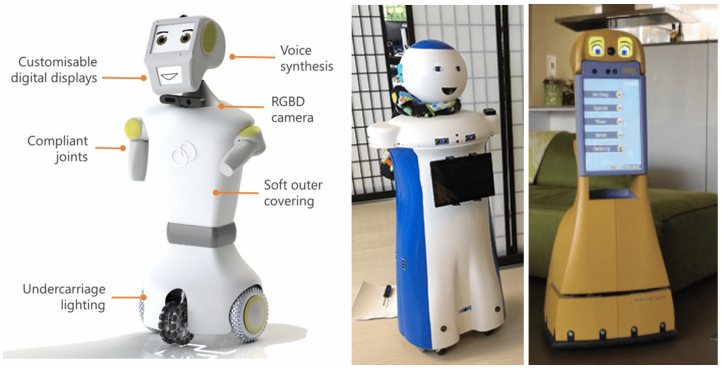


Figura 2.21: Robôs de serviço de cuidados a idosos [57]

estes transformam a entrega e transporte de mercadorias, automatizando os processos em armazéns ou em ambientes urbanos. No setor turístico, estes robôs podem interagir com o público, fornecendo informações, orientação e assistência em vários locais, como aeroportos, centros comerciais e hotéis [58], exemplificado na Figura 2.22 em que um robô de serviço armazena bagagens no hotel Chase Walker.

Figura 2.22: Braço robótico focado em armazenar bagagens [54]

2.3.3 Veículos Autónomos

Segundo a definição, um veículo autónomo é um veículo que, autonomamente, sem interferência humana, é capaz de entender o ambiente envolvente, tomar decisões e conduzir em segurança.

A Sociedade de Engenheiros Automóveis (SAE) enquadra a autonomia automóvel em 6 níveis distintos presentes na Figura 2.23: os três primeiros requerem um condutor para controlar os sistemas principais e conduzir, enquanto os três seguintes já apresentam uma autonomia crescente. No primeiro nível, o computador controla os sistemas básicos de condução até ao último nível em que não é necessária a presença ou controlo humano.

- **Nível 0**: Sem automatização - O condutor humano é responsável por todos os aspetos da condução;
- **Nível 1**: Assistência ao condutor - O veículo dispõe de sistemas que podem prestar assistência à direção ou à aceleração/travagem, mas o condutor mantém o controlo total;
- **Nível 2**: Automatização parcial - O veículo pode controlar a direção e a aceleração/travagem em determinadas condições, mas o condutor deve manter-se atento e monitorizar o ambiente;
- **Nível 3**: Automatização condicional - O veículo pode gerir a maioria dos aspetos da condução em determinados ambientes, mas o condutor deve estar preparado para assumir o controlo quando o sistema o solicitar;
- **Nível 4**: Automatização elevada - O veículo pode funcionar sem intervenção humana em ambientes ou condições específicas, como a condução urbana ou em autoestrada, mas o controlo humano ainda é possível;
- **Nível 5**: Automatização total - O veículo é totalmente autónomo e pode realizar todas as tarefas de condução em todas as condições sem qualquer intervenção humana necessária.

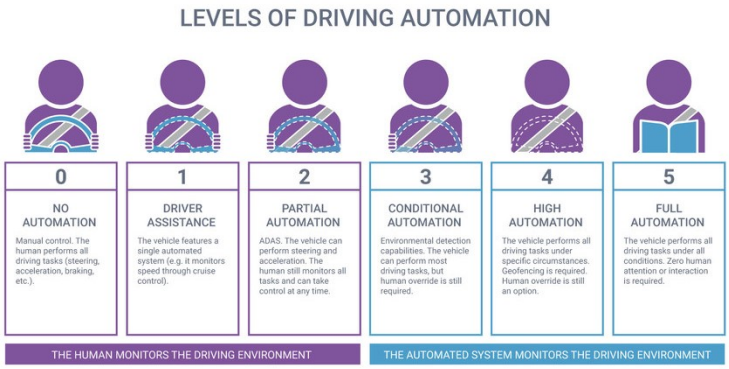


Figura 2.23: Níveis de condução autónoma [59]

Uma boa decisão costuma ser uma decisão bem informada "Decision-making is vital in self-driving cars" [60]. Por isso, antes de a tomar, o veículo deve utilizar os seus módulos, exemplificados na Figura 2.24, como sensores ultrassónicos, câmaras, radares, sistemas LiDAR e sistemas *Global Positioning System* (GPS) para fornecer informações visuais, detetar objetos e obstáculos e navegar de uma forma correta pelo meio envolvente. Estes veículos tendem a utilizar algoritmos de IA para interpretar os dados recolhidos pelos sensores, reconhecer padrões e tomar decisões de condução, sendo que os sistemas de controlo, com base na decisão tomada, gerem a aceleração, a travagem e a direção do veículo [61].

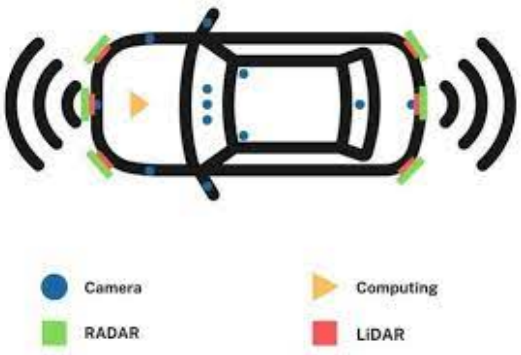


Figura 2.24: Sensores de veículos autónomos [62]

Esta é uma área em constante desenvolvimento, contudo, a imprevisibilidade de alguns acontecimentos inerentes à condução obsta ao avanço tecnológico. Governos e entidades reguladoras tentam estabelecer regulamentos para garantir a segurança e a fiabilidade destes veículos e integrar os veículos autónomos nos sistemas de transporte existentes.

2.3.4 Robôs Colaborativos

Cobots, ou robôs colaborativos, são robôs projetados para trabalhar ao lado de humanos, complementando as suas habilidades e capacidades, ao contrário dos robôs industriais tradicionais, que normalmente estão isolados dos trabalhadores por motivos de segurança. Isto permite que eles executem uma diversidade de tarefas, como a montagem, a embalagem e manutenção de máquinas, aumentando, no geral, a eficiência da linha de produção e diminuindo os custos de mão de obra [63].

Após décadas de evolução, com os robôs a tornarem as indústrias mais produtivas, ainda se registavam limitações à sua área de trabalho, assim como à sua constituição fechada. Por um lado, os trabalhadores não conseguiam alcançá-los facilmente para colaborar no manuseamento e/ou na sua reprogramação; por outro lado, os robôs não estavam à altura de processos mais flexíveis e evolutivos [64].

Numa tentativa de ultrapassar estes obstáculos, os investigadores Michael Peshkin e J. Edward Colgate, da *Northwestern University*, criaram um tipo de robô flexível, aliviando os trabalhadores das fábricas da *General Motor Manufacturing* de atividades extenuantes. Peshkin e Colgate projetaram um sistema de guincho inteligente que levantava cargas após detetar intenções de movimento. Consequentemente, registou-se uma diminuição do esforço muscular. Foi neste momento que surgiu o termo Robôs Colaborativos, *Cobots* [64].

Os *Cobots* estão equipados com sensores avançados, sistemas de controlo que lhes permitem trabalhar de forma segura na proximidade de pessoas, como demonstrado na Figura 2.25, e, por serem alternativas mais baratas e leves comparativamente aos

robôs articulados, tornaram-se muito populares nas indústrias automóvel, eletrónica, logística e oficinas, onde podem automatizar tarefas repetitivas ou perigosas, melhorando a ergonomia e a eficiência do local de trabalho [63].

Figura 2.25: Exemplo de um *cobot* comandado por gestos [65]

Capítulo 3

Inteligência Artificial

Nos dias de hoje, o panorama da tecnologia e da inovação é impulsionado pelos avanços sem paralelo da área de IA, tendo um impacto significativo na integração de sistemas computacionais nos diversos setores. Esta nova tecnologia encontra-se presente em quase todos os aspetos das nossas vidas, remodelando setores, transformando empresas e redefinindo a forma como se encaram as situações quotidianas.

O conceito de IA pode ser definido como a capacidade de um computador digital poder executar tarefas normalmente associadas à inteligência humana. São apelidados de sistemas artificialmente inteligentes, sistemas que possuem a capacidade de processos mentais, como o raciocínio, descoberta de significados, generalização e a aprendizagem com base em experiências passadas, podendo possuir capacidades para decifrar padrões, aprender com os dados e tomar decisões inteligentes sem programação explícita.

Também no domínio da robótica, a integração desta tecnologia tem permitido que os robôs evoluam de sistemas limitados a executar tarefas predefinidas para sistemas inteligentes capazes de aprender, capazes de se adaptar e de desempenhar ações em ambientes complexos e dinâmicos.

3.1 Evolução da Inteligência Artificial

Desde a criação do primeiro computador programável que a humanidade tem explorado a possibilidade de criar máquinas inteligentes e autónomas. Foi só nas décadas de 1940 e 1950 que se começou a equacionar verdadeiramente o processo de criação.

Foram vários os entusiastas como Alan Turing, John von Neumann e Norbert Wiener que tornaram possível os primeiros progressos neste conceito. Alan Turing, considerado o pai da IA, introduziu a ideia de uma máquina que poderia pensar e decidir de forma tão racional e inteligente como um ser humano, sendo indistinguível do mesmo, através do seu famoso "Teste de Turing" [66]. Paralelamente, Warren McCulloch e Walter Pitts apresentavam uma máquina a exemplo do cérebro humano que se tornou fundamental para a criação e desenvolvimento de redes neuronais artificiais [67].

Apesar das várias publicações sobre o tema, o conceito de IA só foi formalmente introduzido numa conferência na universidade de Dartmouth nos Estados Unidos, em 1956, por quatro investigadores John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon (Figura 3.1) [68]. Esta conferência para a discussão do potencial das máquinas inteligentes marcou o início do estudo e investigação das capacidades de IA.

Figura 3.1: Investigadores na conferência de Dartmouth [68]

A década seguinte ficou caracterizada pelos avanços na área com o desenvolvimento de novas técnicas e algoritmos, como sistemas que jogavam xadrez e resolviam problemas matemáticos, demonstrando o potencial da IA para resolver problemas complexos. Também foram criados programas de linguagem natural, como o sistema ELIZA, de Joseph Weizenbaum, que conseguia manter conversas realistas, levando os utilizadores a duvidar se estavam a comunicar com um ser humano, demonstrado na Figura 3.2 [69].

Contudo, durante os finais desta década, as inovações na área enfrentaram limitações significativas. A falta de capacidade computacional e de armazenamento de dados limitava o progresso, conduzindo a um período de redução do interesse pela investigação em IA, o chamado "Inverno da IA" [70].

Figura 3.2: Sistema ELIZA [69]

Nos anos 70 e 80, a investigação em IA ressurgiu e evoluiu com o aumento do poder de processamento dos computadores da época, surgindo, assim, sistemas capazes de reconhecimento de voz e imagem. Nesta altura desenvolveram-se programas que podiam diagnosticar doenças e sistemas de tomada de decisão baseados em regras, sistemas MYCIN, e, já nos anos 80, o Iterative Dichotomiser 3 (ID3) respetivamente [67]. Impulsionados pelo desenvolvimento de novas abordagens, como ML, vários investigadores concentraram-se em algoritmos capazes de aprender com dados, em vez de se basearem em regras predefinidas.

Nos finais da década de 80, as investigações sobre redes neuronais com múltiplas camadas forneceram uma base teórica para as redes neuronais profundas e a introdução de um novo conceito ao domínio, o conceito de DL. Porém, somente no início do século XXI, é que a DL apresentou avanços significativos devido à implementação de RNA, mais especificamente, as Redes Neuronais Profundas e à disponibilidade de grandes quantidades de dados [67].

O século XXI deu início a uma nova era da IA, sendo aplicada às mais diversas áreas, como a saúde, a área automóvel, a área do entretenimento, entre outras. A disponibilidade de grandes quantidades de dados, combinada com os avanços em *hardware*, como o aumento da velocidade das *Graphics Processing Unit* (GPU) e das redes sem fios, conduziram a avanços em várias aplicações, incluindo visão computacional, *Natural Language Processing* (NLP), veículos autónomos, *chatbots* e motores de recomendação a tornarem-se mais abundantes. Em 2004, a *Google* introduziu a sua primeira atualização de algoritmos de pesquisa que se baseou em ML e em 2011, o Watson da International Business Machines Corporation (IBM) derrotou dois campeões humanos no Jeopardy, que demonstraram o potencial da área. Também é de notar que a pandemia de COVID-19 acelerou o desenvolvimento de soluções de saúde alimentadas por IA [66, 70].

3.2 Conceitos Fundamentais de Inteligência Artificial

A IA é um campo alargado em rápida evolução que engloba vários conceitos, teorias e técnicas com o objetivo de tornar máquinas simples em poderosos sistemas inteligentes, capazes de realizar tarefas dinâmicas e complexas. Sendo assim, compreender estes conceitos base é essencial para perceber como funciona e como pode ser aplicada em diversos domínios, permitindo a existência de sistemas inteligentes.

Alguns destes conceitos fundamentais exemplificados na Figura 3.3 incluem: ML, um subconjunto da IA, que envolve algoritmos que permitem às máquinas aprender com os dados sem serem explicitamente programadas; redes neuronais que inspiradas no cérebro humano são fundamentais para o reconhecimento de padrões em conjuntos de dados; DL, um subconjunto de ML que aplica redes neuronais com múltiplas camadas para extrair padrões complexos dos dados; NLP que permite compreender e gerar linguagem humana; e visão computacional que permite às máquinas interpretar, detetar e tomar decisões com base em dados visuais.

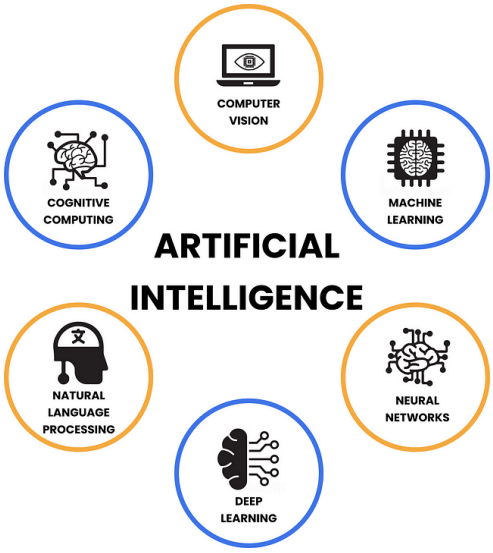


Figura 3.3: Algumas áreas de IA [71]

Estas são algumas das ferramentas que alimentam as soluções baseadas em IA aplicadas no mundo real, moldando o futuro das indústrias, desde os cuidados de saúde e finanças ao entretenimento e transportes.

3.2.1 *Machine Learning*

ML é um pequeno campo incluído na área de IA, definido como a capacidade das máquinas imitarem o pensamento humano, operando de uma forma autónoma, ou seja, sem uma programação explícita. Assim, uma aplicação de ML tem que aprender

com os dados fornecidos e procurar melhorar o seu desempenho ao longo do tempo de uma forma independente [72].

O seu modo de funcionamento é particularmente simples, sendo composto por três fases principais: a fase de treino, a fase de validação e a fase de teste, como se pode observar na Figura 3.4 [73].

A primeira fase caracteriza-se pela utilização de um conjunto de dados com o objetivo de ensinar um modelo a identificar características, padrões ou relações que implicam um resultado final. Desta forma, o modelo, durante o seu treino, atribui um novo peso a cada parâmetro interno de modo a conseguir futuramente replicar o seu treino numa situação definitiva.

Posteriormente, na segunda fase, este modelo é colocado perante um novo conjunto de dados, sendo avaliada a sua capacidade de reconhecer padrões e de os classificar ou avaliada a precisão das suas previsões, tendo em conta o seu treino. Nesta fase, consoante o desempenho do modelo, ajustam-se os valores dos hiperparâmetros, que regem o processo de formação do modelo, não sendo aprendidos a partir dos conjuntos de dados.

Assim, a fase de validação tem a função de controlar o desempenho do modelo, procurando evitar o fenómeno de *overfitting*, um cenário em que o modelo aprendeu características demasiado específicas dos dados de treino, captando o ruído em vez dos padrões subjacentes.

Numa última fase, verificam-se os resultados obtidos perante novos dados, a fim de se obter uma avaliação clara e imparcial do desempenho do modelo, e decide-se se o modelo desenvolvido está devidamente treinado ou se necessita de um novo treino antes de ser implementado.

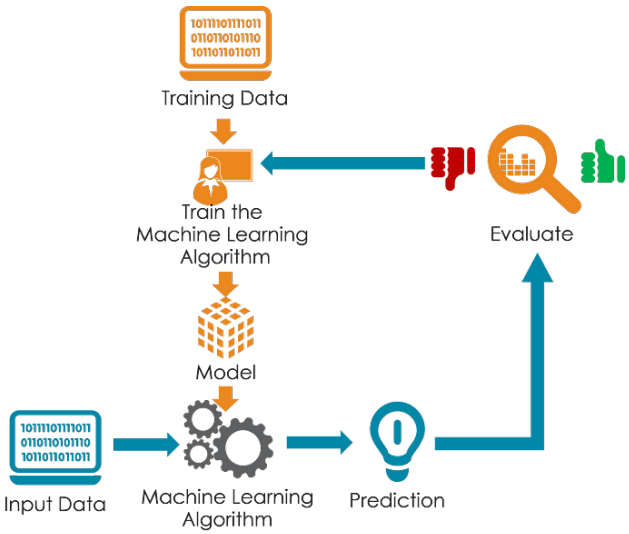


Figura 3.4: Modo de funcionamento de ML [73]

Tendo em conta o seu modo de funcionamento, é possível definir várias técnicas

de aprendizagem, sendo que cada uma tem as suas particularidades e aplicações no mundo atual, como se mostra na Figura 3.5. Estas técnicas incluem: *Supervised Learning*, *Unsupervised Learning*, *Semi-Supervised Learning* e, por último, *Reinforcement Learning* (RL) [74].

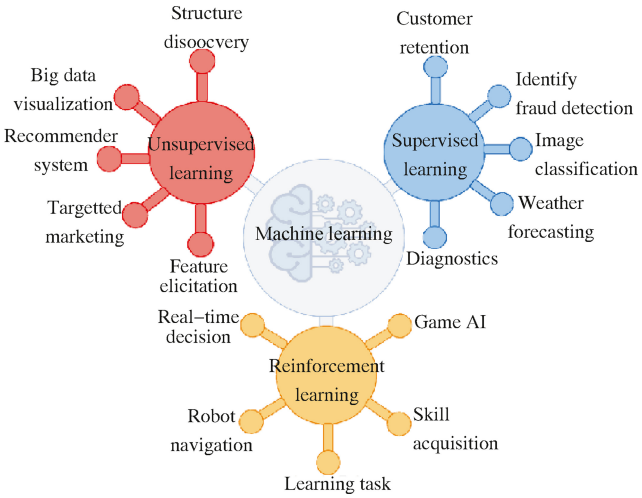


Figura 3.5: Aplicações das técnicas de ML [74]

Supervised Learning

De acordo com a técnica de *Supervised Learning*, o conjunto de dados que alimenta o modelo encontra-se completamente rotulado, ou seja, para cada dado de entrada é previamente conhecida a saída desejada. Posto isto, o modelo aprende a construir as relações que lhe permite atribuir as entradas às respetivas saídas [75] (Figura 3.6).

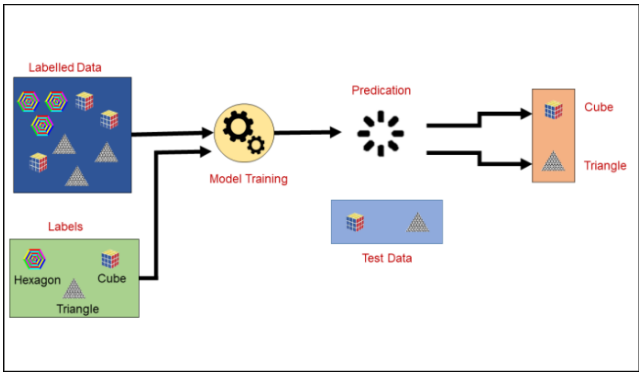


Figura 3.6: Modo de funcionamento de *Supervised Learning* [76]

Assim, esta técnica é normalmente aplicada em dois contextos distintos: problemas de classificação e problemas de regressão.

Nos problemas de classificação, os resultados finais do modelo podem ser classificados em categorias, ou seja, para uma dada entrada, a saída apresenta uma gama de valores finitos. Desta forma, a precisão do modelo é determinada tendo em conta

as entradas que foram corretamente atribuídas à sua categoria e as tentativas totais [77].

Por outro lado, nos problemas de regressão, a saída costuma ser um valor real, uma saída contínua e com valores infinitos. Por esta razão, a validação deste modelo é feita verificando o erro que existiu entre o valor previsto e o valor desejado. Este modelo é amplamente utilizado em previsões monetárias como valores estimados de vendas, entre outras aplicações [78].

Unsupervised Learning

Relativamente à *unsupervised learning*, os dados fornecidos ao modelo não apresentam nenhuma legenda ou classificação prévia. Assim, não existem resultados definidos ou conhecidos. O algoritmo deve então encontrar padrões ou estruturas nos dados que permitam classificar as entradas, por conta própria, sem qualquer orientação de um ser humano, como ilustrado na Figura 3.7 [72].

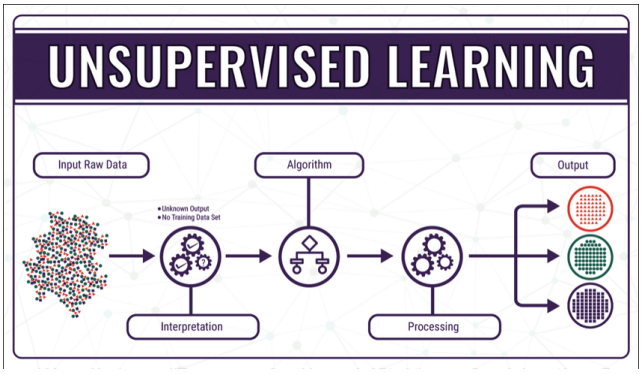


Figura 3.7: Modo de funcionamento de *Unsupervised Learning* [79]

Os algoritmos resultantes desta técnica normalmente podem dividir-se em:

- **Agregação** (*Clustering*) – O modelo, após analisar os dados iniciais, procura agrupar as entradas tendo em conta semelhanças e padrões comuns. Assim, cada grupo ou *Cluster* é caracterizado por uma propriedade diferenciadora das outras [77];
- **Associação** (*Association*) – Estes algoritmos analisam as entradas e estabelecem relações que englobem grandes quantidades de dados, como eventos que tendam a ocorrer em sequência [78].

Semi-Supervised Learning

Ao contrário das duas técnicas anteriores, em que os dados se encontravam todos classificados ou sem nenhuma classificação, este método apresenta um meio termo.

Uma vez que classificar todos os dados é um processo dispendioso e demorado e a não classificação de nenhum tem tendência a não obter os resultados mais precisos

em certos problemas, o método de *Semi-Supervised Learning* é a resposta. Este método consiste em alimentar o modelo com ambos os tipos de dados [80] (Figura 3.8).

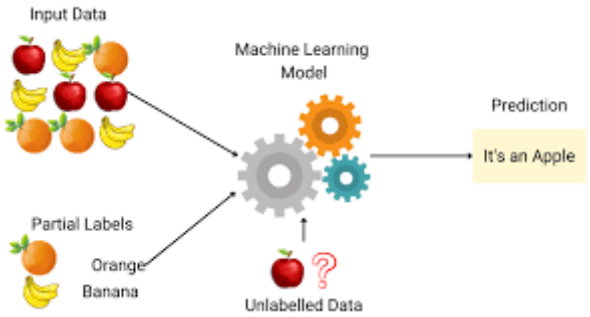


Figura 3.8: Modo de funcionamento de *Semi-Supervised Learning* [81]

Reinforcement Learning

Nesta técnica, um agente aprende a tomar decisões através da interação com o ambiente. Assim, a cada ação que o agente possa executar é atribuída um valor, o modelo, depois, opta por escolher a ação que resulta na obtenção de um valor mais alto, como se pode observar na Figura 3.9 [75].

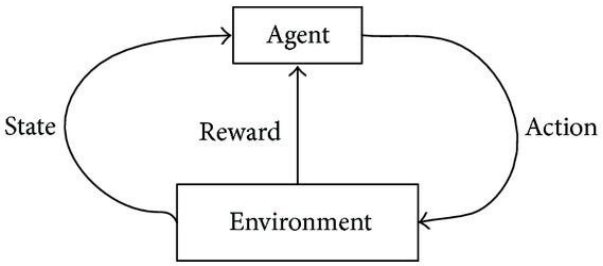


Figura 3.9: Método de aprendizagem de RL [82]

Este método consiste em recompensar o modelo por cada ação acertada ou penalizar por cada "má" escolha. Desta forma, o agente tem como objetivo maximizar a sua recompensa acumulada ao longo do tempo [73].

Este método é amplamente implementado em aplicações como a robótica, condução autónoma, jogos, entre outros.

Um exemplo de um algoritmo particularmente conhecido na área dos jogos é o AlphaZero. Este algoritmo foi criado com o intuito de dominar os jogos de tabuleiro mais jogados. Primeiramente, com o nome AlphaGo foi implementado no jogo Go, um jogo chinês inventado há mais de 2500 anos com o objetivo de conquistar o maior espaço possível e com mais de 10^{170} possíveis jogadas. Após aprender as regras, rever jogos profissionais e treinar, o algoritmo foi capaz de em 2015, pela primeira

vez na história dos algoritmos, derrotar o campeão mundial, no jogo presente na Figura 3.10. Atualmente, encontra-se disseminado em vários jogos de tabuleiro, salientando-se o Go e o Xadrez [83].

Figura 3.10: AlphaGo enfrenta o campeão mundial de Go [84]

3.2.2 Redes Neuronais

As redes neuronais são um subconjunto de ML inspirado na estrutura e no funcionamento do cérebro humano. Da mesma forma que os neurónios se interligam para criar sinapses, os nós de uma rede ligam-se e enviam sinais para serem processados por camadas de nós adjacentes. A cada ligação entre nós é atribuído um peso inicial e, durante o treino, a força desses pesos é ajustada numa tentativa de minimizar os erros nas previsões da rede [85].

A estrutura simples de uma rede consiste em três tipos de camadas, presentes na Figura 3.11: a camada de entrada, a camada oculta de processamento e por fim a camada de saída [86].

A camada de entrada é responsável por receber os dados em bruto e cada nó desta vai representar uma caraterística desses mesmos dados.

A camada oculta processa as entradas, procurando aprender as relações entre as características. É nestas camadas de processamento que as alterações e ajustes dos pesos de cada nó acontecem de um modo progressivo. Se esta etapa de processamento contiver mais do que uma camada de nós, a rede neuronal diz-se profunda [85].

Por fim, a camada de saída é o culminar do trabalho da rede, produzindo um resultado final, que pode ser uma classificação, previsão ou outro resultado, dependendo da tarefa.

Apesar da estrutura base ser simples, a arquitetura de algumas destas redes pode variar muito, desde redes simples em que os sinais se deslocam da entrada para a saída, até estruturas mais complexas, como as *Recurrent Neural Networks* (RNNs) e as *Convolution Neural Networks* (CNNs), concebidas para tipos de dados específicos.

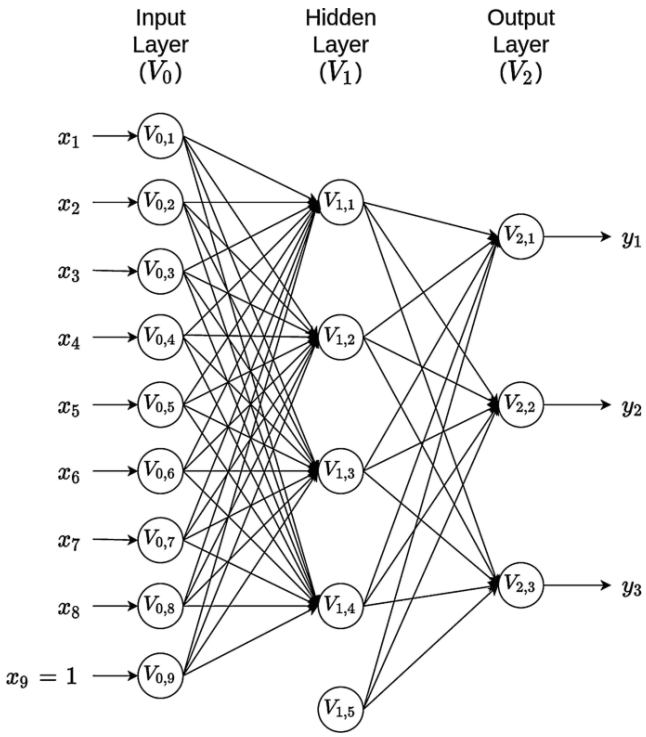


Figura 3.11: Estrutura base de uma rede neuronal [85]

Recurrent Neural Networks

Uma RNN é um tipo de rede neuronal particularmente adequado para dados sequenciais. Estas redes caracterizam-se por poderem ser cíclicas, ou seja, cada processo aplicado a uma entrada pode influenciar as entradas seguintes. Assim, estas redes apresentam memória, permitindo considerar entradas anteriores ao processar a entrada atual [87] (Figura 3.12).

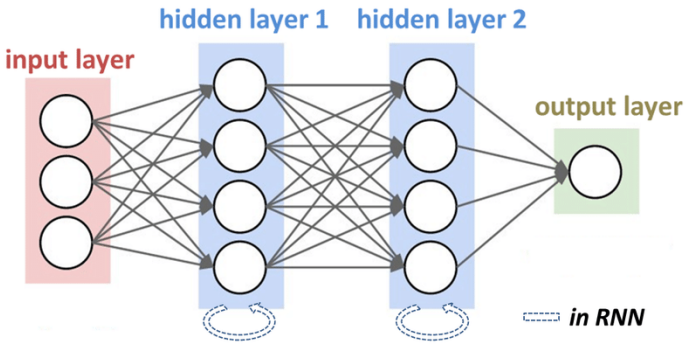


Figura 3.12: Princípio base de uma RNN [88]

Por estas razões, este tipo de redes permite fazer previsão de séries temporais, como valor de ações da banca e de vendas, processamento de linguagem e reconhecimento de voz, tarefas em que a ordem das entradas é significativa [89].

Convolution Neural Networks

Em contraste com a rede anterior, a CNN considera cada entrada como independente. Assim, cada ligação e processamento apenas acontecerá num sentido - da entrada até à saída. Estas redes são utilizadas principalmente para processar estruturas de dados em forma de grelha, como na identificação de padrões em imagens e visão computacional [90, 91].

Uma rede deste tipo é constituída por várias camadas, como camadas convolucionais, camadas de *pooling* e camadas totalmente ligadas. Com cada camada, a CNN aumenta a complexidade das características, ou seja, enquanto as primeiras camadas se concentram em elementos simples, como cores e extremidades; as camadas finais procuram reconhecer elementos ou formas maiores do objeto até finalmente obter o resultado pretendido, como se pode observar na Figura 3.13 [92, 93].

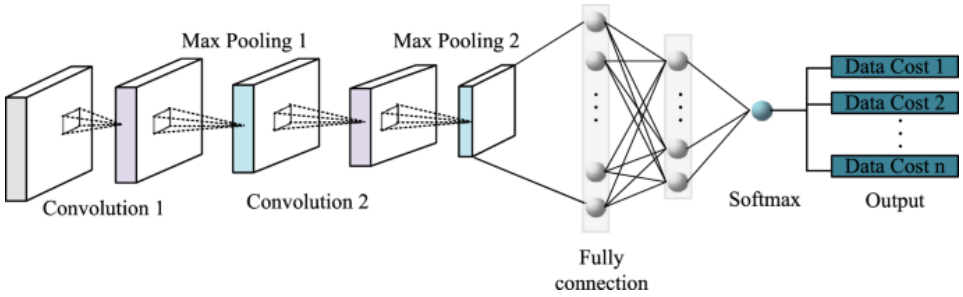


Figura 3.13: Estrutura de uma CNN [94]

As primeiras camadas, as camadas convolucionais, aplicam filtros aos dados de entrada para produzir mapas de características que representem padrões como arestas ou texturas [91].

De seguida, os dados passam pelas camadas de *pooling* que reduzem as dimensões dos mapas de características, preservando as informações mais relevantes. Estas camadas procuram reduzir a carga computacional e a complexidade da rede, no entanto, apesar de melhorar a eficiência, pode ocorrer perda de informação [91].

Por último, as camadas totalmente ligadas transformam as características extraídas pelas camadas anteriores numa saída que representa a probabilidade de cada classe [94].

3.2.3 *Deep Learning*

A DL é uma área do domínio de ML que começou a ser conhecida nos finais da década de 80. Contudo, apenas no início do século XXI, devido à implementação de redes neuronais com múltiplas camadas ocultas, mais especificamente as *Deep Neural Networks* (DNNs), é que começou realmente a ser considerada vantajosa [95].

Aproveitando o conceito das DNNs, a DL consegue captar padrões e representações mais complexos, executando tarefas que podem ser difíceis para modelos menos

profundos, ou seja, com menos camadas de processamento. Assim, aplicando uma hierarquia de camadas, em que as primeiras apresentam as características mais gerais e fáceis de identificar, enquanto as últimas apresentam as mais específicas dos dados iniciais, como se observa na Figura 3.14, modelos de DL são excelentes na extração automática de características de dados brutos [90].

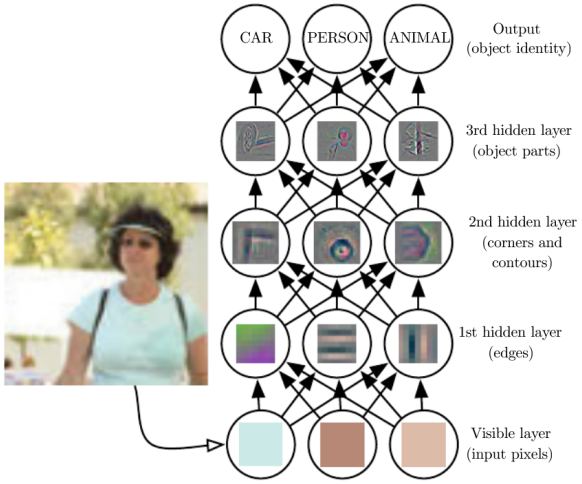


Figura 3.14: Processo de um modelo de DL [90]

O aumento da velocidade das GPU, a disponibilidade de grandes conjuntos de dados prontos a serem implementados e, mais recentemente, a introdução de processadores com unidades dedicadas à IA, têm levado esta área mais além, garantindo o seu crescimento exponencial [96].

Como se pode observar na Figura 3.15, da mesma forma que se verifica o aumento da quantidade de dados providenciados no treino de um algoritmo, também o desempenho manifesta esse aumento. No entanto, de todos os algoritmos existentes, os modelos de DL são os únicos que não apresentam estagnação a partir de uma dada quantidade de entradas. Pelo contrário, o desempenho destes modelos cresce continuamente, aperfeiçoando "arestas" até atingir resultados quase perfeitos.

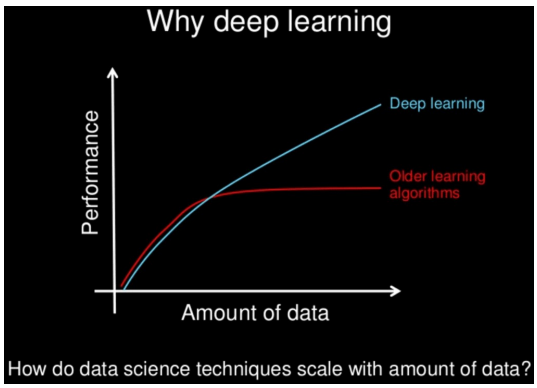


Figura 3.15: Relação entre dados e *performance* em ML [96]

Posto isto, tem sido aplicada com sucesso numa vasta gama de domínios desde visão computacional, em tarefas de deteção de objetos, em problemas de segmentação de imagens, em reconhecimento facial, até processamento de linguagem natural, tradução automática e análise de sentimentos.

3.2.4 *Natural Language Processing*

NLP é um ramo da IA que se concentra em permitir que os computadores entendam, interpretem, repliquem e manipulem a linguagem humana [97]. Através do uso de ML, NLP visa preencher a lacuna entre a comunicação humana e a compreensão das máquinas.

O ramo de NLP pode ser dividido em dois domínios: *Natural Language Understanding* (NLU), que se dedica à análise semântica e contextual do texto, e *Natural Language Generation* (NLG), que se concentra na criação de texto por uma máquina [98].

Em comparação, NLU é o domínio que apresenta mais dificuldades em ser efetivamente implementado, uma vez que existem diversos fatores a considerar na compreensão do sentido de uma frase, a linguagem humana pode ser muito subjetiva. Como exemplo: uma simples frase pode ser analisada de diversas formas e apresentar um significado distinto para cada análise ou uma mesma palavra pode ser utilizada em duas frases distintas e ter duas definições díspares [99].

O processo de um modelo de NLP é constituído por várias etapas até que seja possível interpretar a linguagem humana, contudo, as primeiras são relativamente simples [98, 100]:

- ***Tokenization***: dividir o texto em pequenos fragmentos chamados *tokens*;
- ***Part-of-speech tagging***: marcar as palavras segundo as suas classes gramaticais, (substantivos, verbos, adjetivos, advérbios, pronomes);
- ***Stemming and lemmatization***: transformar as palavras, reduzindo-as às suas formas de raiz;
- ***Stop word removal***: filtrar as palavras comuns que adicionam pouca ou nenhuma informação (preposições, artigos, etc.).

Posteriormente, as ferramentas de NLP podem transformar o texto em algo que uma máquina consiga entender. No fim do processo, o modelo desenvolvido consegue realizar um simples diálogo coerente [98].

Devido à difusão desta tecnologia, as aplicações da mesma encontram-se presentes em vários sistemas designadamente sistemas GPS operados por voz, até *softwares* de deteção de *spam*, passando por *chatbots* de atendimento ao cliente, como se pode visualizar na Figura 3.16 [101].

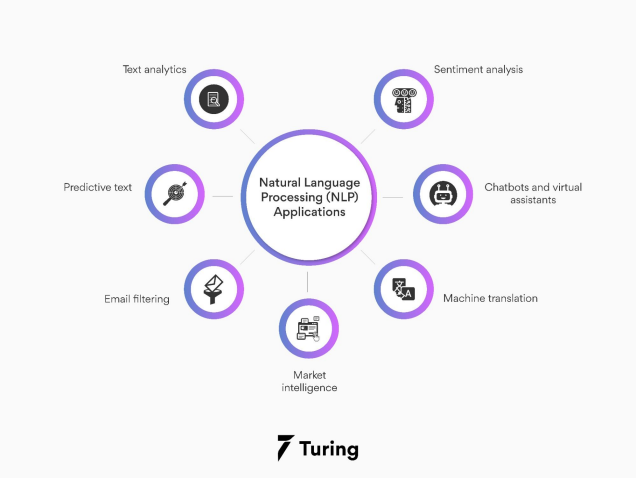


Figura 3.16: Aplicações de NLP [101]

3.2.5 Visão Computacional

A visão computacional, um ramo da IA, é definida como um campo dedicado à análise de imagens e vídeos de forma a permitir que computadores e máquinas reconheçam e processem o seu conteúdo [102].

Um algoritmo deste campo executa, normalmente, três passos: o processamento das entradas, a extração e análise das características e, por fim, a compreensão do conteúdo das imagens e realização da tarefa [103].

Previamente a poder compreender os dados visuais, estes precisam de ser pré-processados, numa tentativa de melhorar as suas características, corrigir distorções ou preparar os dados. As técnicas comuns de pré-processamento incluem filtragem, deteção de margens e transformações do espaço de cor, manipulando os valores dos pixeis da entrada [104].

De seguida, o sistema de visão identifica e extrai as características relevantes dos dados, como arestas, texturas, formas ou objetos.

A última etapa depende da tarefa proposta. Esta pode ser uma tarefa de deteção e reconhecimento de objetos, em que o modelo procura identificar e localizar classes de objetos específicos; uma tarefa de classificação, em que o modelo tenta atribuir corretamente uma etiqueta a uma imagem; ou uma tarefa de segmentação, em que o algoritmo identifica a região do objeto pretendido; como se demonstra na Figura 3.17 [105].

Com o desenvolvimento das redes CNN e da área de DL, o ramo da visão computacional tem crescido, estando cada vez mais presente na vida quotidiana em amplas aplicações como [103]:

- **Veículos autónomos** – onde são aplicados para navegar e tomar decisões em tempo real com base no ambiente visual;

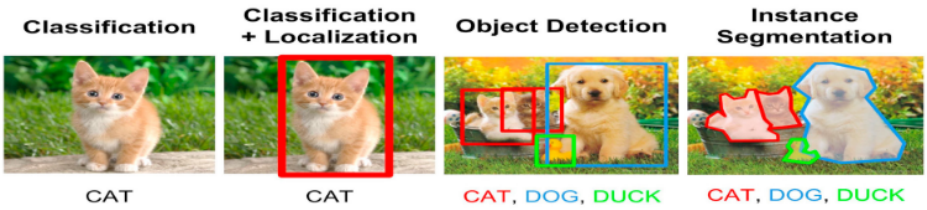


Figura 3.17: Tarefas de visão computacional [105]

- **Diagnósticos médicos** – em que ajudam a identificar possíveis anomalias em raios-X, ressonâncias magnéticas e outros exames;
- **Segurança** – em aplicações de reconhecimento facial e de videovigilância.

3.3 Algoritmos de Inteligência Artificial

A área da IA, como se demonstrou, engloba diversos domínios com métodos diferentes de aprendizagem e gestão de conhecimentos. Os algoritmos são a base destas técnicas, permitindo o desenvolvimento e o ajuste dos sistemas inteligentes para a realização, com sucesso, de tarefas complexas, desde o reconhecimento de padrões em conjuntos de dados até à tomada de decisões em tempo real.

3.3.1 *Linear Regression*

O algoritmo de *Linear Regression* pertence aos algoritmos de *supervised learning* que procura relacionar uma variável em estudo com uma ou mais variáveis independentes, as chamadas características. O principal objetivo deste algoritmo é encontrar uma equação linear ajustada a estas variáveis que consiga prever o resultado esperado, como se ilustra na Figura 3.18 [106, 107, 108].

Assim, pode-se escrever a equação que sustenta este algoritmo como o somatório das características multiplicadas pelo seu peso [110]:

$$y = \beta_0 + \beta_1 \cdot X_1 + \beta_2 \cdot X_2 + ...\beta_n \cdot X_n + e \tag{3.1}$$

- y – representa a variável dependente;
- $X_1, X_2, ..., X_n$ – representam as variáveis independentes;
- β_0 – representa a interseção com a origem;
- $\beta_1, \beta_2, ..., \beta_n$ – representam os coeficientes/pesos;
- e – representa o desvio dos dados observados em relação à previsão.

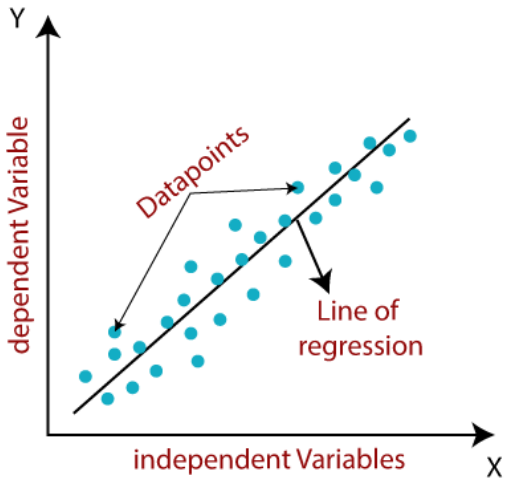


Figura 3.18: *Linear Regression* [109]

Este algoritmo fundamental e simples é a base de vários outros algoritmos mais complexos, estando amplamente aplicado em tarefas que envolvem a previsão de resultados contínuos, como, por exemplo, na área financeira em previsões de mercado, volumes de vendas, entre outros exemplos [111].

3.3.2 *Logistic Regression*

Logistic Regression é um algoritmo focado em problemas de classificação binária, derivado dos algoritmos de *Linear Regression*. Por oposição, *Logistic Regression* é concebida para prever a probabilidade de uma entrada pertencer a uma classe, em detrimento de prever valores contínuos [107].

Assim, transformando o valor de saída da função de regressão linear através de uma função logística para estimar probabilidades, também referida como a função sigmoide, o algoritmo de *Logistic Regression* modela a relação entre as características e o resultado binário final. A função sigmoide é descrita pela equação em que z corresponde ao valor de saída da equação de regressão linear [110]:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.2}$$

- $\sigma(z)$ tende para 1 para z $\rightarrow \infty$;
- $\sigma(z)$ tende para 0 para z $\rightarrow -\infty$;

Esta função sigmoide é responsável por mapear os valores de probabilidade para o intervalo de valores reais entre 0 e 1. Desta forma, caso o resultado da função para uma entrada seja superior a 0.5, então esta entrada pertencerá à Classe 1. Pelo contrário, para valores inferiores, a entrada pertence à Classe 0 [112], como se apresenta na Figura 3.19.

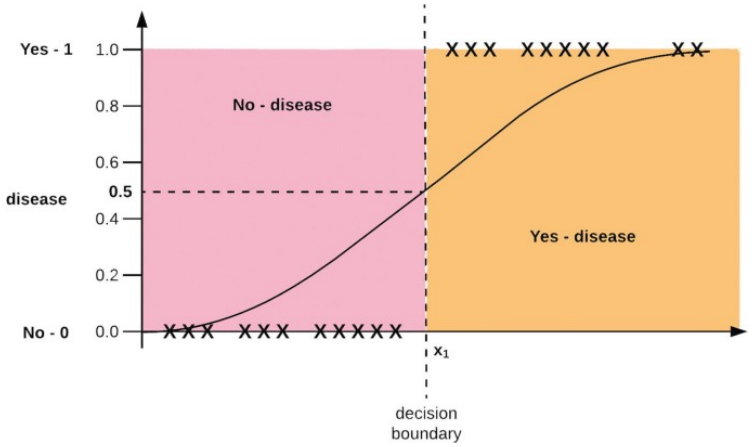


Figura 3.19: Exemplo de *Logistic Regression* [113]

3.3.3 *Decision Tree*

Decision Tree é um dos algoritmos mais poderosos e populares, podendo ser utilizado para tarefas de classificação e de regressão.

O princípio base deste algoritmo é a construção de uma estrutura em árvore que representa um conjunto de decisões e as suas possíveis consequências. Cada nó da árvore representa uma decisão baseada numa caraterística e cada ramo representa um resultado possível. As folhas da árvore representam as decisões ou previsões finais [106, 107, 110, 114].

Neste algoritmo, a árvore é construída progressivamente a partir da sucessiva divisão do conjunto de dados em subconjuntos, de modo a criar grupos tão distintos quanto possível, como se pode observar na Figura 3.20. A divisão assenta na seleção do melhor atributo/característica com base em determinados critérios, como o ganho de informação ou a impureza de Gini. Este processo ocorre até ser cumprido um critério de paragem, como atingir uma profundidade máxima ou a falta de um número mínimo de instâncias num nó/folha [86].

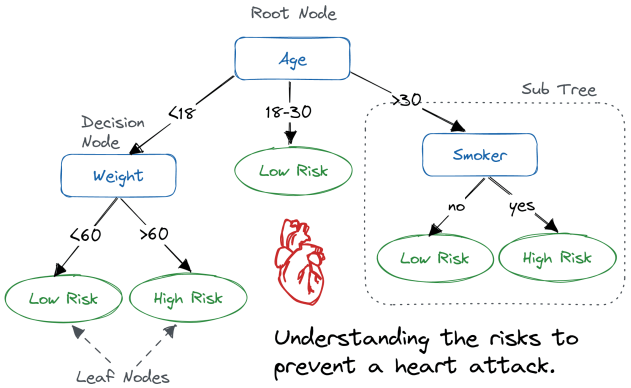


Figura 3.20: Exemplo de um algoritmo de *Decision Tree* [115]

Uma das maiores vantagens deste algoritmo é a sua fácil interpretação, permitindo uma análise célere das várias decisões e compreensão do resultado final. No entanto, este algoritmo tende a sofrer de *overfitting*, especialmente quando as árvores vão aumentando a sua profundidade e complexidade. Para além disso, pequenas alterações ou ruídos que sobressaiam do conjunto de dados resultam em estruturas potencialmente diferentes [114].

3.3.4 *Random Forest*

Random Forest é um algoritmo de *ensemble learning* que combina vários algoritmos de *Decision Tree* para produzir um modelo mais robusto e preciso.

A ideia base deste algoritmo é criar uma série de árvores de decisão, construídas a partir de subconjuntos aleatórios do conjunto de dados, permitindo avaliar diversos conjuntos de características e a sua relevância no treino do modelo. A decisão final recai sobre a maioria dos resultados de cada árvore para tarefas de classificação ou a média dos valores para tarefas de regressão, como se pode visualizar na Figura 3.21 [116].

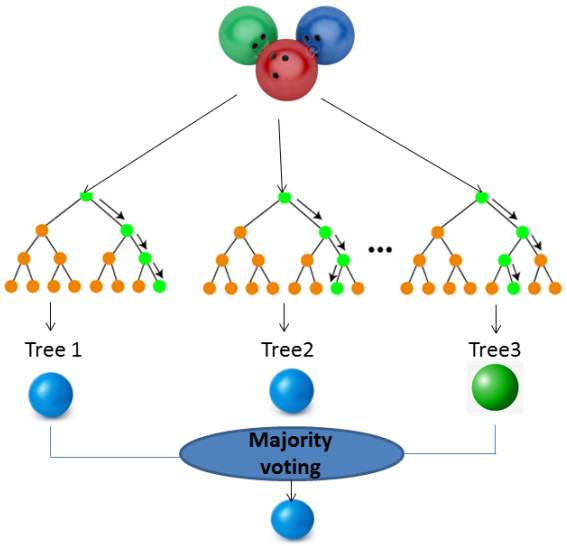


Figura 3.21: Exemplo de um algoritmo de *Random Forest* [117]

A aleatoriedade na escolha das características a ser testadas e a junção de vários algoritmos de árvore reduzem o risco de *overfitting*, melhorando o desempenho global da previsão, ao contrário do que o que acontece com o algoritmo simples de *Decision Tree* [107].

3.3.5 *K-Nearest Neighbor*

O algoritmo *K-Nearest Neighbor* (KNN) é um algoritmo simples que procura atribuir um novo dado a um grupo, tendo em conta a distância para os seus vizinhos mais

próximos, uma vez que parte do pressuposto de que classes idênticas apresentam propriedades semelhantes [107, 118].

Em primeiro lugar, este algoritmo começa o seu processo por guardar os dados de treino, em vez de efetivamente aprender com estes, um fenómeno de *Lazy Learning*.

Em segundo lugar, o algoritmo calcula a distância do novo elemento a todos os dados guardados, através de métricas de distância como a Euclidiana, a Manhattan ou Minkowski [119].

- Distância Euclidiana:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - yi)^2} \tag{3.3}$$

  - x_i – Coordenada do primeiro ponto;
  - y_i – Coordenada do segundo ponto;
  - n – Dimensão do espaço.

- Distância Manhattan:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.4}$$

  - x_i – Coordenada do primeiro ponto;
  - y_i – Coordenada do segundo ponto;
  - n – Dimensão do espaço.

- Distância Minkowski:

$$d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{3.5}$$

  - x_i – Coordenada do primeiro ponto;
  - y_i – Coordenada do segundo ponto;
  - n – Dimensão do espaço;
  - p – Ordem de distância.

Por fim, atribui-se uma classe a este novo elemento, consoante a sua distância a "k" vizinhos próximos, como se demonstra na Figura 3.22, em que o ponto em questão é atribuído à classe B devido a $k = 3$ vizinhos mais próximos.

É um algoritmo de rápida implementação, devido a não precisar de uma fase de treino. Contudo, devido ao *Lazy Learning*, requer mais recursos de memória e de armazenamento do que a maior parte dos modelos quando deparados com *datasets* extensos, uma vez que guarda em memória os dados de treino [121].

Na mesma medida, por calcular as distâncias entre o novo ponto de dados e todos os exemplos de treino, este algoritmo requer imensos recursos computacionais [122].

Para além disto, não distingue as características a nível de relevância, o que leva a um aumento do número de resultados incorretos [122].

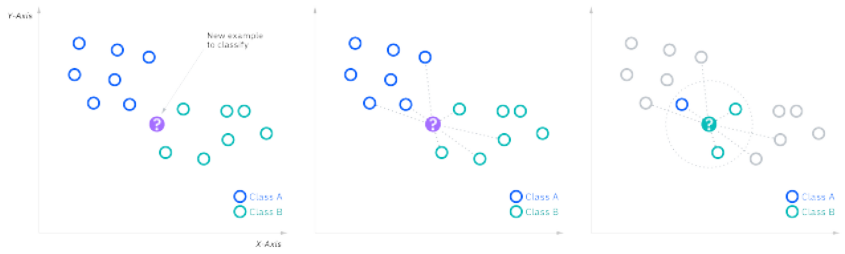


Figura 3.22: Exemplo de um algoritmo de KNN [120]

3.3.6 *Support Vector Machines*

As *Support Vector Machines* (SVM) são uma das mais recentes técnicas de *supervised learning*, em que ideia principal é encontrar o hiperplano ideal para separar os pontos de dados em diferentes classes no espaço de características [86, 123].

O hiperplano pode ser, assim, definido como o limite de decisão que separa os pontos de dados pertencentes a diferentes classes. Este algoritmo tenta maximizar a margem entre os pontos mais próximos das classes, que formam os conhecidos *support vectors*, como é visível na Figura 3.23, uma vez que este facto permite aumentar a capacidade de generalização do modelo para novos dados e diminuir o erro presente nas suas previsões [106, 107].

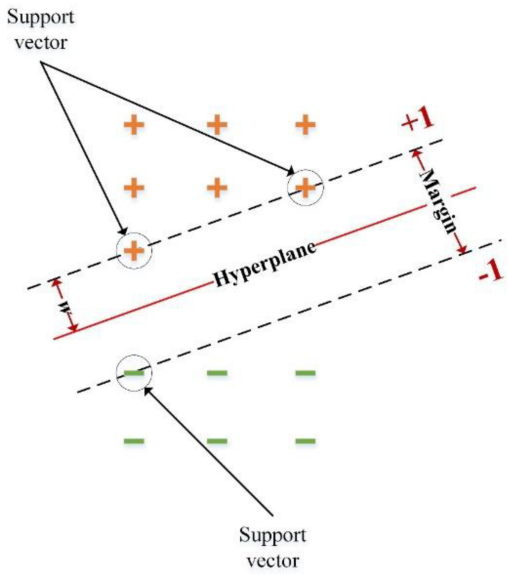


Figura 3.23: Esquema de um hiperplano de SVM [124]

Porém, quando os dados não são linearmente separáveis, este algoritmo aplica funções de *kernel* para transformar o espaço de características original num espaço de dimensão superior, de modo a permitir a separação linear, como se pode observar na Figura 3.24 [86, 125].

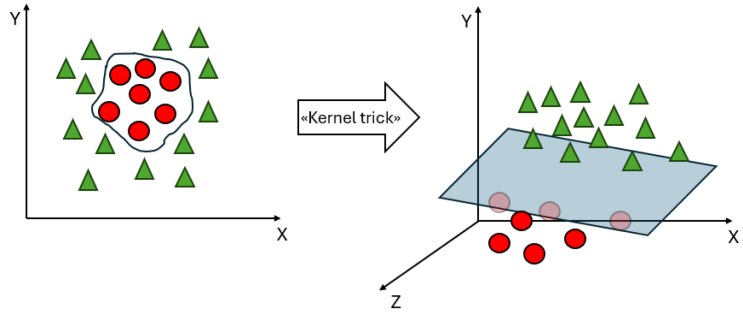


Figura 3.24: Exemplo de um truque de *kernel* [126]

Noutras situações, caso a separação dos pontos de dados não seja perfeita, existe a possibilidade de uma margem suave, ou seja, o algoritmo procura o melhor hiperplano que separe a maioria dos dados, contudo, adiciona uma penalização por cada ponto na classe incorreta, obrigando a um compromisso entre a maximização da margem e a minimização dos erros.

3.3.7 *K-Means Clustering*

K-Means Clustering é um dos algoritmos de *Unsupervised Learning* mais populares, focado em tarefas de agregação. Ao contrário dos algoritmos de *Supervised Learning*, em que os modelos são treinados a partir de dados rotulados, este algoritmo procura padrões em dados não rotulados, agrupando pontos de dados semelhantes, em *clusters*, com base nas suas características [106, 127].

Por isso, este algoritmo começa por selecionar aleatoriamente "K" centros iniciais para a criação dos *clusters*. Em seguida, atribui os pontos de dados em análise a um dos "K" grupos criados, dependendo da sua distância ao centro destes, como se demonstra na Figura 3.25a. Por fim, quando todos os dados se encontram atribuídos a um grupo, recalculam-se os centros de cada *cluster*, tendo em conta os novos pontos (Figura 3.25b) [86, 128].

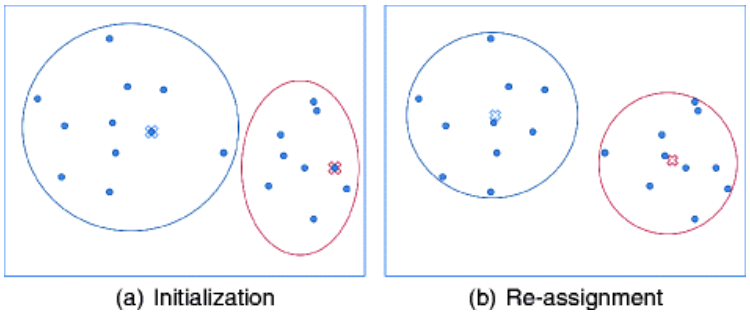


Figura 3.25: Exemplo do algoritmo de *K-Means Clustering* [129]

Este processo é repetido até se encontrarem os centros ótimos e os *clusters* se tornarem o mais distintos quanto possível.

É um algoritmo acessível, eficiente e flexível, porém, é dependente de uma boa escolha de fatores iniciais. Este algoritmo necessita que o número "K" de *clusters* a serem criados seja próximo do valor ótimo. Para além disso, a posição inicial dos centros pode ter um impacto significativo nos *clusters* finais, levando a soluções sub-ótimas [107, 130].

3.3.8 *Q-Learning*

O algoritmo de *Q-Learning* inclui-se na categoria de algoritmos de RL, em que um agente procura, a partir da interação com o ambiente, aprender as melhores ações a tomar num determinado estado [131].

A base deste algoritmo é a"Q-Table", uma matriz que faz corresponder um estado do ambiente a uma possível ação para o agente. Assim, os valores armazenados, conhecidos como"Q-values", estimam a recompensa esperada pelo agente ao tomar uma determinada ação para um determinado estado. Estes valores inicializados arbitrariamente são atualizados iterativamente através da experiência do agente, tendo em conta a regra de diferença temporal [132]:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A)) \tag{3.6}$$

- $Q(S, A)$ – corresponde ao"Q-Value" para o estado S e a ação A;
- $Q(S', A')$ – corresponde ao"Q-Value" para o estado futuro S' e a melhor ação futura A';
- α – a taxa de aprendizagem do modelo;
- γ – o fator de desconto para recompensas futuras;
- R – recompensa calculada para o estado atual e para a ação realizada.

O processo deste algoritmo é, assim, caracterizado por 5 fases principais descritas na Figura 3.26.

Durante o processo de aprendizagem, o agente pode escolher entre explorar novas ações para adicionar à sua tabela, o fenómeno de *exploration*, ou pode tirar vantagens de ações já conhecidas que produzem recompensas elevadas, *exploitation*. A escolha reside em estratégias como a ϵ-*Greedy Policy*, onde o agente tende a escolher a ação com maior"Q-value" conhecido, mas ocasionalmente toma ações aleatórias de exploração [134].

Por não utilizar um modelo tipo do ambiente em que é implementado, o algoritmo *Q-Learning* é ideal para uma vasta gama de aplicações em ambientes dinâmicos imprevisíveis ou complexos [107].

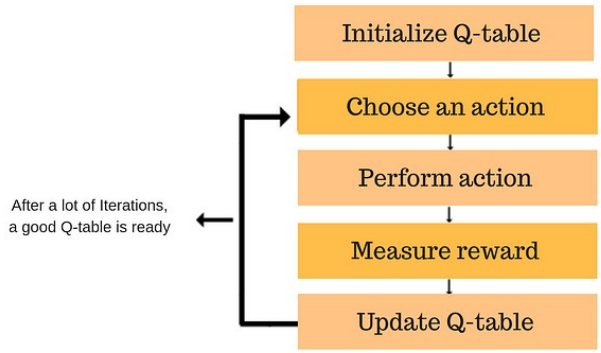


Figura 3.26: Fluxograma do algoritmo de *Q-Learning* [133]

Todavia, apresenta algumas desvantagens que o tornam pouco recomendado para certas situações. Em espaços de estados alargados ou contínuos, o tamanho da"Q-Table" pode tornar impraticável a implementação deste algoritmo; para além disso, nestes ambientes em que as recompensas futuras podem atrasar-se durante vários estados, existe uma dificuldade acrescida em propagar as informações úteis para os estados anteriores. Em certos contextos, também pode ser complicado explorar eficientemente o ambiente, o que pode levar a uma aprendizagem lenta ou a resultados sub-ótimos [132].

Tendo isto em mente, este algoritmo exige um ajuste cuidadoso da sua taxa de aprendizagem, do seu fator de desconto e do valor da estratégia ϵ-*Greedy Policy*, com o intuito de colmatar os problemas referidos.

3.3.9 *Proximal Policy Optimization*

O algoritmo de *Proximal Policy Optimization* (PPO) inclui-se nos algoritmos de RL, para treinar agentes a executar as ações mais válidas num ambiente. Ao contrário dos algoritmos como *Q-Learning*, que se concentram na estimativa do valor de ações ou estados, a categoria em que o algoritmo de PPO se insere procura atualizar a política de modo a maximizar as recompensas esperadas [135].

As políticas são estratégias ou regras que o modelo utiliza para decidir qual a ação a tomar em diferentes situações, ou seja, a ideia central é ajustar os parâmetros da política para melhorar a escolha das ações. Posteriormente, avalia-se até que ponto as ações do agente correspondem às recompensas esperadas. No final, a política deve estar ajustada de modo a dar preferência à escolha das ações que correspondem a recompensas mais elevadas [136].

O algoritmo de PPO é um incremento em relação aos métodos anteriores da categoria de *Gradient Policy*, em que era um desafio garantir uma aprendizagem estável e eficiente. Mudanças demasiado grandes na política podem implicar a degradação

do desempenho, enquanto mudanças muito pequenas podem abrandar significativamente o processo de aprendizagem [137].

O algoritmo de PPO aborda esta questão estabelecendo um equilíbrio entre a exploração e o aproveitamento. Para tal, aplica a seguinte função de objetivo para otimizar a política e evitar grandes atualizações da mesma [138]:

$$L^{clip}(\theta) = \hat{\mathbb{E}}_t \left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{3.7}$$

- $r(\theta)$ – rácio de probabilidade entre a nova política e a política anterior;
- $\hat{A}_t$ – função de vantagem que avalia a qualidade de uma ação em comparação com a ação média num estado específico;
- ϵ – hiperparâmetro que controla a gama de alterações permitidas à política.

A função "clip" implementada na função de objetivo garante que a atualização da política é limitada, levando a uma aprendizagem estável, o que torna este algoritmo mais eficiente, em comparação com outros métodos de *Gradient Policy* [138].

Por estas razões, tem sido aplicado numa variedade de domínios, demonstrando bons desempenhos nas suas tarefas, como: na área da robótica, por exemplo no controlo do movimento de braços robóticos ou robôs com pernas; na área de entretenimento, na formação de agentes de IA para jogar videojogos ou jogos de tabuleiro, na área dos transportes, em que procura otimizar as decisões da condução autónoma, entre outros [139].

3.3.10 *You Only Look Once*

You Only Look Once (YOLO) é um algoritmo de deteção de objetos que, ao contrário dos métodos tradicionais, visiona a sua tarefa como um único problema de regressão. Assim, este aplica uma única rede neuronal para realizar a previsão das *bounding boxes* dos objetos e a classificação dos mesmos, a partir de imagens completas, numa única avaliação [140].

Em comparação com os algoritmos prévios à introdução do YOLO, que precisavam de duas redes neuronais separadas para detetar e classificar objetos, o YOLO apresenta apenas uma, o que lhe garante uma maior velocidade de implementação. Para além disso, a capacidade de análise de uma imagem completa de uma só vez permite captar o contexto dos objetos detetados, aumentando assim a sua precisão e diminuindo a taxa de falsos positivos [141].

A ideia central do YOLO é dividir a imagem de entrada numa grelha de células $S*S$ em que cada célula é responsável por prever "B" *bounding boxes* e a probabilidade "C" de cada classe na célula, como está representado na Figura 3.27 [142].

Cada *bounding box* é representada por cinco valores: x, y, largura, altura e uma pontuação de confiança. Os primeiros quatro valores enquadram a *bounding box*

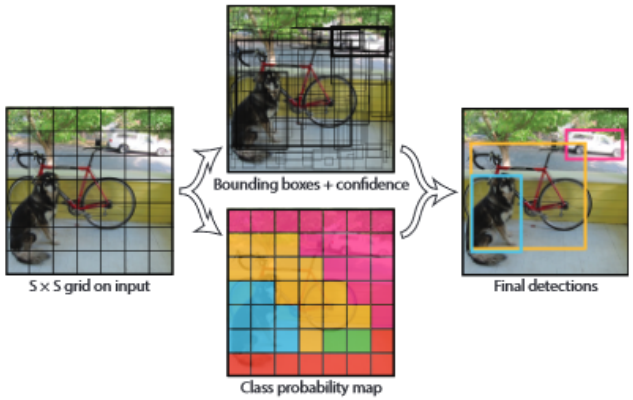


Figura 3.27: Modelo de YOLO em funcionamento [142]

no espaço da imagem, enquanto a pontuação de confiança reflete a *Intersection over Union* (IoU) entre a caixa prevista e a caixa verdadeira, como apresentado na equação (3.8).

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{3.8}$$

A estrutura de um modelo YOLO é constituída por três componentes principais: a cabeça, o pescoço e a *backbone*, representados na Figura 3.28 (a versão completa encontra-se no Anexo B).

A *backbone* é a parte da rede constituída por 24 camadas convolucionais para detetar as principais características de uma imagem e processá-las.

De seguida, o pescoço, composto por 2 camadas totalmente ligadas, utiliza as características identificadas pelas camadas anteriores para fazer previsões sobre probabilidades de classes e coordenadas das *bounding boxes*.

A cabeça é a camada de saída da rede responsável por classificar e detetar o objeto na imagem de entrada.

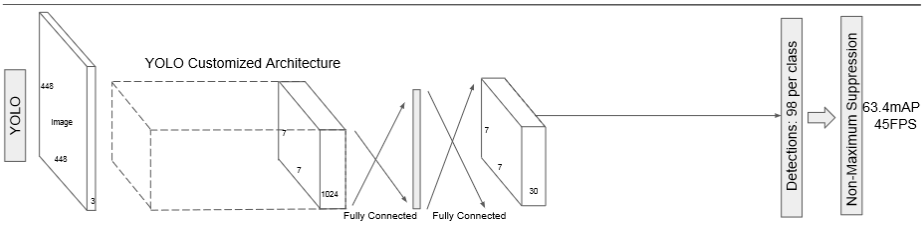


Figura 3.28: Estrutura base simplificada do modelo YOLO [143]

Em comparação com modelos anteriores de deteção de imagens, o YOLO é superior em celeridade, sendo um dos modelos mais populares e precisos em imagens com contexto, diminuindo a taxa de falsos positivos.

3.3.11 *Single Shot Multibox Detector*

O *Single Shot Multibox Detector* (SSD) é outro algoritmo de deteção de objetos que aborda as tarefas de deteção e classificação numa única passagem pela rede neuronal.

Este algoritmo aplica uma CNN para extrair mapas de características de uma imagem de entrada. Destaca-se a utilização de vários mapas de características de diferentes escalas, que captam informações de diferentes resoluções, permitindo detetar eficazmente objetos de diferentes tamanhos [144], como se demonstra na escala 8x8 da Figura 3.29b e na escala 4x4 da Figura 3.29c.

De seguida, para cada localização do mapa de características, o SSD prevê um conjunto de *Anchor Boxes* padrão, com vários tamanhos e escalas. Para cada caixa predefinida, o algoritmo prevê o ajuste desta caixa para melhor se adaptar à forma do objeto e prevê a probabilidade do objeto dentro da caixa pertencer a uma classe, como se pode observar na Figura 3.29c [143].

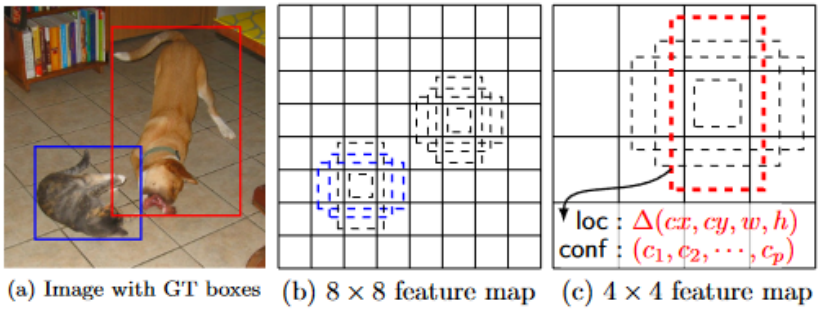


Figura 3.29: Modelo SSD em funcionamento [143]

A Figura 3.30 apresenta a estrutura base deste modelo.

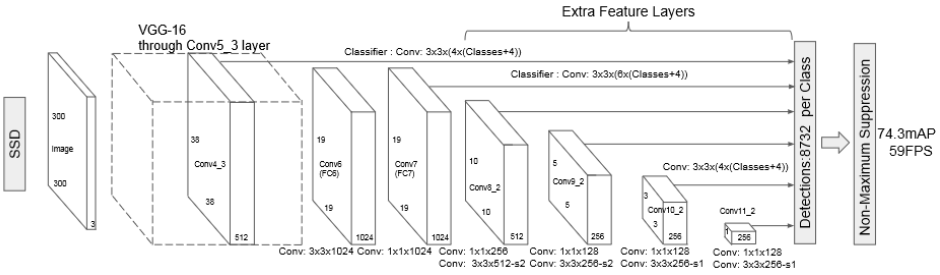


Figura 3.30: Estrutura do modelo SSD [143]

Este algoritmo é altamente eficiente e pode processar imagens rapidamente, tornando-o adequado para aplicações em tempo real. Em comparação com o algoritmo de YOLO, consegue detetar objetos de diferentes tamanhos com grande precisão. No entanto, apesar da sua velocidade ser comparável com a do YOLO, em certas situações pode exigir mais recursos de *hardware* para ser implementado.

3.3.12 *R-CNN*

O *R-CNN* é um algoritmo de deteção de objetos em que numa primeira fase propõe regiões que podem conter objetos; numa segunda fase é responsável pela classificação dessas regiões e pelo ajuste das *bounding boxes*. Estes fatores determinam uma maior lentidão na sua deteção, em comparação com os algoritmos anteriores de deteção de objetos [145].

Assim, o algoritmo de *R-CNN* começa por dividir a imagem de entrada em várias regiões ou sub-regiões, em que cada uma destas apresenta uma proposta de uma região que contenha um potencial objeto, como se demonstra na etapa 2 da Figura 3.31. Estas propostas, normalmente, baseiam-se em algoritmos ou métodos externos como *Selective Search*, por exemplo, que procura identificar regiões com texturas, cores ou formas diferentes, para criar um conjunto diversificado de propostas [146].

Seguidamente, estas 2000 propostas de regiões (aproximadamente) servem de entrada a uma rede CNN focada na extração do mapa de características, a etapa 3 da Figura 3.31 [145].

Em terceiro lugar, o mapa de características é enviado para um modelo de ML para classificação segundo as classes de objetos de interesse, a última etapa da Figura 3.31. Um dos modelos mais aplicados é o modelo de SVM, treinado em especial para cada classe que tentará determinar se a proposta de região contém ou não uma instância dessa classe [147].

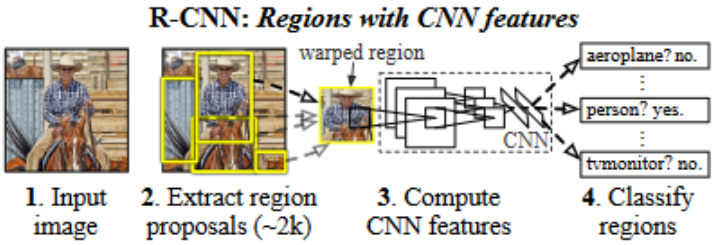


Figura 3.31: Algoritmo de R-CNN em funcionamento [145]

Por último, o *R-CNN* também pode identificar as *bounding boxes* de cada classe. Assim, um modelo de regressão separado é treinado para refinar a localização e o tamanho da *bounding box* em torno do objeto detetado [146].

Embora apresente valores de precisão elevados, a ineficiência na computação torna o processo de deteção muito lento, o que impede a sua aplicação em tempo real [147].

3.4 Conclusões

Tendo em conta os objetivos propostos para este projeto, nem todos os algoritmos explorados podem ser implementados no sistema a desenvolver. Cada algoritmo possui certas particularidades e vantagens, sendo que alguns se encontram dependentes do contexto em que são aplicados.

Para este projeto, escolheu-se implementar o algoritmo de *Q-Learning* para a tomada de decisões. A sua estrutura e modo de funcionamento foram as principais razões da sua escolha, uma vez que permitem que o sistema melhore a sua *performance* através da interação com o ambiente e recolha de recompensas. Como não precisa de um modelo tipo do ambiente em que é inserido, torna-se ideal para uma variedade de tarefas.

Do mesmo modo, selecionou-se o modelo YOLO para a componente de visão computacional devido às suas capacidades de deteção de objetos em tempo real numa só passagem pela rede e à sua elevada precisão e velocidade, em comparação com outros modelos de deteção de objetos (Figura 3.32).

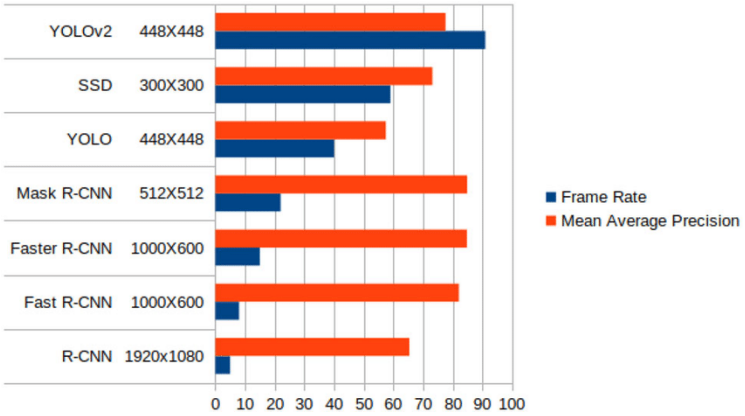
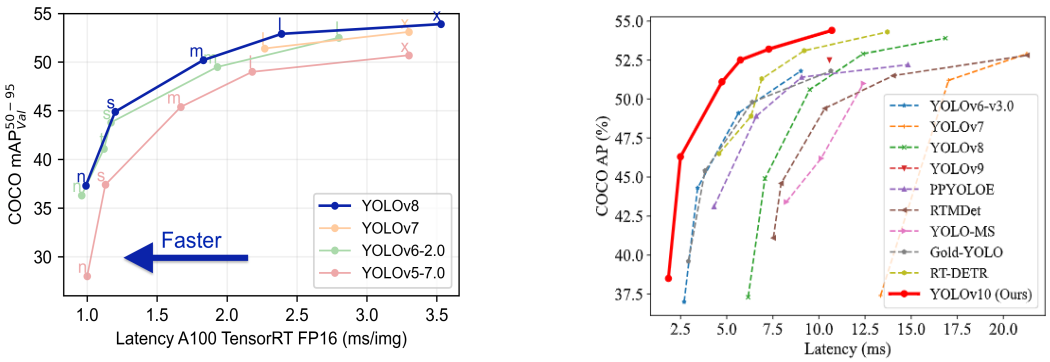


Figura 3.32: Comparação de resultados de vários modelos de deteção de objetos [148]



(a) *Performance* do YOLOv8 [149]

(b) *Performance* do YOLOv10 [150]

Figura 3.33: Comparação de várias versões de YOLO

Desde o seu lançamento inicial, o YOLO sofreu melhorias significativas com cada nova versão a oferecer uma maior velocidade de inferência e precisão de deteção. Como se demonstra na Figura 3.33, para o mesmo conjunto de dados, as versões mais recentes em cada comparação, o YOLOv8 na Figura 3.33a e o YOLOv10 na Figura 3.33b, apresentam os melhores resultados. No entanto, optou-se por aplicar o YOLOv5 neste projeto que, apesar de não ser a versão mais recente, é amplamente suportada e reconhecida pela sua fácil implementação e popularidade, apresentando resultados satisfatórios.

Capítulo 4

Módulos de Braços Robóticos

Este projeto tem como elemento central e crucial o braço robótico. O mesmo é responsável por interagir com o meio ambiente de uma forma precisa e controlada. Deste modo, as suas características base têm que ser meticulosamente analisadas para não ocorrerem danos, problemas ou dificuldades acrescidas à execução do projeto.

Existem vários modelos de braços robóticos disponíveis em diversos distribuidores de material de robótica, no entanto, nem todos os modelos apresentam os requisitos mínimos ou a capacidade para atingir os objetivos propostos.

Posto isto, após uma longa análise, a escolha final do braço robótico ficou reduzida a cinco modelos bastante distintos entre si (Figuras 4.1 - 4.5):

- Adeept RaspArm 4-DOF Robotic Arm Kit:

Figura 4.1: Adeept RaspArm 4-DOF Robotic Arm Kit [151]

- Keyestudio 4-DOF Robot Arm Kit:

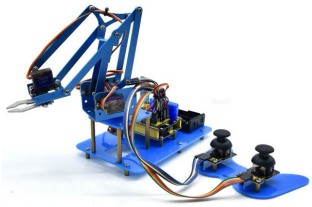


Figura 4.2: Keyestudio 4-DOF Robot Arm Kit [152]

- Adeept 5-DOF Robotic Arm Kit:

Figura 4.3: Adeept 5-DOF Robotic Arm Kit [153]

- Hiwonder ArmPi mini 5-DOF Robotic Arm Kit:

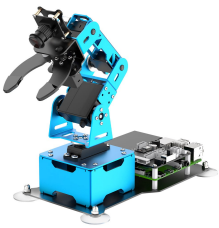


Figura 4.4: Hiwonder ArmPi mini 5-DOF Robotic Arm Kit [154]

- Makerfabs 6-DOF Robot Arm Kit:

Figura 4.5: Makerfabs 6-DOF Robot Arm Kit [155]

4.1 Estruturas

Partindo da análise da estrutura de cada um destes modelos, como se pode observar, cada modelo apresenta um número diferente de graus de liberdade. Como já foi referido anteriormente, em robótica, graus de liberdade referem-se à quantidade de movimentos independentes que um robô pode efetuar. Por isso, de modo a acrescentar um grau de liberdade normalmente é necessária a criação de uma nova articulação ou a adição de um novo motor.

No fundo, a importância dos graus de liberdade consiste em proporcionar mais flexibilidade e precisão de movimentos, permitindo realizar tarefas mais variadas e alcançar posições mais complexas.

Analisando os modelos em questão, os braços robóticos com 4 graus de liberdade têm geralmente uma estrutura mais simples, com os seus movimentos limitados à rotação da base, à inclinação do ombro e do cotovelo e à abertura da garra[1]. Por outro lado, os 5-DOF, possuem um grau de liberdade extra, a rotação do pulso, o que acaba por os tornar capazes de realizar tarefas mais complexas. Finalmente, os braços robóticos com 6 graus têm à sua disposição todos os movimentos anteriores com a adição da inclinação do pulso.

Tendo em conta o espaço 3D, apenas os braços robóticos com 6-DOF podem atingir todas as poses arbitrárias no espaço de trabalho, ou seja, são considerados manipuladores totalmente atuados. Os outros modelos são manipuladores sub-atuados, uma vez que podem atingir uma posição alvo, mas podem não atingir a orientação do alvo, tendo em conta a sua área de trabalho.

O material da composição de cada braço robótico também é um dos fatores de maior relevância aquando do processo de decisão. O material de cada peça deve ser resistente e leve para permitir que pequenos motores consigam operar as várias articulações, mantendo a rigidez necessária para movimentos precisos, oferecendo assim um equilíbrio entre robustez e peso. Desta forma, os materiais mais aplicados na construção de braços robóticos tendem a ser ligas de metal leve ou plásticos resistentes.

Os modelos analisados podem-se dividir em dois grupos consoante o material da sua composição: o modelo da Hiwonder e o modelo da Makerfabs pertencem ao grupo da liga de alumínio, enquanto os outros modelos são maioritariamente compostos por peças acrílicas (Figura 4.6), o que leva o seu preço a baixar significativamente.

Pode-se concluir que, para o contexto deste projeto, os modelos que devem ter primazia devem ser os com um maior número de graus de liberdade. O sistema projetado necessita de ser capaz de atingir as várias posições de jogo e a sua orientação para realizar a tarefa proposta, de jogar o jogo do galo de uma forma eficaz. Assim, tendo em conta a análise anterior, os modelos com 4 graus como o RaspArm e o *kit*

[1] Os fabricantes consideram a abertura da garra como um grau de liberdade.

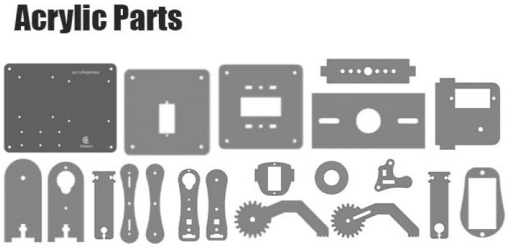


Figura 4.6: Peças acrílicas do módulo Adeept 5-DOF [153]

KeyeStudio perdem pontos por apresentarem uma falta de articulações para realizar os movimentos e atingir a orientação indicada. Na mesma medida, o seu material não é o mais resistente, comparado com o alumínio, na medida em que material acrílico pode sofrer mais facilmente danos ao final de várias utilizações. No entanto, ambos os materiais são excelentes para robôs sujeitos a pouca carga, fazendo a escolha do material ser menos relevante para este projeto. Na Tabela 4.1 resumem-se as principais características da estrutura destes módulos.

Tabela 4.1: Resumo da estrutura dos módulos

Modelo	**DOF**	**Material**	**Características**	Preço
Adeept RaspArm Kit	4	Acrílico	Peso e Agilidade	92,33 €
Keyestudio Robot Arm Kit	4	Acrílico	Peso e Agilidade	52,61 €
Adeept Robotic Arm Kit	5	Acrílico	Peso e Agilidade	64,62 €
Hiwonder ArmPi mini Kit	5	Alumínio	Resistência	125,74 €
Makerfabs Robot Arm Kit	6	Alumínio	Resistência	88,92 €

4.2 Atuadores

Os atuadores são componentes críticos nos braços robóticos, responsáveis por articular as várias partes do braço, permitindo realizar os movimentos pedidos. O desempenho e a precisão dos movimentos dependem maioritariamente da qualidade dos seus atuadores, neste caso os motores. Assim, a seleção de motores adequados para o controlo de um braço robótico requer a consideração de diversos fatores, como binário, velocidade e tensão de funcionamento.

Começando por analisar os motores constituintes de cada modelo, o RaspArm da Adeept é acompanhado por 4 servos SG-5010 (2 suplentes) e 2 MG946R. O servo SG-5010 (Figura 4.7) é caracterizado por operar entre os 4,8 e os 6,6 V, o que provoca uma alteração de velocidade entre 300 e 375 $^{\circ}/s$, respetivamente. Outro fator que também varia com o aumento da tensão é o binário atingido. Assim, para 4,8 V o binário é de 5,5 kg.cm e para 6,6 V é 6,5 kg.cm [156].

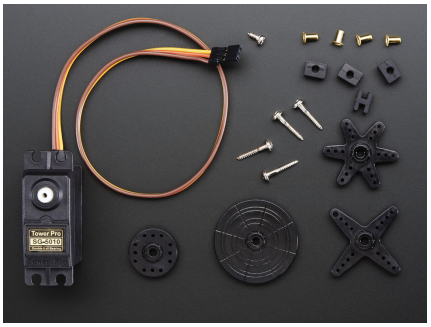
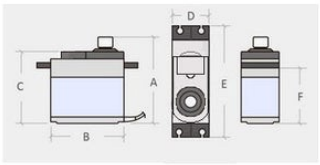


Figura 4.7: Servo-motor SG-5010 [156]

Por outro lado, o servo acompanhante, o MG946R apresenta os mesmos intervalos de tensão e de velocidade, contudo, este servo produz muito mais binário. O intervalo de binário deste servo é entre os 10,5 kg.cm e os 13 kg.cm (Figura 4.8) [157].



PRODUCT CONFIGURE TABLE

Weight(g)	55
Torque(kg)(4.8v)	10.5
Speed(sec/60deg)	0.2
A(mm)	42.7
B(mm)	40.9
C(mm)	37
D(mm)	20
E(mm)	54
F(mm)	26.8

Figura 4.8: Servo-motor MG946R [157]

Na Tabela 4.2 apresentam-se as principais características destes servo-motores.

Tabela 4.2: Servo-motores do módulo RaspArm

Servo	**Tensão (V)**	**Vel. ($^\circ/s$)**	**Binário (kg.cm)**	**Qtd.**
SG-5010	4,8 a 6,6	300 a 375	5,5 a 6,5	4
MG946R	4,8 a 6,6	300 a 375	10,5 a 13	2

Em relação ao modelo da KeyeStudio, este apresenta 4 servos MG90s, motores pequenos que operam entre os 4,8 V e os 6,0 V perfazendo 600 até 750 $^\circ/s$, respetivamente. O binário destes motores é pequeno, sendo apenas de 1,8 kg.cm até 2,2 kg.cm, dependendo da tensão de entrada [158]. A Tabela 4.3 resume as especificações deste servo.

Tabela 4.3: Servo-motores do módulo KeyeStudio

Servo	**Tensão (V)**	**Vel. ($^\circ/s$)**	**Binário (kg.cm)**	**Qtd.**
MG90s	4,8 a 6,0	600 a 750	1,8 a 2,2	4

O *kit* de 5-DOF da Adeept é acompanhado por 6 motores AD002 (1 suplente) que operam a 4,8 V com uma velocidade de 545,45 $^\circ/s$ e produzindo 2,0 kg.cm [159]. Na Tabela 4.4 apresentam-se as características deste servo.

Tabela 4.4: Servo-motores do módulo Adeept 5-DOF

Servo	**Tensão (V)**	**Vel. ($^\circ/s$)**	**Binário (kg.cm)**	**Qtd.**
AD002	4,8	545,45	2,0	6

Analisando agora os servos do ArmPi Mini da Hiwonder, este possui um LD-1501MG para o controlo da rotação da base, um LDX-218 para controlo da inclinação do ombro e 3 LFD-01M para as restantes articulações, como se apresenta na Figura 4.9.

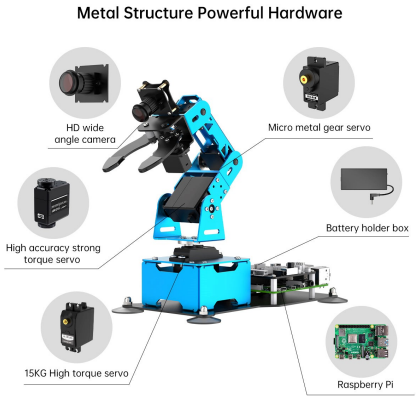


Figura 4.9: Componentes do modelo Hiwonder [154]

O LD-1501MG e o LDX-218 são servo-motores de binário elevado, desde 13 kg.cm a 6,0 V até 17 kg.cm a 7,0 V e 15 kg.cm a 6,0 V e 17 kg.cm a 7,4 V, respetivamente. No entanto, o aumento do binário vem a custo da velocidade dos motores, apenas 375 $^\circ/s$ no limite da sua tensão operacional, 7,4 V [160, 161]. Posto isto, estes servos são indicados para articulações que suportem mais peso e precisem de mais força. Por outro lado, o LFD-01M é um motor que opera entre os 4,8 V e os 6,0 V, com uma velocidade de 500 a 600 $^\circ/s$, produzindo binário de 1,5 kg.cm até 1,8 kg.cm, para os limites respetivos [162]. A Tabela 4.5 é responsável por resumir as capacidades dos servo-motores deste módulo.

Tabela 4.5: Servo-motores do módulo Hiwonder

Servo	**Tensão (V)**	**Vel. ($^\circ/s$)**	**Binário (kg.cm)**	**Qtd.**
LD-1501MG	6,0 a 7,0	375	13,0 a 17,0	1
LDX-218	6,0 a 7,4	375	15,0 a 17,0	1
LFD-01M	4,8 a 6,0	500	1,5 a 1,8	3

Tendo em conta o *kit* da Makerfabs, este oferece 5 MG996R (1 suplente) e 3 YF-6125MG. Os MG996R são os servos responsáveis por controlar a garra do braço robótico pois apresentam menos binário que os seus pares. Estes servos operam entre os 4,8 V e os 7,2 V, a uma velocidade entre os 352,94 e os 428,57 $^{\circ}/s$, produzindo desde 9,4 até 11 kg.cm, respetivamente [163]. Os YF-6125MG apresentam uma gama maior, operando entre os 4,8 V e os 7,2 V, produzindo 23,5 até 26,8 kg.cm para uma velocidade de 333,33 a 428,57 $^{\circ}/s$ como apresentado na Figura 4.10 [164]. Na Tabela 4.6 encontram-se as principais características dos servos deste modelo.

YF-6125MG		
	Maximum Pulse Width	50us
	Maximum Angle	180°
	Motor	Iron Core
	Weight	60g
	Bearing	2BB
	Output tooth	25T(Futaba)
	Cable	JR 30cm
Running Speed	0.18/60° @4.8V	0.14/60° @7.2V
Steering Torque	23.5Kg.cm @4.8V	26.8Kg.cm @7.2V
Control Voltage	4.8V	7.2V
Servo Size	40.5x20.5x40.5mm	
Dead Zone Width	2us	

Figura 4.10: Servo-motor YF-6125MG [164]

Tabela 4.6: Servo-motores do módulo Makerfabs

Servo	**Tensão (V)**	**Vel.** ($^{\circ}/s$)	**Binário (kg.cm)**	**Qtd.**
MG996R	4,8 a 7,2	352,94 a 428,57	9,4 a 11,0	5
YF-6125MG	4,8 a 7,2	333,33 a 428,57	23,5 a 26,8	3

Pode-se concluir que os servo-motores oferecidos pelo modelo da Makerfabs são os motores com uma maior capacidade para tarefas pesadas, como levantar objetos. Por outro lado, os motores do *kit* de 5-DOF da Adeept e os motores da KeyeStudio são os mais fracos entre os modelos analisados, deixando a dúvida que consigam levantar objetos e manter a posição do braço robótico estável. Um braço robótico deve ser estável e preciso. Caso os motores não possuam binário suficiente para manter a sua posição, quando sujeitos a um pouco de carga, então não são os mais indicados para o projeto.

Assim, procurando um ponto de equilíbrio, o modelo RaspArm, o *kit* Hiwonder e o *kit* Makerfabs são os que apresentam uma maior versatilidade de funções, possuindo motores com um nível de binário elevado para as articulações base, ou seja, para as articulações fulcrais para suportar e estabilizar o peso e carga do braço robótico.

4.3 Controladores

Um braço robótico não consegue operar eficazmente sem um "cérebro" que processe os sinais de entrada e identifique uma reação. Esta implica um controlo preciso das diversas articulações. Sendo assim, cada braço robótico necessita normalmente de um controlador e uma placa de controlo dos motores, um *Servo Driver*. Esta placa é responsável por proteger o controlador visto os motores consumirem mais potência do que a fornecida pelo controlador sem se danificar.

A gama de controladores para este tipo de aplicações é extensa, existindo vários modelos com características e um modo de operar similares. As maiores diferenças aparecem quando é necessária uma maior capacidade computacional, como por exemplo, a implementação de algoritmos de IA.

Nos modelos de braços robóticos analisados, os controladores oferecidos cingiam-se a dois tipos de controladores, Arduino e Raspberry PI.

4.3.1 Modelos Arduino

Começando pelos modelos com controladores Arduino, o modelo da KeyeStudio, oferece um controlador baseado no Arduino UNO com um ATmega328p.

O ATmega328p apresenta um processador de 8 bits de alta *performance* e de baixo consumo, funcionando a 16 MHz de frequência de *clock*. Para além disso, tem 32 kB de memória *flash* disponível, 2 kB de *Static Random-Access Memory* (SRAM) e 1 kB de *Electrically Erasable Programmable Read-Only Memory* (EEPROM), como apresentado na Tabela 4.7. Em termos de funcionalidades, possui 32 registos de 8 bits para uso genérico, com 6 portas de *Phase Width Modulation* (PWM), num total de 23 pinos de *General-Purpose Input/Output* (GPIO), tendo 6 deles capacidades analógicas e a *Universal Synchronous and Asynchronous Receiver-Transmitter* (USART) é programável.

Tabela 4.7: Características do ATmega328p

Arquitetura	**Frequência de *Clock***	**Memória (kB)**	**Registos**	**GPIO**
8 bits	16 MHz	32+2+1	32	23

Este controlador é acompanhado por uma placa de expansão de *Servo Driver* cujo fabricante não divulga muitas especificações, apenas divulgando a tensão e a corrente de entrada, 7 V a 15 V e 5 A, o tamanho da placa e o desenho final. O controlo dos servo-motores é realizado pelos pinos de PWM da placa de expansão, ligados diretamente ao controlador, como se demonstra na Figura 4.11.

O segundo modelo que apresenta um controlador Arduino é o *kit* de 5-DOF da Adeept. No entanto, existem duas versões deste modelo, uma compatível com Raspberry PI, que será apresentada mais à frente, e a outra oferecendo uma placa de

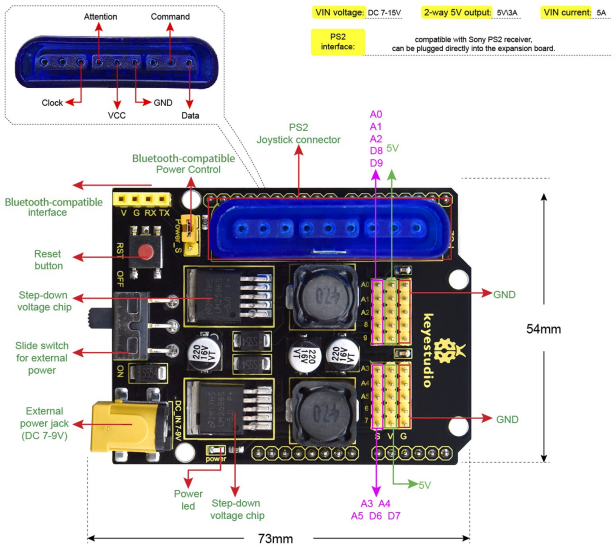


Figura 4.11: Placa de expansão de *Servo Driver* do *kit* KeyeStudio [152]

Servo Driver baseada num Arduino UNO, tendo um microcontrolador ATmega328p já incluído, a exemplo do modelo analisado anteriormente. Assim, o controlo dos motores é realizado, pelos pinos de PWM da placa de *Servo Driver* adaptada, apresentada na Figura 4.12.

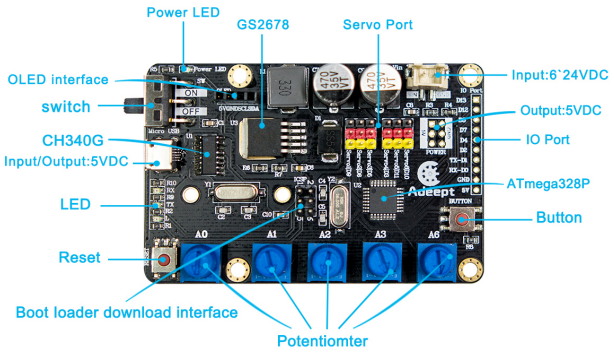


Figura 4.12: Placa Arduino *Servo Driver* do Adeept 5-DOF Kit

4.3.2 Modelos Raspberry PI

Falando agora do controlador Raspberry PI, este pode ser classificado como um pequeno computador, uma vez que, possui os elementos básicos: um processador dedicado, uma placa gráfica, uma memória e um sistema operativo, uma distribuição leve de Linux. Podem-se encontrar diversas iterações deste controlador no mercado, mas o modelo de braço robótico em causa, apenas é compatível com as versões do Raspberry PI 3 e do Raspberry PI 4.

O Raspberry PI 3B apresenta um processador de 64 bits *Quad Core* de 1,2 GHz e uma gráfica Videocore IV de 400 MHz, tendo ainda 1 GB de *Random Acess Memory* (RAM) nativa (Figura 4.13). Em termos de funcionalidades mais relevantes, este controlador pode comunicar por *Ethernet*, *Bluetooth* e *Wireless* e possui 40 pinos de GPIO, 4 portas *Universal Serial Bus* (USB) 2.0 e 1 porta para ligação de uma câmara Raspberry PI (resumido na Tabela 4.10).

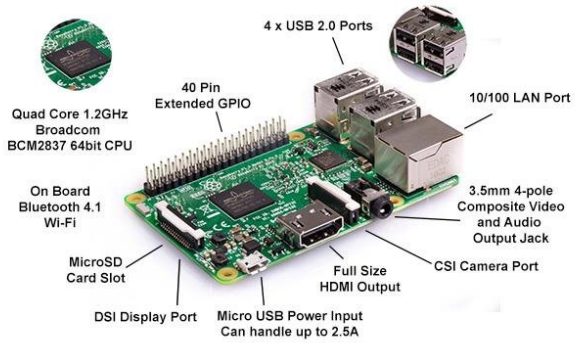


Figura 4.13: Raspberry PI 3B [165]

O Raspberry PI 4B, representado na Figura 4.14, é a evolução do 3B, apresentando melhores componentes: um processador de 64 bits *Quad Core* de 1,8 GHz e uma gráfica Videocore IV com 500 MHz. A memória RAM não se encontra limitada a 1 GB, podendo ser alterada para 2 GB, 4 GB ou 8 GB. Em termos de funcionalidades mais relevantes as diferenças para a iteração anterior aparecem nos protocolos de comunicação, que se encontram mais atualizados e nas portas de interligação, das anteriores 4 portas USB 2.0, agora 2 destas são compatíveis com USB 3.0 (resumido na Tabela 4.10).

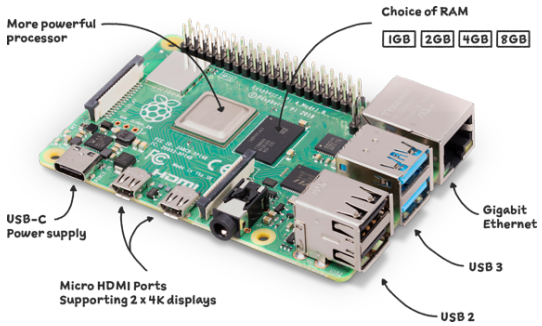


Figura 4.14: Raspberry PI 4B [166]

Em relação ao modelo Adeept 5-DOF compatível com Raspberry PI, este oferece apenas a placa de *Servo Driver*, ou seja, a placa Arm HAT que realiza a *interface* entre o controlador e os servo-motores, representada na Figura 4.15. Esta placa contém um PCA9685 responsável, em específico, pelo controlo dos motores, que comunica com o controlador através de *Inter-Integrated Circuit* (I2C).

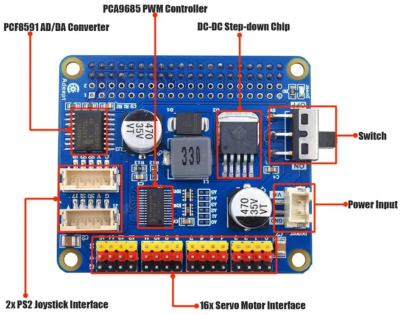


Figura 4.15: Placa Arm HAT Adeept 5-DOF [153]

O PCA9685 é um Circuito Integrado (IC) designado para controlar sinais PWM, sendo programável para frequências de 40 Hz até 1000 Hz. Este IC opera entre os 2,3 V e 5,5 V e possuí 4 endereços I2C para permitir o endereçamento de grupos de equipamentos em simultâneo [167]. A Tabela 4.8 apresenta as principais características deste IC.

Tabela 4.8: Características do IC PCA9685

IC	Tensão	Frequência	Comunicação	Endereços
PCA9685	2,3 V a 5,5 V	40 Hz a 1000 Hz	I2C	4

O *kit* RaspArm a exemplo do outro modelo da Adeept analisado, também não oferece o controlador para o qual é compatível, Raspberry PI 3 ou Raspberry PI 4, vendendo apenas a placa Robot HAT de controlo dos motores, comunicando com o controlador através de I2C. Esta placa faz uso do PCA9685 para controlo de motores em PWM e de um IC L298.

O IC L298 é um *Dual Full-Bridge Driver*, para controlo de equipamentos de carga elevada a partir de sinais *Transistor-Transistor Logic* (TTL), sinais em que o sinal lógico 0 é entre os 0 V e os 0,8 V e o sinal lógico 1 é entre 2 V e 5 V. Este *driver* pode ser alimentado com valores de tensão até 7 V, no entanto, pode fornecer tensões até aos 46 V e 4 A [168]. Por estas razões, este *driver* é normalmente aplicado em projetos de robótica e de carros autónomos com microcontroladores para o controlo de motores, uma vez que, pode fornecer potências elevadas que os microcontroladores não suportam. A Tabela 4.9 apresenta as principais características deste IC.

Tabela 4.9: Características do IC L298

IC	Tensão Entrada	Tensão Saída	Corrente Saída	Operação
L298	até 7,0 V	até 46 V	até 4 A	TTL

Em relação ao conteúdo do modelo da Hiwonder, este, ao contrário dos modelos anteriores, providencia um Raspberry PI 4B e uma placa de expansão para o controlo dos servo-motores (Figura 4.16). Esta placa possui 6 canais de PWM e 2 canais de barramento série para controlo de servo-motores e, ainda possui 4 canais para controlo de motores DC. Para além destes canais, contém ainda 3 ligações para comunicação I2C, o protocolo predileto para comunicação entre equipamentos com o Raspberry PI, e 2 *ports* GPIO.

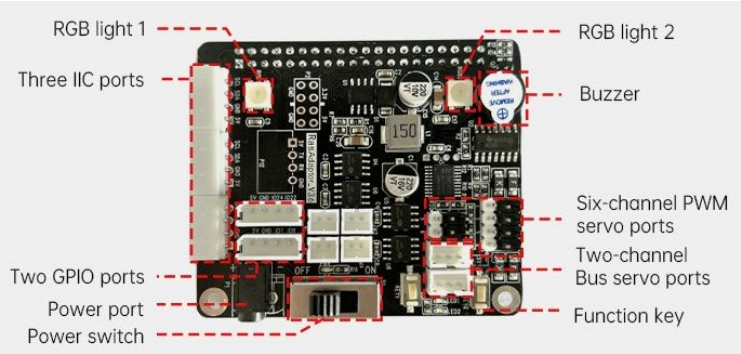


Figura 4.16: Placa de expansão da Hiwonder para controlo dos servos [154]

Finalmente, o modelo da Makerfabs apresenta um novo Raspberry PI, o PICO e a sua placa de controlo de servos. A placa contém um tradutor de nível lógico de 8 bits e 8 canais de *ports* GPIO para controlar os servos diretamente. O RP2040 ou, como é conhecido normalmente, Raspberry PI PICO, apresenta um processador *Dual Core* de 133 MHz com 2 MB de memória *flash* e 264 KB de SRAM (Figura 4.17). Em termos de funcionalidades, possui 26 pinos de GPIO a 3,3 V, sendo 3 deles analógicos, 2 interfaces para cada protocolo de comunicação, *Serial Peripheral Interface* (SPI), I2C e *Universal Asynchronous Receiver / Transmitter* (UART) e 16 canais de PWM (resumido na Tabela 4.10).

4.4 Decisão Final

Após uma análise minuciosa dos vários modelos de braços robóticos, considerando a sua estrutura, os seus atuadores e os seus controladores, é chegada a hora de selecionar o modelo mais adequado para este projeto.

Tendo em conta os graus de liberdade dos diferentes modelos, pode-se concluir que é necessário um mínimo de 5 servo-motores para garantir que o braço robótico possa alcançar as posições pedidas e manipular as peças do jogo de forma eficaz. Assim, a escolha fica reduzida aos modelos com 5 e 6. No entanto, o sexto motor não é essencial, uma vez que a inclinação do pulso adiciona complexidade desnecessária a este projeto.

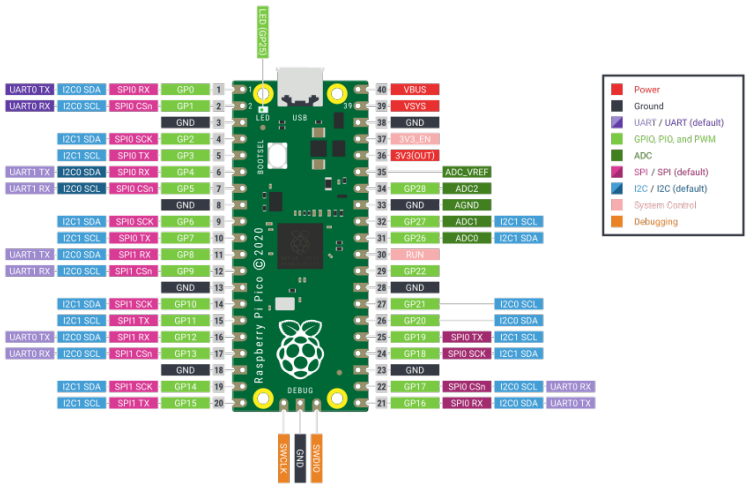


Figura 4.17: Raspberry PI PICO [169]

Considerando agora os atuadores, resumidos na Tabela 4.11, os motores oferecidos por cada modelo precisam de ter a capacidade de suportar o peso do braço e dos objetos que este possa levantar sem comprometer a precisão dos seus movimentos. Desta forma, os motores com binário inferior a 2 kg.cm podem não ser os mais indicados para o controlo deste projeto.

Outro fator importante é a capacidade do controlador e a sua interface de comunicação. Os modelos com controladores Arduino oferecem uma opção mais simples e económica, no entanto, não apresentam capacidade de processamento para algoritmos de IA complexos, sendo adequados para projetos mais simples.

Para colmatar este ponto, é necessária uma comunicação com um equipamento externo com uma maior capacidade de computação. Já os modelos baseados em Raspberry PI oferecem esta capacidade de processamento e computação, permitindo a implementação de algoritmos de IA.

Para este projeto é necessário um controlador com capacidade de processamento e computação para a implementação de algoritmos de IA. Deste modo, os modelos com Raspberry PI e controlo por I2C são as melhores escolhas.

Tendo em conta todos os fatores apresentados, a melhor alternativa para atingir os objetivos propostos é o ArmPi mini da Hiwonder. Contudo, um fator menos explorado, o custo de cada braço robótico, impossibilita a aquisição deste modelo, uma vez que, o seu custo total ultrapassa o orçamento máximo estabelecido de 100 €. Assim sendo, é necessário realizar um compromisso em certos fatores de modo a se chegar a um ponto de equilíbrio.

Por fim, decidiu-se adquirir o Adeept 5-DOF Robotic Arm Kit para Raspberry Pi, um modelo maioritariamente inferior ao modelo da Hiwonder, em termos de composição e atuadores.

Tabela 4.10: Características dos modelos de Raspberry PI

Modelo	**Processador**	**Gráfica**	**Memória**	**GPIO**	**Coms.**
3B	*Quad-Core* 1,2 GHz	Videocore IV 400 MHz	1 GB	40	*Ethernet* *Bluetooth* *Wireless* USB 2.0 Câmara
4B	*Quad-Core* 1,8 GHz	Videocore IV 500 MHz	1 GB a 8 GB	40	*Ethernet* *Bluetooth* *Wireless* USB 2.0 USB 3.0 Câmara
PICO	*Dual-Core* 133 MHz	-	2 MB + 246 KB	26	SPI I2C UART

Recapitulando, o *kit* selecionado é composto por material acrílico, apresentando 5 graus de liberdade[2]: rotação da base e do pulso, inclinação do ombro e do cotovelo e abertura da garra, como representado na Figura 4.18. É acompanhado por motores AD002 que operam a 4,8 V, produzindo até 2,0 kg.cm. O controlador adquirido é o Raspberry PI 3B que apresenta um processador de 64 bits *Quad Core* de 1,2 GHz e 1 GB de RAM nativa.

Em suma, o *kit* da Adeept escolhido oferece um equilíbrio entre custo e funcionalidade, fornecendo as características necessárias para atingir os objetivos propostos.

[2]Segundo o fabricante.

Tabela 4.11: Resumo dos atuadores de cada módulo

Servo	**Tensão (V)**	**Vel.** $(^\circ/s)$	**Binário (kg.cm)**
SG-5010	4,8 a 6,6	300 a 375	5,5 a 6,5
MG946R	4,8 a 6,6	300 a 375	10,5 a 13
MG90s	4,8 a 6,0	600 a 750	1,8 a 2,2
AD002	4,8	545,45	2,0
LD-1501MG	6,0 a 7,0	375	13,0 a 17,0
LDX-218	6,0 a 7,4	375	15,0 a 17,0
LFD-01M	4,8 a 6,0	500	1,5 a 1,8
MG996R	4,8 a 7,2	352,94 a 428,57	9,4 a 11,0
YF-6125MG	4,8 a 7,2	333,33 a 428,57	23,5 a 26,8

Figura 4.18: Módulo selecionado [153]

4.5 Teste do Módulo Adeept 5-DOF Robotic Arm Kit

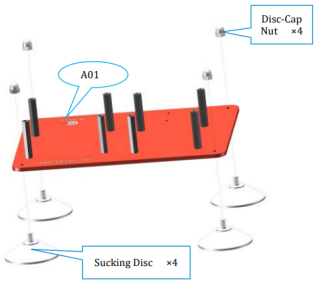
A Adeept, fabricante do *kit* selecionado, oferece a cada utilizador dos seus produtos um manual detalhado de instruções para a montagem e controlo de uma primeira instalação e utilização dos seus modelos. Para este, o tratamento não foi diferente, providenciando um manual simples que acompanha todas as etapas do arranque do seu produto, desde a montagem do equipamento à programação de um exemplo das funcionalidades do braço robótico.

4.5.1 Montagem

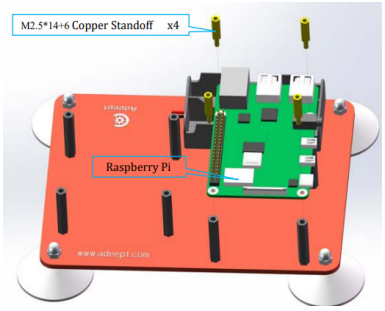
A primeira etapa da implementação deste sistema é claramente a montagem e teste do braço robótico escolhido, o Adeept 5-DOF Robotic Arm Kit. Para este efeito, o fabricante do *kit* já providencia um manual de instruções detalhado com o intuito de facilitar este processo demorado.

Resumindo o processo de montagem, podem-se agrupar os vários passos em (Figura 4.19-4.21):

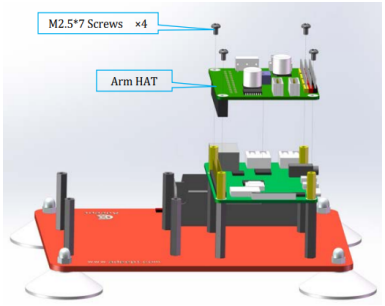
1. **Montagem da Base**
    - Fixar ventosas de suporte à placa de base;
    - Fixar controlador à base;
    - Montar a placa de controlo dos servos no controlador;
    - Montar mesa rotacional e servo de rotação da base.
2. **Montagem do Braço**
    - Montar a estrutura mecânica do braço;
    - Fixar o servo de inclinação do cotovelo;
    - Fixar o servo de inclinação do ombro;



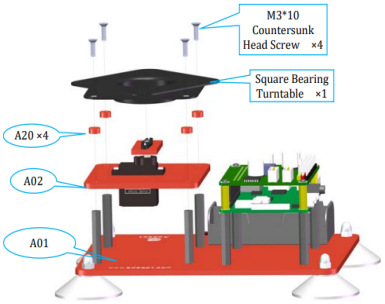
(a) Fixar ventosas de suporte à placa de base



(b) Fixar controlador à base



(c) Montar a placa de controlo dos servos no controlador



(d) Montar mesa rotacional e servo de rotação da base

Figura 4.19: Montagem da base

- Fixar o braço à estrutura base.

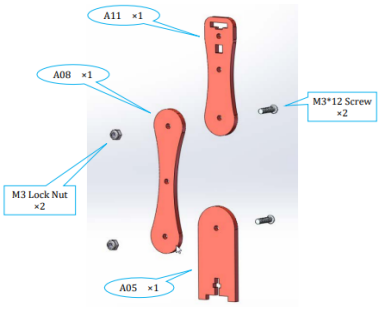
3. **Montagem da Garra**

- Fixar servo de rotação do pulso;
- Fixar servo de abertura da garra;
- Montar a estrutura da garra;
- Fixar a garra ao pulso.

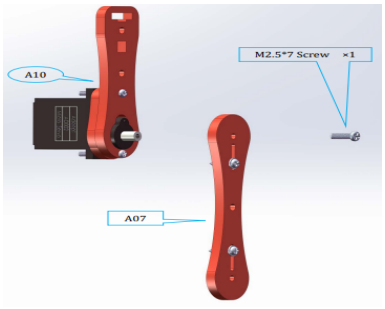
4.5.2 Controlo

Com o braço robótico já montado, agora é a vez de ligar o controlador e criar a lógica de controlo dos seus movimentos. Como já foi referido, o controlo dos servo-motores é realizado através de sinais PWM enviados pela placa Arm HAT e esta comunica através de I2C com o Raspberry PI.

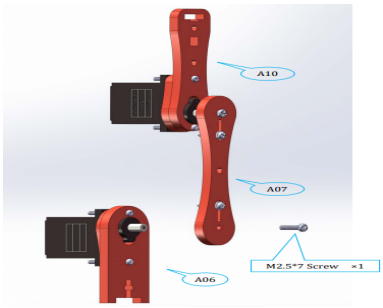
A exemplo da montagem do braço, o fabricante do *kit* também envia um pequeno manual para ajudar na implementação de um exemplo funcional do braço robótico. No entanto, ao analisar o código disponibilizado, é evidente a utilização de bibliotecas de Python de alto-nível sem uma maior explicação da sua função no controlo do braço robótico. Contudo, este código, explanado no fluxograma da Figura 4.22, é um excelente ponto de partida para a criação de um controlo à medida.



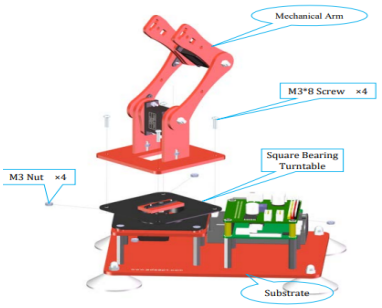
(a) Montar a estrutura mecânica do braço



(b) Fixar o servo de inclinação do cotovelo

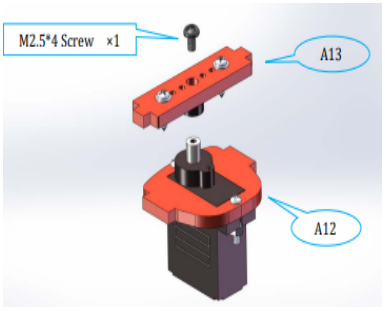


(c) Fixar o servo de inclinação do ombro

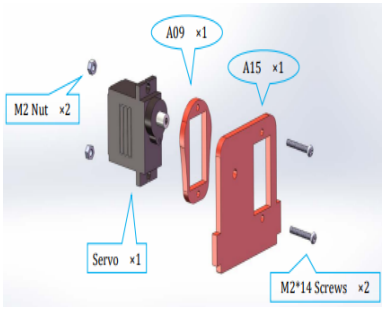


(d) Fixar o braço à estrutura base

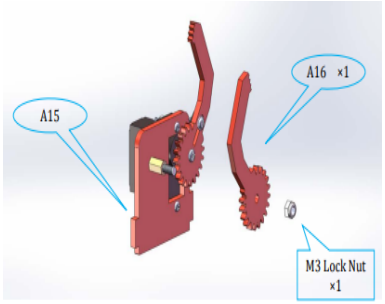
Figura 4.20: Montagem do braço



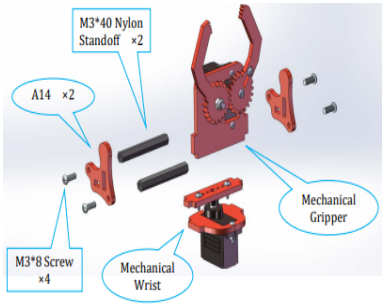
(a) Fixar servo de rotação do pulso



(b) Fixar servo de abertura da garra



(c) Montar a estrutura da garra



(d) Fixar a garra ao pulso

Figura 4.21: Montagem da garra

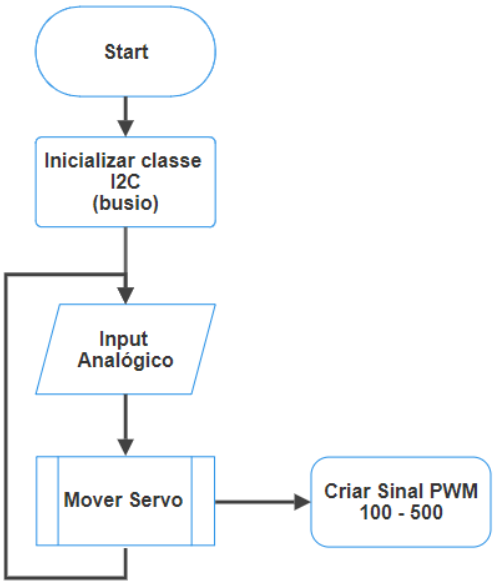


Figura 4.22: Fluxograma do código original do *kit* Adeept 5-DOF [153]

Analisando o *software* do fabricante, é possível encontrar as bases do controlo do braço robótico tendo em conta os sinais do utilizador através de um comando de jogos. Assim, primeiramente, o *software* converte estes *inputs* analógicos em ordens PWM para cada motor através de uma biblioteca da empresa Adafruit específica para o controlo do IC PCA9685, o componente principal da placa Arm HAT, sendo que o valor máximo de PWM (500) corresponde a 180° e o mínimo (100) a 0°. Posteriormente estas ordens são enviadas pela biblioteca busio, que tem o propósito de criar uma interface simples para comunicação I2C, sem que sejam precisos grandes conhecimentos de microcontroladores para obter resultados.

4.5.3 Resultados e Conclusões

A montagem do *kit* de braço robótico revelou-se relativamente simples, as instruções fornecidas eram detalhadas e as peças estavam devidamente identificadas de modo a facilitar o processo. As maiores dificuldades encontram-se na montagem dos servo-motores que para além de ser recomendado que estes estejam numa posição específica previamente a serem aplicados na estrutura, estes também não são muito robustos. Isto provoca a que as suas engrenagens, devido ao esforço necessário para aparafusar os encaixes, se danifiquem, deixando as peças soltas.

Os testes do controlo do braço robótico utilizando o código básico fornecido serviram para demonstrar a capacidade do braço para executar movimentos básicos. O código não é o mais apropriado para este projeto, uma vez que a mudança da posição dos motores é realizada de uma forma imediata e brusca, não tendo em

conta movimentos suaves e controlados. No entanto, demonstra o potencial para a implementação de algoritmos de controlo mais complexos.

Relativamente à *performance* geral do braço robótico, é um modelo sólido para pequenos projetos no domínio da robótica. Contudo apresenta sérias limitações, sendo a mais importante a capacidade dos motores. Estes servos são adequados para operações ligeiras e específicas, todavia, o seu fraco binário revela-se quando estes motores não conseguem suportar o próprio peso do braço, falhando a manter a sua posição ou sobreaquecendo, provocando danos irreparáveis nestes motores. A maior parte destas situações acontecem quando o braço se encontra em posições com um centro de massa deslocado fora da estrutura ou quando se tenta levantar objetos mais pesados. Para se ultrapassar esta desvantagem é necessário adquirir servos extra com mais capacidade e mais potentes, um ligeiro contratempo.

4.6 Movimentos do Módulo Adeept Robotic Arm

Os movimentos controlados de um braço robótico são cruciais para executar tarefas com precisão e exatidão. Para controlar estes movimentos, aplicam-se as equações da cinemática que descrevem as relações entre os elementos do braço e o espaço tridimensional. Estas equações dividem-se em dois tipos principais: cinemática direta e cinemática inversa.

A cinemática direta refere-se ao cálculo da posição e da orientação da ponta final do braço, tendo em conta os ângulos das articulações. Por outro lado, a cinemática inversa calcula os ângulos das articulações necessários para obter a posição e a orientação finais desejadas.

4.6.1 Cinemática Direta

A cinemática direta determina a posição final do sistema com base nos ângulos de cada servo-motor. Para facilitar o cálculo da cinemática, pode-se considerar o sistema como tendo 3 graus de liberdade. Assim, a posição final coincide com a articulação do pulso.

Existem diversas formas e técnicas para o cálculo da cinemática direta, uma das mais comuns e normalizadas é a convenção Denavit-Hartenberg (DH), que descreve as eventuais relações entre as ligações de um braço robótico. Cada elo pode ser configurado com base em quatro parâmetros:

- θ - Ângulo do elo em torno do eixo z;
- d - Deslocamento do elo ao longo do eixo z;
- a - Comprimento do elo ao longo do eixo x;
- α - Torção do elo em torno do eixo x;

O primeiro passo na aplicação destes parâmetros é a definição dos eixos de cada articulação. Assim considerando a placa base como o ponto de partida fixo, todas as outras articulações terão que se apresentar em função dos eixos definidos pela base, quando as articulações se encontram na sua posição de repouso. No final obtém-se os eixos demonstrados na Figura 4.23.

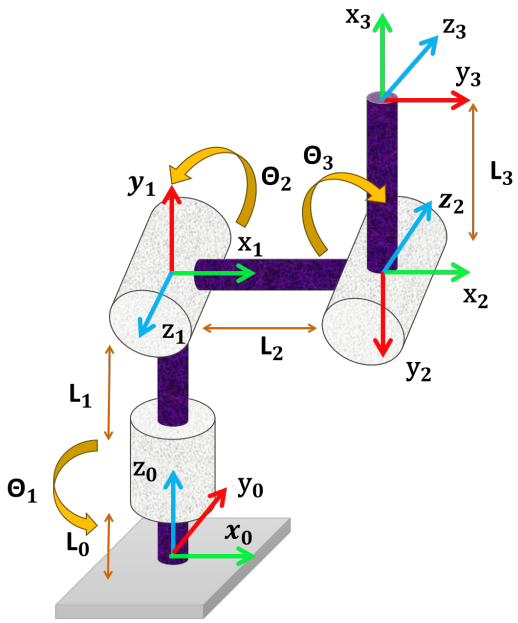


Figura 4.23: Configuração dos eixos

Com os eixos definidos, podem identificar-se os parâmetros de DH para cada elo, conforme a Tabela 4.12.

Tabela 4.12: Parâmetros Denavit-Hartenberg para o modelo

i	a_i	α_i	d_i	ϕ_i
1	0	$\frac{\pi}{2}$	$L_0 + L_1$	θ_1
2	L_2	π	0	θ_2
3	L_3	0	0	$\theta_3 - \frac{\pi}{2}$

Através da análise e medição da estrutura do braço robótico, sabe-se que, a distância L_0 corresponde a 4,5 cm, a distância L_1 a 5,5 cm, L_2 a 6,5 cm e L_3 a 11 cm.

De seguida, calculam-se as matrizes de transformação de cada elo segundo as leis da convenção de DH:

$$T_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cdot \cos\alpha_i & \sin\theta_i \cdot \sin\alpha_i & a_i \cdot \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cdot \cos\alpha_i & -\cos\theta_i \cdot \sin\alpha_i & a_i \cdot \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Analisando o sistema, pode observar-se que a primeira matriz, apresenta o elo do ombro em função da base. O comprimento da ligação entre a base e o ombro é de 4,5 cm, adicionado ao desvio no eixo z da base à origem com o comprimento de 5,5 cm. Esta articulação apresenta, ainda, uma rotação no sentido original do eixo z de $\frac{\pi}{2}$. Pelo preenchimento da matriz, obtém-se os dados:

$$T_1^0 = \begin{bmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & -\cos(\theta_1) & 0 \\ 0 & 1 & 0 & 5,5+4,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Em relação à segunda matriz, esta define a articulação do cotovelo em função dos eixos do ombro. Assim, o comprimento da ligação é de 6,5 cm em x e este elo apresenta um sentido diferente do eixo z do elo anterior, sendo α_2 igual a π, obtendo-se:

$$T_2^1 = \begin{bmatrix} \cos(\theta_2) & \sin(\theta_2) & 0 & 6,5 \cdot \cos(\theta_2) \\ \sin(\theta_2) & -\cos(\theta_2) & 0 & 6,5 \cdot \sin(\theta_2) \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Finalmente, a articulação do pulso em função do cotovelo. O comprimento da ligação é de 11 cm e não apresenta rotação do eixo z, assim:

$$T_3^2 = \begin{bmatrix} \cos(\theta_3 - \frac{\pi}{2}) & -\sin(\theta_3 - \frac{\pi}{2}) & 0 & 11 \cdot \cos(\theta_3 - \frac{\pi}{2}) \\ \sin(\theta_3 - \frac{\pi}{2}) & \cos(\theta_3 - \frac{\pi}{2}) & 0 & 11 \cdot \sin(\theta_3 - \frac{\pi}{2}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

Por fim, pode obter-se a matriz final, que transforma a posição do pulso tendo em conta a matriz base, através da multiplicação das várias matrizes. A matriz final apresenta a orientação e posição do atuador final.

$$T_3^0 = T_1^0 \times T_2^1 \times T_3^2 \tag{4.5}$$

Desta matriz final retiram-se as equações da cinemática direta presentes nas equações (4.6), (4.7) e (4.8) para as coordenadas de posição x, y e z, respetivamente.

$$\begin{aligned} x = {} & 11 \cdot \cos(\theta_1) \cdot \cos(\theta_2) \cdot \cos\left(\theta_3 - \frac{\pi}{2}\right) \\ & + 6,5 \cdot \cos(\theta_1) \cdot \cos(\theta_2) \\ & + 11 \cdot \cos(\theta_1) \cdot \sin(\theta_2) \cdot \sin\left(\theta_3 - \frac{\pi}{2}\right) \end{aligned} \tag{4.6}$$

$$y = 11 \cdot \sin(\theta_1) \cdot \cos(\theta_2) \cdot \cos\left(\theta_3 - \frac{\pi}{2}\right) + 6,5 \cdot \sin(\theta_1) \cdot \cos(\theta_2) + 11 \cdot \sin(\theta_1) \cdot \sin(\theta_2) \cdot \sin\left(\theta_3 - \frac{\pi}{2}\right) \quad (4.7)$$

$$z = 11 \cdot \sin(\theta_2) \cdot \cos\left(\theta_3 - \frac{\pi}{2}\right) - 11 \cdot \cos(\theta_2) \cdot \sin\left(\theta_3 - \frac{\pi}{2}\right) + 6,5 \cdot \sin(\theta_2) + 5,5 + 4,5 \quad (4.8)$$

Ao testar estas equações para algumas posições, pode verificar-se a sua veracidade, em comparação com a realidade do braço robótico. A Tabela 4.13 e a Figura 4.24 apresentam alguns dos ângulos testados e as suas respetivas posições.

Tabela 4.13: Teste da cinemática direta

θ_i	**X**	**Y**	**Z**
$(0^\circ, 0^\circ, 0^\circ)$	6,5	0,0	21,0
$(0^\circ, 90^\circ, 0^\circ)$	-11,0	0,0	16,5
$(0^\circ, 90^\circ, 90^\circ)$	0,0	0,0	27,5

4.6.2 Cinemática Inversa

Por oposição, a cinemática inversa determina os ângulos de cada servo baseados na posição final do sistema. Considerando o sistema como tendo 3 graus de liberdade, desta vez aplica-se trigonometria para o cálculo dos ângulos-base da cinemática inversa, como se demonstra na Figura 4.25.

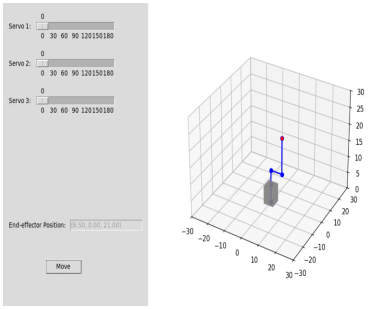
O primeiro ângulo, o ângulo da base θ_1, corresponde à rotação da base em torno do eixo z. Assim, pode-se formar um triângulo entre o centro da base, a abcissa da posição e a projeção da posição sobre o plano horizontal (triângulo entre os pontos 0-P_x-P').

A hipotenusa deste triângulo é o segmento de reta 'r' (Base-Posição) e, assumindo as coordenadas da posição final representadas por (P_x, P_y, P_z), calcula-se o ângulo da base pela relação:

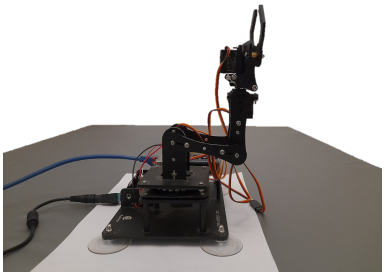
$$\theta_1 = \arctan 2\left(\frac{P_y}{P_x}\right) \quad (4.9)$$

Os restantes 2 ângulos exigem um pouco mais de atenção, uma vez que estão dependentes um do outro.

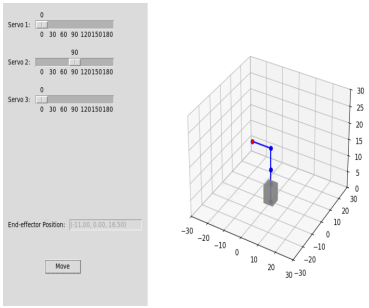
Podem-se desenhar 2 triângulos como apresentado na Figura 4.26: o primeiro apresenta os vértices na articulação do ombro, na do cotovelo e na posição final do



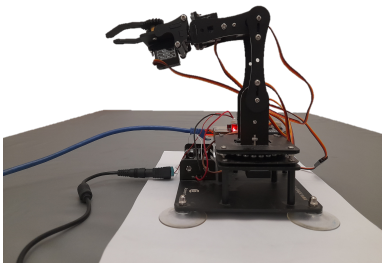
(a) Simulação dos ângulos $(0^{\circ}, 0^{\circ}, 0^{\circ})$



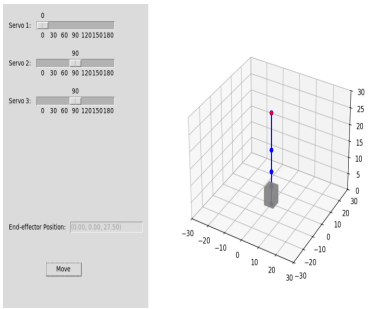
(b) Realidade dos ângulos $(0^{\circ}, 0^{\circ}, 0^{\circ})$



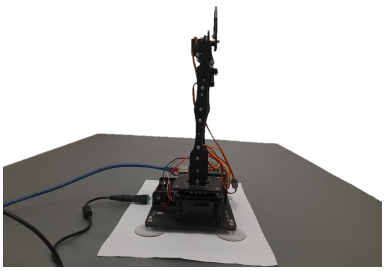
(c) Simulação dos ângulos $(0^{\circ}, 90^{\circ}, 0^{\circ})$



(d) Realidade dos ângulos $(0^{\circ}, 90^{\circ}, 0^{\circ})$



(e) Simulação dos ângulos $(0^{\circ}, 90^{\circ}, 90^{\circ})$



(f) Realidade dos ângulos $(0^{\circ}, 90^{\circ}, 90^{\circ})$

Figura 4.24: Teste da cinemática direta

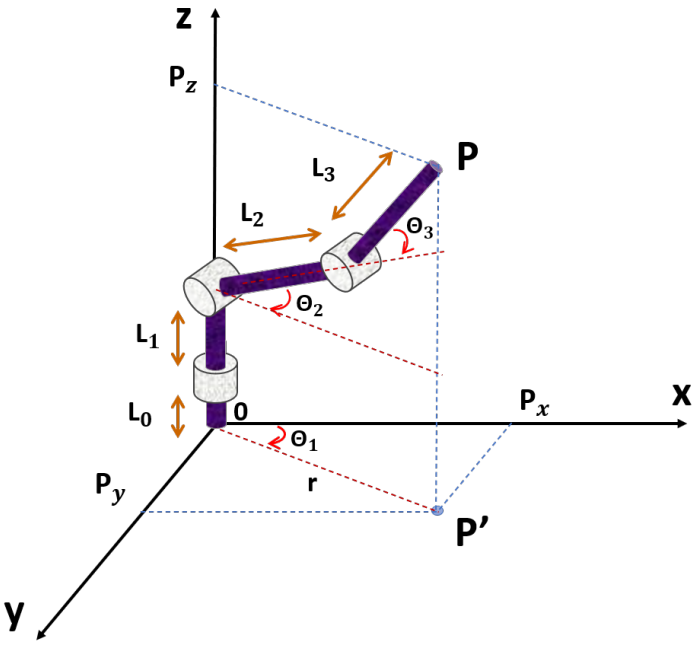


Figura 4.25: Ângulos e trigonometria do modelo

atuador (triângulo Q-P-R); o segundo triângulo, adjacente ao primeiro, apresenta os vértices no ombro, na posição final e na projeção da posição final sobre o plano horizontal deslocado para a articulação do ombro (triângulo Q-P-T).

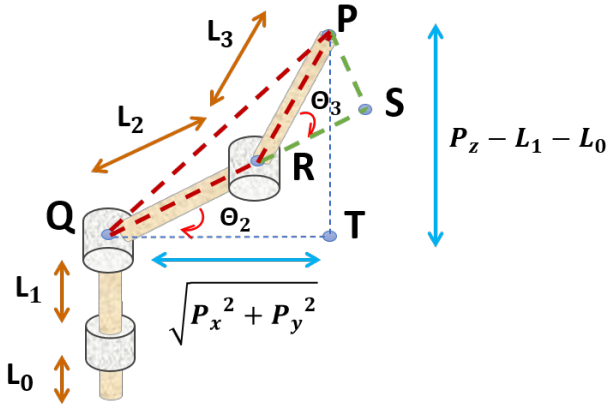


Figura 4.26: Triângulos do ombro e cotovelo

Aplicando a lei dos cossenos ao triângulo representado na Figura 4.27, consegue-se calcular o ângulo da articulação do cotovelo, o ângulo θ_3 através da equação (4.10).

$$\theta_3 = \arccos\left[\frac{P_x^2 + P_y^2 + (P_z - L_1 - L_0)^2 - L_2^2 - L_3^2}{2 \cdot L_2 \cdot L_3}\right] \tag{4.10}$$

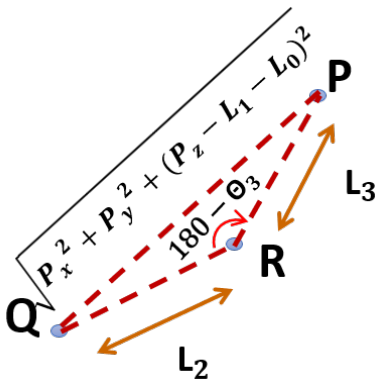


Figura 4.27: Triângulo Q-P-R

Por fim, com θ_1 e θ_3 calculados, descobre-se θ_2 pela subtração do ângulo $P\hat{Q}R$ ao ângulo $P\hat{Q}T$ (Figura 4.28) como expresso na equação (4.11).

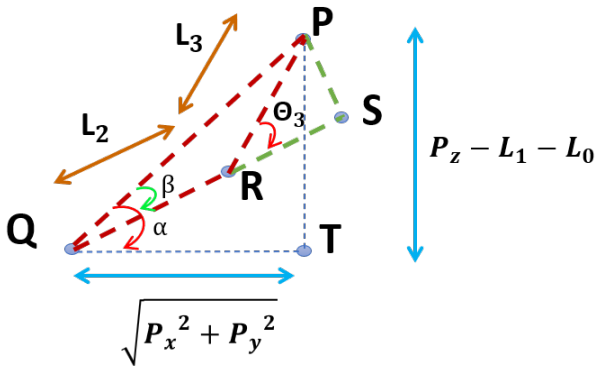


Figura 4.28: Cálculo do θ_2

$$
\begin{aligned}
\theta_2 &= \alpha - \beta \\
\alpha &= \arctan\left(\frac{P_z - L_1 - L_0}{\sqrt{P_x^2 + P_y^2}}\right) \\
\beta &= \arctan\left(\frac{\sin\theta_3 \cdot L_3}{L_2 + \cos\theta_3 \cdot L_3}\right) \\
\theta_2 &= \arctan\left(\frac{P_z - L_1 - L_0}{\sqrt{P_x^2 + P_y^2}}\right) - \arctan\left(\frac{\sin\theta_3 \cdot L_3}{L_2 + \cos\theta_3 \cdot L_3}\right)
\end{aligned}
\tag{4.11}
$$

Devido às equações envolvidas, o problema da cinemática inversa pode conduzir a múltiplas soluções válidas, frequentemente designadas por configurações *elbow up* e *elbow down*, uma vez que o braço robótico pode atingir o mesmo ponto no espaço com diferentes ângulos θ.

Capítulo 5

Desenvolvimento do Sistema

A arquitetura do sistema proposta neste capítulo destaca a análise das principais características dos componentes de *hardware* e *software*, assim como a sua integração de modo a alcançar os objetivos pretendidos.

O sistema projetado para jogar o jogo do Galo controla, através de um Raspberry PI 3B, um braço robótico com cinco graus de liberdade. O controlo deste braço robótico envolve um modelo YOLO de visão computacional para identificação de objetos em tempo real, treinado para identificar os vários elementos do jogo do Galo e um algoritmo de tomada de decisão, o algoritmo *Q-Learning* de RL, que utiliza os resultados do YOLO para controlar o braço robótico de forma autónoma durante o jogo.

Finalmente, uma interface interativa facilita o controlo e possibilita o acompanhamento em tempo real de todo o processo de controlo.

5.1 Arquitetura Proposta

A arquitetura proposta pode ser dividida em quatro componentes principais interligados de modo a possibilitar o sucesso da tarefa, jogar o jogo do galo.

O sistema projetado, representado na Figura 5.1, utiliza uma câmara para capturar em tempo real as imagens do tabuleiro do jogo.

Numa segunda fase, estas imagens são enviadas para o controlador, o Raspberry PI 3B, que aplicando um modelo de visão computacional, mais especificamente, um modelo YOLO, procura identificar e localizar os vários elementos do jogo. A saída

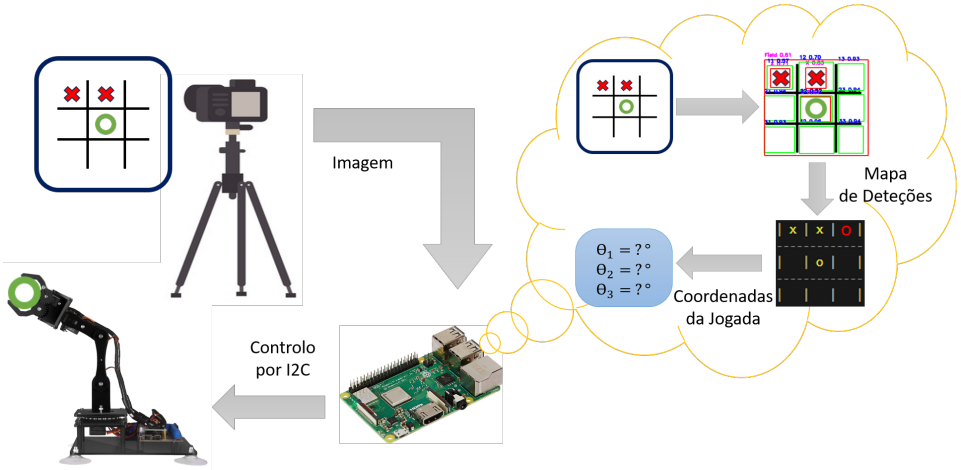


Figura 5.1: Arquitetura proposta

deste módulo é um mapa de deteções que permite ao algoritmo de tomada de decisão avaliar o estado atual do jogo e determinar a melhor jogada.

Posteriormente, o Raspberry PI traduz as coordenadas da jogada a realizar em ângulos de articulação e comanda através de I2C os servo-motores do braço robótico para executar a jogada pensada.

5.1.1 Braço Robótico

O braço robótico é o elemento principal do sistema, é responsável pela interação física com o ambiente, executando as ordens dadas pelo controlador e movendo-se para posições específicas no espaço.

O modelo Adeept 5-DOF Robotic Arm Kit, presente na Figura 5.2, é controlado pelo Raspberry PI 3B através de comunicação I2C, para receber os comandos de movimento com base nas decisões tomadas pelo algoritmo de tomada de decisão e manipular as peças, procurando executar as jogadas determinadas pelo sistema como a melhor ação para ganhar no jogo do galo.

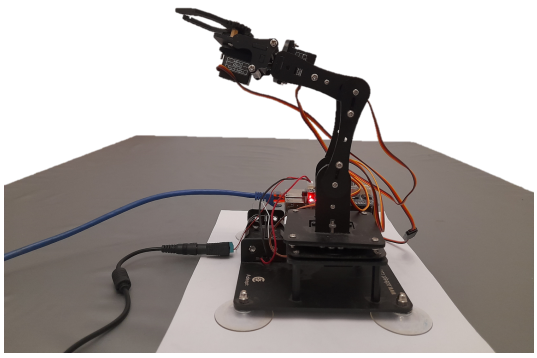


Figura 5.2: Braço robótico

5.1.2 Visão Computacional

O sistema de visão computacional tem como função interpretar as imagens de entrada e compreender o estado do jogo em tempo real. O sistema baseia-se num modelo YOLO, um algoritmo focado na deteção de objetos, para identificar a posição das peças do jogo no tabuleiro. Posteriormente, a informação é enviada a fim de ser tomada uma decisão informada.

Assim, o sistema de visão computacional foca-se na identificação e classificação das peças do jogo do galo e determinação das suas posições no tabuleiro de jogo, permitindo ao sistema compreender o estado atual do mesmo e mover o braço robótico para a posição onde vai realizar a jogada. A Figura 5.3 ilustra os resultados da identificação e classificação dos elementos do jogo do galo.

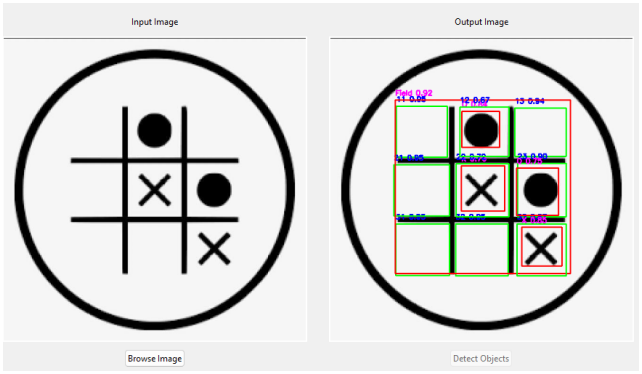


Figura 5.3: Resultados da visão computacional

5.1.3 Tomada de Decisão

O núcleo deste sistema é o algoritmo de tomada de decisão, cuja responsabilidade abrange a análise do estado do jogo e a determinação da jogada ideal através de vários algoritmos, em especial o algoritmo de *Q-Learning* que melhora o seu desempenho ao longo do tempo, aprendendo com as experiências passadas.

Assim, pode-se descrever o modo de funcionamento da tomada de decisão através de quatro etapas. Primeiro o algoritmo recebe as informações da visão computacional sobre o estado atual do tabuleiro de jogo, que inclui as posições de todas as peças em campo; em seguida, determina a jogada ideal; em terceiro lugar emite os comandos de movimento para o braço robótico; e, por fim, atualiza a sua base de conhecimentos com base no resultado atual.

5.2 Interface Gráfica

A interface gráfica desenhada permite ao utilizador interagir com o sistema e acompanhar o processo de controlo do braço robótico, proporcionando um ambiente claro

e compreensivo (Figura 5.4).

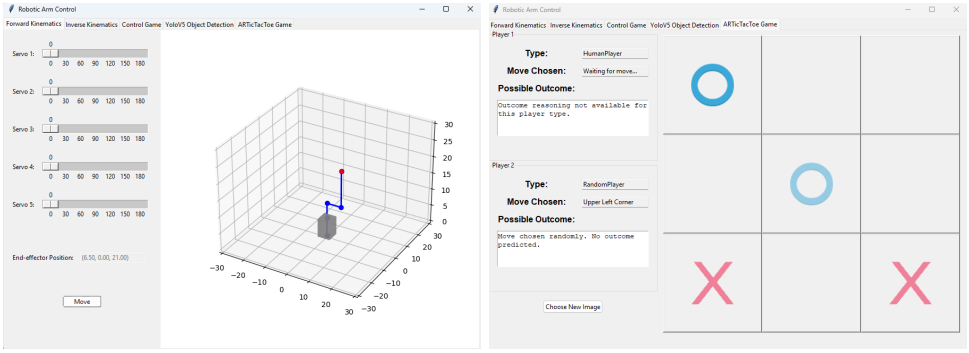
Esta interface encontra-se dividida em várias secções, cada uma focada num aspeto diferente do sistema. Assim, a primeira secção foi concebida para habilitar o controlo individual do braço, permitindo que se comande cada atuador para um ângulo específico. A segunda secção foca-se nos movimentos direcionados para uma posição, permitindo que o utilizador mova o braço robótico para um ponto no plano cartesiano.

Em relação à terceira secção, numa fase inicial inicia o objetivo deste sistema, possibilitando a configuração da ordem dos jogadores e das características do modelo de tomada de decisão, permitindo o início do jogo.

Em seguida, a componente de visão computacional permite a escolha da imagem representativa do estado do jogo e a demonstração da sua análise.

Por fim, a secção final representa o estado do jogo atual, mostrando as posições de todas as peças no tabuleiro virtual, tal como detetadas pelo sistema de visão. Para além disso, descreve a jogada ideal e envia a informação ao braço robótico.

Desta forma, a interface gráfica está fortemente integrada com os outros componentes do sistema, recebendo dados do sistema de visão para atualizar o tabuleiro virtual; apresentando as decisões tomadas pelos algoritmos como o *Q-learning* e permitindo o envio de comandos para o braço robótico.



(a) Secção de controlo do braço (b) Secção do estado do jogo

Figura 5.4: *Interface* desenhada

5.3 Ferramentas de *Software*

O desenvolvimento do sistema proposto envolveu diversas ferramentas e bibliotecas, de modo a possibilitar e facilitar a implementação das várias funcionalidades apresentadas anteriormente.

5.3.1 Linguagem de Programação

A linguagem de programação escolhida para o desenvolvimento foi Python, uma vez que é conhecida pela sua simplicidade, legibilidade e extensas bibliotecas. Além disso, é uma das linguagens mais proeminentes e utilizadas nos dias de hoje. Com mais de 137 mil bibliotecas, a uma distância de uma linha de código e uma rápida instalação, é considerada uma mais-valia, fatores que tornam esta linguagem ideal para projetos que envolvam IA [170].

5.3.2 Bibliotecas

De entre as múltiplas bibliotecas de Python que poderiam ser implementadas para cada fase do projeto, procurou-se selecionar as que apresentassem um maior número de funcionalidades relacionadas com o presente contexto. Assim, recorreu-se às bibliotecas: Tkinter, PyTorch, Open-Source Computer Vision (OpenCV) e Smbus para aplicação nas fases mais importantes do projeto.

Tkinter

Tkinter é uma biblioteca *open-source* de desenvolvimento de *Graphical User Interface* (GUI), concebida para ser aplicada em códigos Python [171].

A arquitetura desta biblioteca assenta na biblioteca Tk, a biblioteca GUI utilizada por Tcl/Tk e Perl, que por sua vez é implementada em linguagem C.

PyTorch

PyTorch é uma *framework open-source* baseada na biblioteca Torch. Foi desenvolvida na linguagem Python pela equipa do Meta (antigo Facebook) e é das poucas *frameworks* que permite o ajuste de redes neuronais de forma arbitrária sem penalizações ou atrasos [172].

É uma das duas bibliotecas focadas em ML mais populares e tem sido implementada em vários projetos, desde pesquisa de texto, passagem de texto a voz, classificação de imagens e vídeos, entre outros.

OpenCV

OpenCV é, como o próprio nome indica, uma biblioteca *Open-Source*, de uso livre e grátis para fins pessoais e comerciais com o código fonte disponível, destinada à utilização em aplicações de visão computacional [173].

Esta biblioteca ficou disponível a partir de 2006 e atualmente encontra-se integrada em três das linguagens de programação mais conhecidas, Python, Java e

C++. Possui mais de 2500 algoritmos focados em processamento de imagens e vídeos, reconhecimento facial, identificação e rastreamento de objetos e muitas outras aplicações nas áreas da IA.

Smbus

A biblioteca Smbus fornece uma *interface* para um protocolo baseado em I2C, através da qual vários componentes do sistema podem comunicar entre si e com o resto do sistema pelas mesmas duas linhas de comunicação [174].

Esta biblioteca utiliza o barramento I2C, para enviar e receber dados, permitindo que o controlador, por exemplo, comande os movimentos dos atuadores e receba dados de sensores.

5.3.3 Ambientes de Desenvolvimento

A implementação deste sistema nunca seria possível sem o acesso e utilização de alguns ambientes de desenvolvimento focados nas várias etapas deste projeto.

O primeiro ambiente é o Visual Studio Code (VS Code), um editor de código desenvolvido em *open-source* pela Microsoft [175]. Este é um ambiente leve e focado no desenvolvimento e teste de código de programação, contendo vários extensões para as diferentes linguagens de programação, desde Python a C. Por estas razões foi o editor escolhido para o desenvolvimento de todo o *software* do sistema.

O RoboFlow é uma plataforma direcionada para a área da visão computacional em IA com milhares de *datasets* e modelos pré-treinados na área. É conhecida por oferecer uma *interface* simples e fácil de usar para criar, anotar e pré-processar conjuntos de dados de imagens, tendo sido de extrema relevância para obter dados para o treino do modelo de visão computacional [176].

A Kaggle é uma plataforma de ciência de dados da Google designada para a área de ML. Esta permite aos utilizadores explorar e publicar *datasets* ou treinar modelos de IA, oferecendo ambientes virtuais com recursos gratuitos de GPU e ferramentas de colaboração [177]. Possibilitaram o treino eficaz e em reduzido tempo do modelo de visão computacional implementado neste projeto.

Capítulo 6

Implementação

Este capítulo versa a implementação dos diversos módulos que compõem o sistema proposto, apresentando e explicitando o desenvolvimento das várias etapas idealizadas nos capítulos anteriores. Assim, este capítulo abrange a montagem e controlo do braço robótico escolhido, apresentando o *software* criado para o controlador Raspberry PI; abrange também o treino e aplicação dos algoritmos de IA selecionados; por fim, a integração de todos os sistemas.

6.1 Controlo do Braço Robótico

Como se referiu anteriormente, o controlo dos servo-motores deve ser realizado através de sinais PWM enviados pela placa Arm HAT, sendo que esta comunica através de I2C com o Raspberry PI. Tendo isto em mente, pode-se criar a lógica de controlo dos movimentos do braço robótico.

O código criado define três classes principais responsáveis pelo controlo preciso de cada servo-motor do braço robótico.

A primeira, a classe "PCA9685" (apresentada no Anexo A), fornece funções para configuração e comunicação com a placa de servo *driver* em especial com o IC PCA9685, apresentando funções para definição da frequência e controlo dos sinais de PWM gerados pela placa. Para este efeito, é necessário aplicar a biblioteca smbus, que facilita a comunicação entre equipamentos através de I2C.

Em relação à segunda classe, "Servo", esta contém as funções para o controlo de um servo único. No entanto, através da biblioteca Threading, permite-se o controlo

simultâneo de vários servos por *multi-threading*. A inicialização desta classe torna o objeto principal numa referência à classe "PCA9685" apresentada anteriormente. Esta herda as funções de controlo e comunicação por I2C, em que se passa, como parâmetros, o número do canal à qual o servo está ligado, o ângulo pretendido e a condição de paragem.

A função principal desta classe, representada na Figura 6.1, assegura que o valor do ângulo pretendido se encontra no intervalo válido (0°-180°) e calcula o sinal PWM correspondente para o mesmo. Em seguida, ajusta gradualmente o valor PWM enviado para a placa, de modo a realizar pequenos passos e obter, consequentemente, movimentos suaves.

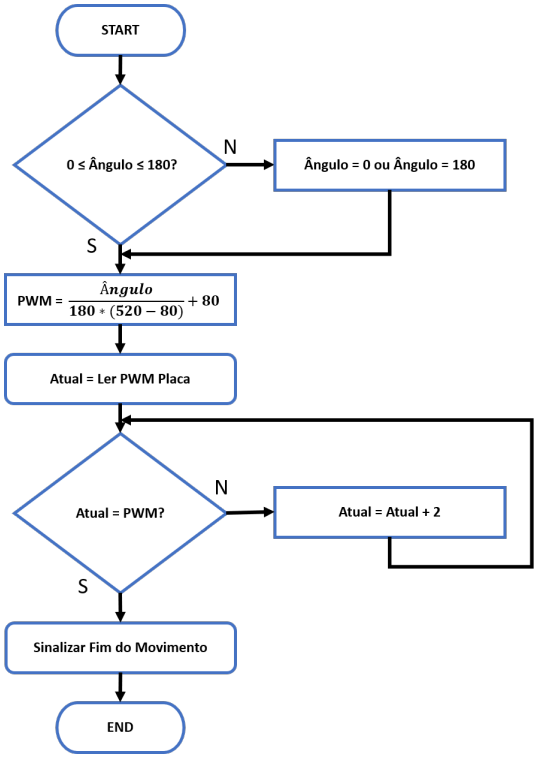


Figura 6.1: Função principal da classe "Servo"

Quando o valor atual de PWM iguala o valor pretendido, a *thread* pode aguardar numa barreira se a condição de paragem estiver ativa. Desta forma, cada *thread* de servo funciona de forma independente, no entanto, implementa-se uma barreira para garantir que nenhuma *thread* de servo começa um novo movimento dessincronizado dos outros servos, ou seja, com um mecanismo de sincronização de barreira bloqueia-se cada *thread* até que todas tenham chamado a função "wait", sinalizando que acabaram o seu movimento.

Por fim, a última classe, "ServoController" fornece uma interface de alto nível para o controlo do braço robótico. Desta forma, esta aplica as classes descritas anteriormente para inicializar um objeto "PCA9685" e definir a frequência PWM inicial

para 50 Hz. Além disso, inicializa todos os servo-motores para uma posição inicial, através da classe "Servo". Por fim, a função principal da classe "ServoController" inicia as *threads* para o controlo simultâneo dos servo-motores para os ângulos desejados.

6.2 Visão Computacional

A componente de visão computacional desempenha um papel fundamental no sistema idealizado. Esta componente é responsável pelo reconhecimento e localização de objetos no espaço de trabalho, para o braço robótico poder efetivamente manipular os objetos reconhecidos e conseguir cumprir o objetivo de jogar o jogo do galo.

Com este intuito, treinou-se e implementou-se um algoritmo de YOLO, um modelo de deteção de objetos conhecido pela sua velocidade e precisão.

O YOLO processa imagens completas em tempo real, numa única passagem pela rede, prevendo simultaneamente as *bounding boxes* e as probabilidades de cada classe.

6.2.1 Fase de Pré-Processamento

O êxito do domínio da visão por computador depende em grande medida da qualidade e da preparação do conjunto de dados que utiliza. Assim, recolheram-se *datasets* com uma vasta gama de diferentes situações e contextos, proporcionando a diversidade necessária para uma robusta deteção de objetos.

No total, reuniram-se 6 conjuntos de dados anotados provenientes do repositório público Roboflow [178, 179, 180, 181, 182], totalizando 4207 imagens. Na Figura 6.2 apresentam-se algumas imagens destes *datasets*.

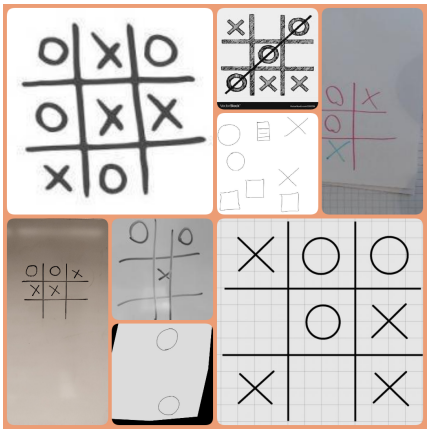


Figura 6.2: Exemplos do *dataset*

Porém, alguns destes conjuntos de dados não continham todas as anotações e classes que o projeto necessitava. Embora muitos dados estivessem pré-anotados, foi necessário inspecionar e corrigir manualmente algumas das etiquetas para garantir que correspondiam aos objetos específicos que o modelo iria detetar.

Deste modo, aplicando as ferramentas que o Roboflow disponibiliza, tentou anotar-se o maior número de imagens quanto possível com as 12 classes que se definiram para este projeto: 1 classe para a área de jogo, 1 classe para o primeiro jogador (X), 1 classe para o segundo jogador (O) e 9 classes para determinar as várias posições da área de jogo.

Apesar de se utilizar as ferramentas do Roboflow, o processo de anotação das várias imagens é demorado. Por isso, criou-se um pequeno *script* para tentar otimizar o processo.

Este *script* foca-se na criação de uma *interface* que facilita a anotação das imagens pelo utilizador, com recurso à biblioteca OpenCV de Python. Numa primeira fase, é capaz de desenhar na imagem as *bounding boxes* já definidas no ficheiro de etiquetas (*labels*). Em seguida, o utilizador pode desenhar uma nova região na imagem na qual existe um objeto pretendido, através da função "selectROI", como ilustrado na Figura 6.3. Por fim, o utilizador decide qual a classe a atribuir à região desenhada e o *script* trata de transformar as coordenadas da região e adicioná-las às anotações da imagem.

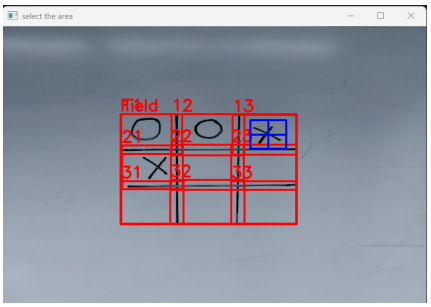


Figura 6.3: Seleção da região da classe pretendida

É de salientar que para a versão do modelo de YOLO que se vai implementar, YOLOv5, as anotações precisam de apresentar uma ordem bem definida:

1. *Index* da Classe;
2. Pixel x do centro da *bounding box*;
3. Pixel y do centro da *bounding box*;
4. Comprimento da *bounding box*;
5. Altura da *bounding box*.

Para além disso, as coordenadas da *bounding box* devem estar normalizadas de 0 a 1, consoante o tamanho da imagem em pixeis.

O final desta fase fica marcado pela remoção de imagens repetidas ou desajustadas ao contexto do projeto. Assim, o *dataset* final apresenta 3000 imagens divididas em 2500 imagens para treino, 375 para validação e 125 para teste.

6.2.2 Fase de Treino do Modelo

O primeiro passo para o treino do modelo é a criação de um ficheiro de configuração que define a estrutura do *dataset*, com os endereços para os dados de cada fase e o número de classes. Este ficheiro "data.yaml" aponta para os diretórios que contêm as imagens de treino, validação e teste e especifica quantas categorias de objeto ou classes o modelo precisa de contabilizar, bem como os nomes pretendidos para cada uma, como se apresenta na Listagem 6.1.

```
path: ./dataset
train:
 - train/images
val:
 - valid/images
test:
 - test/images
nc: 12
names: ['Field', '0', 'X', '11', '12', '13', '21', '
    22', '23', '31', '32', '33']
```

Listagem 6.1: Ficheiro "data.yaml"

Para efetivamente treinar e implementar um modelo de YOLOv5, é importante configurar os ficheiros necessários e definir a estrutura de treino do modelo. A versão mais recente do YOLOv5 está disponível no repositório da Ultralytics disponível no GitHub e contém os ficheiros base para a implementação completa de um modelo de visão computacional.

Após se aceder à pasta do YOLOv5, é necessário instalar as bibliotecas que suportam estes modelos, tais como: matplotlib; numpy; OpenCV; pillow; scipy; torch; tensorflow; entre outras, conforme a Listagem 6.2.

```
pip install -r yolov5/requirements.txt
```

Listagem 6.2: Linha de instalação dos requisitos do YOLO

Dentro da pasta "yolov5/models" observam-se todas as arquiteturas dos modelos pré-construídos que se podem implementar (Figura 6.4), sendo que à medida que

aumenta o tamanho da rede do modelo, aumenta também a sua capacidade de fazer previsões acertadas. No entanto, a velocidade do modelo é inversamente proporcional ao tamanho. Por um lado, YOLOv5X é a maior rede e apresenta uma precisão mais alta que os seus pares (50.7 *Mean Average Precision* (mAP)). Por outro lado, o YOLOv5n é a versão mais pequena e mais rápida a fazer previsões (6.3 ms).

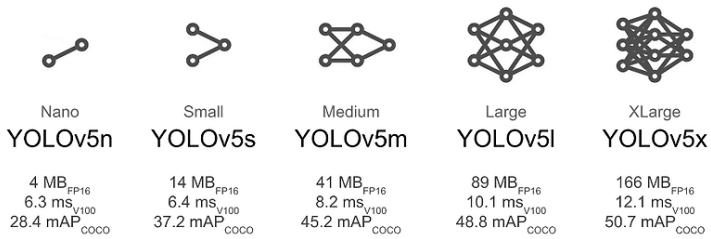


Figura 6.4: Modelos pré-treinados de YOLOv5 [183]

Como passo seguinte no desenvolvimento do projeto, configura-se o ficheiro do modelo do YOLO que se pretende implementar, consoante o número de classes que este tem de identificar. Para isso, altera-se a primeira linha do ficheiro, colocando o número de classes pretendidas a seguir ao parâmetro "nc" (*number of classes*), como se ilustra na Listagem 6.3.

```
1 # YOLOv5 by Ultralytics, GPL-3.0 license
2
3 # Parameters
4 nc: 12  # number of classes
```

Listagem 6.3: Alteração do ficheiro do modelo do YOLO

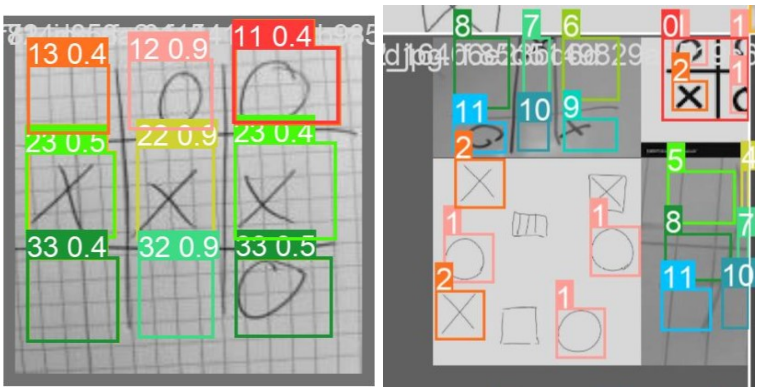
O modelo aplicado neste projeto necessita de identificar e localizar os objetos em tempo real, de ser computacionalmente leve e ao mesmo tempo de ser um modelo com alta precisão. Por isso, optou-se por aplicar um modelo de YOLOv5s.

Em seguida, apesar de não ser recomendado, é necessário alterar o ficheiro de hiperparâmetros que se encontra na pasta "yolov5/data/hyps", de modo a desativar algumas técnicas de *data augmentation* que podem causar problemas no contexto deste projeto. Neste âmbito, editou-se o ficheiro padrão "hyp.scratch-low.yaml" e alteraram-se as linhas "flipud", "fliplr", "mosaic" e "mixup", reduzindo a probabilidade destas técnicas para o mínimo (Listagem 6.4).

```
flipud: 0.0 # image flip up-down (probability)
fliplr: 0.0 # image flip left-right (probability)
mosaic: 0.0 # image mosaic (probability)
mixup: 0.0 # image mixup (probability)
```

Listagem 6.4: Alteração do ficheiro "hyp.scratch-low.yaml"

Estas técnicas desativadas servem para aumentar o número de imagens de treino através do processo de inversão ou de junções e corte de várias imagens. Estas técnicas são comuns e amplamente aplicadas sem nenhum revés. Todavia, para este projeto, caso se inverta uma imagem, a ordem das células do campo de jogo também fica invertida, o que leva as anotações a não possuir a classe certa; ou seja, como se apresenta na Figura 6.5a a célula que correspondia ao canto superior esquerdo (classe '11'), após uma inversão horizontal irá estar colocada na posição do canto superior direito (classe '13') sem haver uma alteração da sua classe. Na mesma medida, a junção e corte de várias imagens fazem com que o tabuleiro de jogo não se apresente completo na imagem, mas seja alvo de recortes (Figura 6.5b). Estes factos levam ao crescimento de classificações erradas, diminuindo a precisão do modelo e à confusão de classes.



(a) Resultados da inversão de imagens

(b) Resultados da junção e corte de imagens

Figura 6.5: Resultados das técnicas desativadas

A última etapa desta fase é o treino do modelo. Posto isto, executa-se o *script* "train.py" fornecido na pasta original. Este *script* permite a passagem dos ficheiros alterados anteriormente e das configurações gerais de treino como, por exemplo, o tamanho das imagens, o tamanho do lote, o número de épocas de treino e os pesos pré-treinados.

O tamanho do lote é o número de amostras processadas antes de atualizar os pesos do modelo. Este parâmetro deve ser o mais elevado, quanto possível, de

processar pela memória da GPU, de modo a melhorar a capacidade de previsão do modelo. O *script* de treino vem preparado para ajustar este parâmetro consoante a memória disponível, basta passar o valor de -1 neste parâmetro.

O número de épocas diz respeito ao número de vezes que o modelo vê o conjunto de dados, influenciando o tempo de treino do modelo. Um maior número de épocas conduz a uma melhor aprendizagem, mas também pode aumentar o risco de *overfitting*. Assim sendo, o valor dos primeiros treinos deve ser elevado de modo a descobrir-se em que época é que se começa a verificar este fenómeno.

Em vez de treinar o modelo do zero, é possível começar a partir de pesos pré-treinados em grandes *datasets*. Esta abordagem acelera o processo e geralmente resulta num bom desempenho, caso os grandes *datasets* se adequem ao contexto, o que não acontece propriamente neste projeto.

Assim, a linha de execução do *script* "train.py" passa como parâmetros as configurações gerais de treino e os ficheiros alterados, designadamente o ficheiro de configuração "data.yaml" criado, o ficheiro do modelo a treinar, "yolov5s.yaml", e o ficheiro dos hiperparâmetros. A linha de execução deverá ter a forma da Listagem 6.5.

```
1 python yolov5/train.py --img 416 --batch -1 --epochs
      300 --data data.yaml --cfg yolov5/models/yolov5s
     .yaml --weights '' --hyp yolov5/data/hyps/hyp.
     scratch-low.yaml
```

Listagem 6.5: Linha de execução do treino do YOLO

Com o treino concluído, os resultados do modelo ficam disponíveis na pasta "yolov5/runs/train/exp/". Esta pasta contém os ficheiros para análise do desempenho do modelo e para implementação. Em relação aos ficheiros para implementação, o YOLOv5 guarda os pesos ajustados que tiveram uma melhor *performance* no ficheiro "best.pt" e os pesos da última época de treino, "last.pt", dentro da pasta "weights".

Relativamente aos resultados do modelo treinado, para analisar as métricas de desempenho, existem duas representações, "results.csv" em que os resultados se apresentam numa tabela para cada época ou "results.png" que apresenta o gráfico de evolução de cada métrica, como: precisão, *Recall* e mAP.

Estes ficheiros incluem, ainda, os valores de perda calculados durante o treino e validação que ajudam a compreender a evolução da aprendizagem do modelo em cada época: *object loss*, que representa o erro na previsão da existência de objetos; *class loss*, que mede o erro na classificação correta dos objetos e *bounding box loss*, que mede o erro de localização dos objetos detetados.

Assim, pelo que se observa na Figura 6.6, os valores de perda do modelo treinado são diminutos, o que implica que o modelo consegue efetivamente detetar os objetos pretendidos.

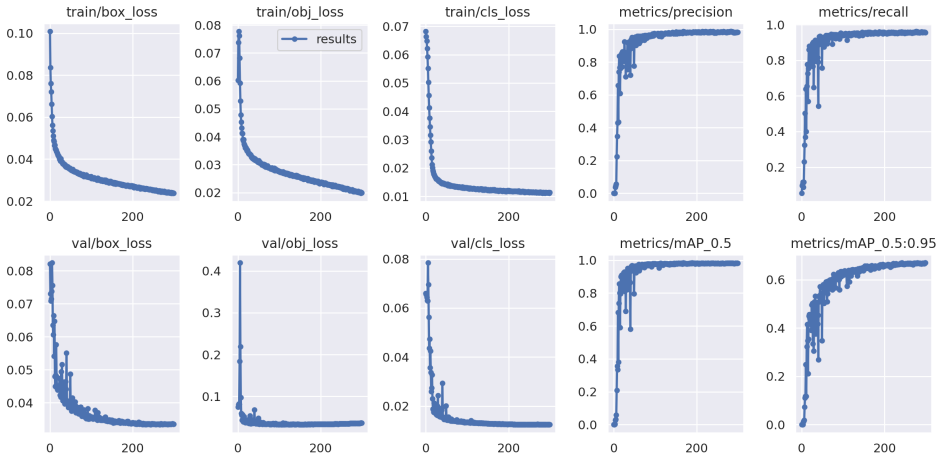


Figura 6.6: Desempenho geral do modelo treinado

Através de uma análise mais profunda dos ficheiros de resultados, pode-se descobrir que o valor de precisão do modelo mais elevado acontece na época 284 em que chega aos 98,7 %, indicando que o modelo acertou a maioria das suas previsões. Esta época é marcada por valores de perda de treino entre 1,1 % para classes e 2,5 % para *bounding boxes*; quanto aos valores para a fase de validação, estes apresentam-se entre os 1,2 % e os 3,5 %, respetivamente. Também apresenta um valor de *recall* de 95,5 %, o que sugere que o modelo é capaz de detetar a grande maioria dos objetos nas imagens.

Os valores de mAP para esta época refletem a precisão geral do modelo na localização dos objetos. O modelo apresenta um mAP_0.5 (mAP para valores de IoU de 50 %) elevado de 98,1 %, contudo, para valores mais altos (mAP_0.5:0.95), apresenta uma percentagem de 66,7 %, indicando que o modelo é bom a detetar objetos para uma tolerância pouco rigorosa de sobreposição da *bounding box* prevista sobre a verdadeira. No entanto, para uma medida mais rigorosa, a qualidade da deteção diminui.

As métricas principais de desempenho encontram-se apresentadas na Tabela 6.1.

Tabela 6.1: Métricas do desempenho do modelo

Época	**Precisão**	***Recall***	**mAP_0.5**	**mAP_0.5:0.95**
284	98,7 %	95,5 %	98,1 %	66,7 %

Por último, interpretando a matriz de confusão (Figura 6.7) consegue-se visualizar que o modelo consegue detetar com elevada precisão a maioria das classes.

Porém, é possível descobrir que o modelo apresenta dificuldades em distinguir a classe da célula central do campo de jogo (’22’) com a de campo (’Field’). Esta confusão é compreensível, uma vez que a célula central pode ser interpretada como um campo mais pequeno.

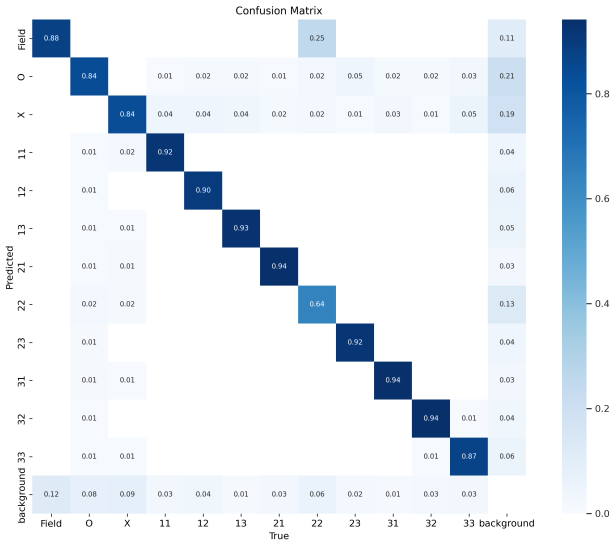


Figura 6.7: Matriz de confusão do modelo treinado

6.2.3 Fase de Implementação

Com o sucesso do treino do modelo YOLO, o passo seguinte foi integrar este módulo no sistema robótico global. O modelo atua como o componente central para a deteção e localização de objetos, permitindo que o braço robótico identifique e interaja com os objetos no ambiente.

O primeiro passo, na integração deste módulo, consiste em carregar o modelo para que este possa inferir sobre as imagens que lhe vão ser enviadas e detetar as várias classes. Através da função “load” da biblioteca Pytorch é fácil carregar o modelo, apenas é necessário indicar qual o tipo de modelo treinado e a localização do ficheiro de pesos ajustados a implementar. A linha de execução toma a forma da Listagem 6.6.

```
1 model = torch.hub.load('./yolov5', 'custom', source=
      'local', path='./yolov5/run/train/exp/best.pt')
```

Listagem 6.6: Integração do modelo de YOLO treinado

Em seguida, recebe-se a imagem e, numa pequena fase de pré-processamento, com recurso à biblioteca de OpenCV, carrega-se e redimensiona-se a imagem a detetar para corresponder ao tamanho para o qual o modelo foi ajustado.

Depois do modelo prever as *bounding boxes* e classificar os objetos na imagem, é necessário interpretar e processar os resultados:

1. Converter os resultados numa lista – os resultados inicialmente apresentam-se num *Pandas Dataframe* e consistem num conjunto de *bounding boxes* em torno dos objetos detetados, e numa pontuação de confiança para cada previsão;
2. Eliminar as previsões abaixo de 50 % de confiança;
3. Desenhar as previsões na imagem original;
4. Guardar as previsões num dicionário, que mapeie a classe às coordenadas na imagem.

Por fim, este módulo envia o dicionário das previsões para o módulo geral de controlo, que assegura que a aplicação e a tarefa são realizadas.

6.3 Tomada de Decisão

O algoritmo de decisão é responsável por determinar as ações a tomar com base no estado atual do ambiente. Desenvolveram-se três algoritmos com processos de decisão distintos: um algoritmo de decisão aleatória, o algoritmo *MiniMax* que avalia todas as possibilidades e escolhe a que produz uma recompensa de alta qualidade e o *Q-learning*, um algoritmo de RL, que permite a aprendizagem das ações, que produzem as recompensas mais elevadas, sem necessitar de um modelo-tipo do ambiente. Assim, aplicaram-se estes algoritmos para a tomada de decisão quanto à jogada a realizar no jogo do galo.

Inicialmente, a classe "TicTacToeGame" é responsável por criar e controlar o jogo do galo que serve de base ao projeto e ambiente de treino do modelo de RL. Esta classe constrói o tabuleiro de jogo, utilizando uma lista de listas para conter o estado do mesmo. Cada elemento destas listas, inicialmente, é definido como um espaço (' ') para representar uma célula vazia e vai sendo povoado consoante as jogadas efetuadas.

O método principal desta classe, ilustrado na Figura 6.8, permite que um jogador coloque o seu símbolo ('X' ou 'O') no tabuleiro, na linha e coluna especificadas. Esta função, "make_move", em primeiro lugar, verifica se a célula selecionada está vazia (' '). Se a célula estiver vazia, a jogada é validada e a função atualiza o tabuleiro com o símbolo do jogador atual. Em seguida, altera o jogador atual para permitir outra jogada ('O' se o jogador atual for 'X', e vice-versa). Por outro lado, se a jogada for considerada inválida, por a célula já se encontrar ocupada, a função imprime uma

mensagem de erro e não atualiza o tabuleiro de jogo, obrigando a tentar novamente.

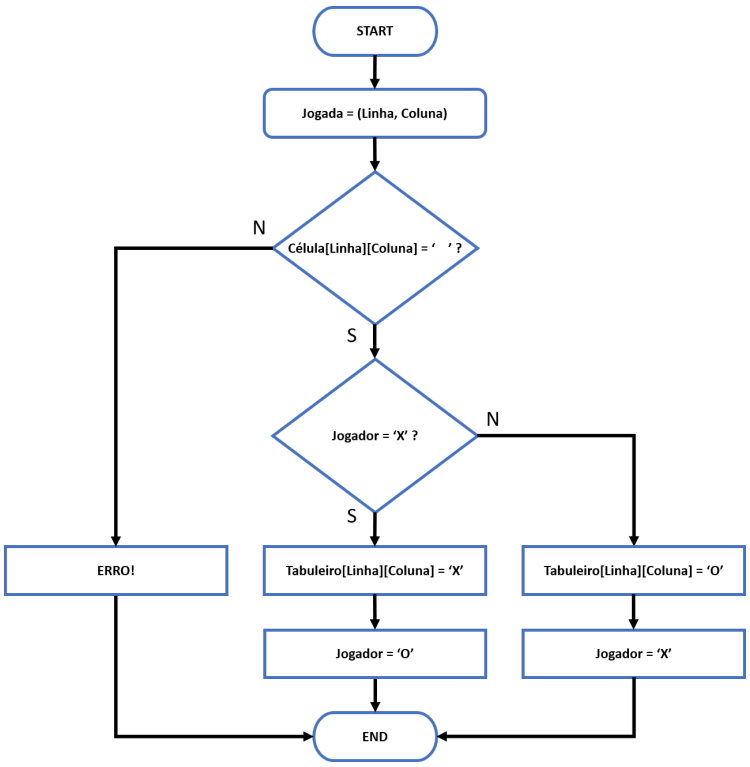


Figura 6.8: Fluxograma da função "make_move"

Após cada jogada, é necessário utilizar o método "check_winner" para determinar se o jogo acabou e se há um vencedor ou se o jogo terminou num empate. Para tal, verificam-se todas as possíveis combinações vencedoras (linhas, colunas ou diagonais completas com símbolos iguais). Se alguma destas condições for satisfeita, o jogo é marcado como terminado, e recupera-se o símbolo do jogador vencedor ('X' ou 'O'). Caso nenhuma condição seja satisfeita, embora o tabuleiro de jogo esteja completo, o método devolve um estado de empate, 'Draw'. Caso o jogo não tenha terminado, a função "retorna" 'None', sinalizando que o jogo pode continuar, passando a vez para o jogador seguinte.

Em relação aos algoritmos dos jogadores, o algoritmo aleatório é o mais simples. Este tipo de jogador seleciona uma ação aleatória a partir do conjunto de jogadas disponíveis. Não possui qualquer estratégia para a escolha dos movimentos e não tem em conta o estado do jogo.

O algoritmo *MiniMax* é uma abordagem mais sofisticada. O algoritmo procura minimizar as recompensas do adversário, enquanto tenta maximizar as recompensas do próprio jogador. Para isto, assume que o adversário está a aplicar o seu melhor movimento e procura minimizar as suas hipóteses de ganhar.

Assim, este algoritmo envolve a exploração recursiva, em forma de árvore, de todas as jogadas futuras possíveis, procurando continuar a maximizar a pontuação do jogador e minimizar a pontuação do adversário. O algoritmo simula o estado do jogo, em que um novo movimento implica um novo ramo da árvore; altera os turnos de cada jogador e avalia o resultado de todas as jogadas possíveis até que o jogo atinja um estado terminal.

Neste estado, atribuem-se as recompensas à jogada que levou a este resultado. Em caso de:

- **Vitória** – O algoritmo dá uma pontuação positiva, caso a jogada pertença ao jogador para o qual se está a maximizar, ou uma pontuação negativa para o adversário ou jogador a que se pretende minimizar as recompensas;
- **Empate** – O algoritmo devolve uma pontuação nula, refletindo o facto de nenhum jogador ter vantagem.

Estes valores são passados de volta para a árvore ou ramo que iniciou a jogada e utilizados para avaliar qual a jogada que proporciona o melhor resultado.

Desta forma, em cada nível da árvore de jogo, se for a vez do jogador ao qual se pretende maximizar as recompensas, escolhe-se a jogada com a pontuação mais alta. Pelo contrário, se for a vez do jogador que está a minimizar, o algoritmo escolhe a jogada com a pontuação mais baixa.

Este padrão continua recursivamente até o algoritmo ter explorado todas as jogadas possíveis e determinado a melhor jogada a realizar.

Como se demonstra na Figura 6.9, o algoritmo de *MiniMax* decide que a melhor jogada a fazer para o jogador 'X' é no meio da coluna da direita, uma vez que as melhores decisões desse ramo terminam numa situação em que na pior das hipóteses, o jogo termina empatado.

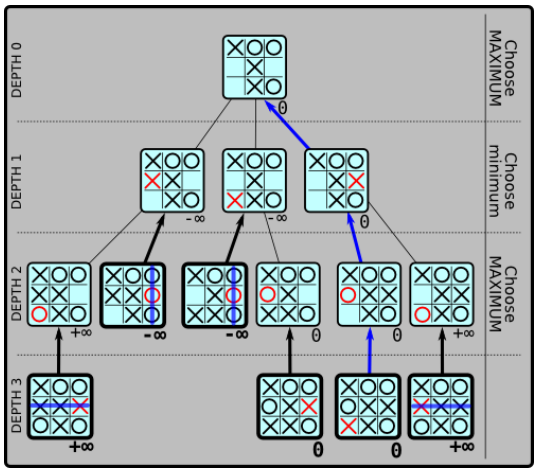


Figura 6.9: Exemplo da lógica do algoritmo de *MiniMax* [184]

Relativamente à classe de implementação do algoritmo de *Q-Learning*, esta necessita de três parâmetros-chave que controlam o processo de aprendizagem do agente: a taxa de aprendizagem do modelo, o fator de desconto e o valor de ϵ, para aplicar a estratégia ϵ-*Greedy Policy*. Uma das primeiras funções desta classe é criar a "Q-Table" como um dicionário vazio, que é posteriormente preenchida com os valores "Q" para cada par estado-ação.

Tendo em conta o modo de funcionamento deste algoritmo, descrito na Subsecção 3.3.8, desenvolveram-se várias funções, maioritariamente, de suporte da função principal "choose_move" que decide qual a melhor jogada para o estado presente. Estas funções incluem:

- **"get_gameState"** – converte a lista do tabuleiro de jogo numa variável dupla, que serve como estado para a "Q-Table".
- **"get_q_value"** – obtém o valor "Q" para o par estado-ação. Se o estado não se encontrar na tabela, inicializa o valor "Q" a 0.
- **"choose_action"** – decide se o agente vai explorar ou aproveitar os valores conhecidos:
  - ***Exploration*** – o agente escolhe uma ação aleatória entre os movimentos disponíveis (células vazias);
  - ***Exploitation*** – o agente escolhe a ação entre os movimentos disponíveis que apresente o valor "Q" mais elevado para o estado atual, segundo a função "get_q_value";

Assim, a função "choose_move" é responsável por aplicar estas funções de suporte para gerar as jogadas disponíveis e determinar a melhor ação. Esta função principal trata, depois, de realizar a jogada no tabuleiro de jogo com recurso à função "make_move" da classe "TicTacToeGame", apresentada anteriormente.

Após fazer a jogada, verifica-se se o jogo atingiu um estado terminal (vitória, derrota ou empate) e atribui-se uma recompensa em conformidade, positiva para uma vitória, negativa para uma derrota e uma recompensa ligeiramente positiva para um empate.

Posteriormente, a função simula as melhores respostas do adversário e penaliza o agente, se for provável que o adversário ganhe na jogada seguinte, incentivando o agente a bloquear o adversário de forma estratégica.

Por fim, esta função chama a função "learn", que é o núcleo da aprendizagem do algoritmo, que atualiza os valores "Q" da tabela segundo a equação característica deste modelo, a regra de diferença temporal, apresentada na Subsecção 3.3.8.

De modo a implementar um algoritmo capaz de ganhar o jogo do galo, treinou-se este mesmo durante um total de 310000 jogos contra os vários oponentes desenvolvidos, utilizando um valor decrescente de ϵ contra cada adversário. No fim do treino, obtiveram-se os resultados demonstrados na Tabela 6.2.

Tabela 6.2: Resultados treino de *Q-Learning*

Adversário	**Vitória X**	**Empate X**	**Vitória O**	**Empate O**
Aleatório	77,38 %	13,19 %	32,73 %	13,57 %
Q-Learning Novo	56,36 %	19,56 %	32,28 %	15,42 %
Q-Learning Treinado	55,90 %	19,95 %	21,19 %	19,59 %
Minimax	0,00 %	52,19 %	0,00 %	1,16 %

Para validar o modelo, reduziu-se o valor de exploração para o mínimo e testou-se novamente o algoritmo contra os seus adversários. Os resultados estão demonstrados na Tabela 6.3, onde se vê claramente que o modelo reduziu a sua taxa de derrota, procurando efetivamente posições em que ganha ou, na pior das hipóteses, empata o jogo.

Tabela 6.3: Resultados finais de *Q-Learning*

Adversário	**Vitória X**	**Empate X**	**Vitória O**	**Empate O**
Aleatório	91,0 %	6,0 %	45,0 %	14,0 %
Q-Learning Novo	84,0 %	12,0 %	37,0 %	8,0 %
Q-Learning Treinado	72,0 %	23,0 %	19,0 %	15,0 %
Minimax	0,0 %	99,0 %	0,0 %	2,0 %

Como se pode observar, o algoritmo de *Q-Learning* progrediu substancialmente e encontra-se pronto a aplicar no projeto.

6.4 *Interface* Gráfica

A *interface* gráfica é responsável por permitir o controlo principal do sistema e a interação com os vários módulos. Esta foi desenhada de modo a facilitar o controlo e teste do braço robótico e a realização da tarefa proposta de jogar o jogo do galo.

Foi desenvolvida a partir da biblioteca Python, Tkinter, e está dividida em vários separadores, cada um deles centrado nas diferentes funcionalidades essenciais para o controlo do braço robótico. As duas primeiras demonstram e permitem o controlo manual do braço, aplicando a cinemática do sistema. O terceiro separador configura os algoritmos e inicia a lógica do jogo que vai decorrendo durante os separadores seguintes. Em quarto lugar, detetam-se os objetos utilizando o modelo YOLOv5 treinado. Por fim, o último separador apresenta o estado do jogo.

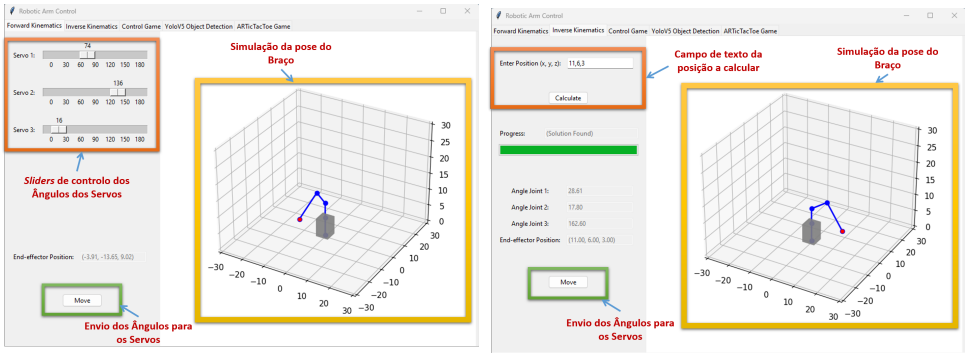
Esta interface permite transições suaves entre os diferentes módulos, oferecendo aos utilizadores uma forma clara e intuitiva de interagir com o sistema.

6.4.1 Controlo do Braço Robótico

Os dois primeiros separadores, apresentados na Figura 6.10, são, como referido anteriormente, responsáveis por permitir introduzir os ângulos para os servo-motores em *sliders* ou introduzir as coordenadas do ponto final num campo de texto e comandar os movimentos do braço em tempo real. Por detrás desta *interface*, faz-se uso dos *inputs* com o intuito de calcular a cinemática direta e inversa do sistema.

Ao mesmo tempo, atualiza-se em tempo real um gráfico em três dimensões, segundo os ângulos inseridos ou calculados, com o intuito de demonstrar o comportamento do braço robótico para os valores dados.

Depois de calcular a cinemática, a *interface* permite, através de um botão, o envio dos ângulos para os servos do robô, utilizando a classe de controlo apresentada anteriormente. O sistema assegura um movimento suave e coordenado, ajustando com precisão cada servo para atingir as posições ou os ângulos calculados.



(a) Separador de controlo do braço por ângulos

(b) Separador de controlo do braço por posição

Figura 6.10: *Interface* de controlo manual do braço robótico

6.4.2 Parametrização do Sistema

O terceiro separador (Figura 6.11) permite configurar os parâmetros gerais do jogo como o tipo de jogadores e os hiperparâmetros destes e iniciar o jogo do galo.

Assim, este separador inclui duas gamas de 4 botões de opção para o utilizador decidir qual é o tipo do jogador 1 e o tipo do jogador 2, determinando o algoritmo de tomada de decisão. Estes tipos de jogador incluem:

- **Jogador Humano**;
- **Jogador Aleatório**;
- **Jogador *Minimax***;

- **Jogador *Q-Learning*.**

De modo a permitir parametrizar os hiperparâmetros de cada jogador, entradas de texto permitem ao utilizador escrever o número desejado para cada valor, obtendo-se uma configuração flexível do ambiente do jogo. Estes parâmetros incluem a taxa de exploração (ϵ), a taxa de aprendizagem e fator de desconto para o algoritmo de *Q-Learning* e a profundidade máxima de exploração para o algoritmo de *Minimax*.

No final, um botão permite ao utilizador começar a jogar o jogo do galo, iniciando a lógica principal de controlo desta tarefa em *background*.

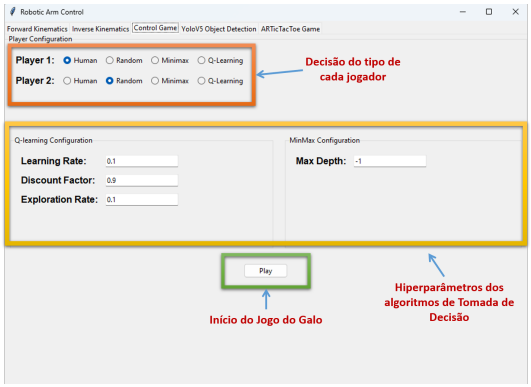


Figura 6.11: *Interface* de parametrização do sistema

Assim, este separador simplifica o processo de personalização do comportamento de cada jogador, alterando os cenários da tarefa.

6.4.3 Deteção de Objetos

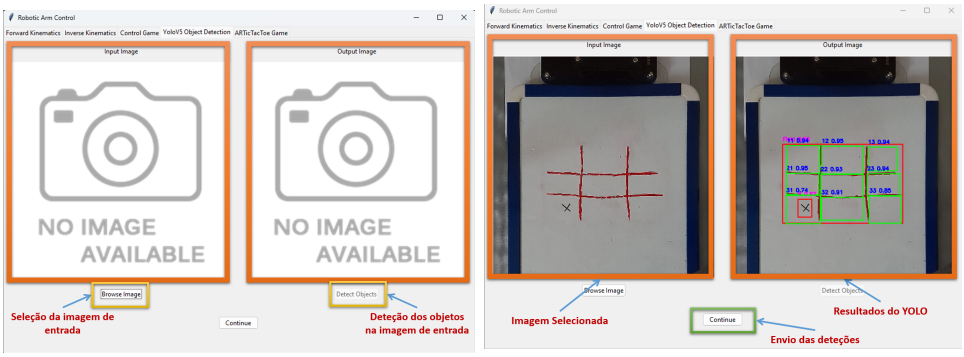
Este separador, apresentado na Figura 6.12, dedicado a interagir com o modelo treinado de YOLOv5, apresenta dois botões, o primeiro para procurar e selecionar a imagem de entrada nos ficheiros da máquina e o segundo para aplicar o modelo e detetar os objetos nesta imagem.

Estas imagens podem ser posteriormente visualizadas nos dois ecrãs, presentes no centro deste separador, focados na imagem de entrada e nos resultados da deteção.

Após a deteção, um terceiro botão permite o envio dos objetos identificados e do estado do tabuleiro de jogo para o módulo seguinte, que processa esta informação de modo a prosseguir com a lógica do jogo.

6.4.4 Estado do Jogo

O separador final do tabuleiro de jogo fornece uma representação visual do estado atual do jogo, juntamente com informações sobre cada jogada realizada por cada jogador e uma breve justificação para esta (Figura 6.13).



(a) Fase inicial da deteção de objetos

(b) Apresentação dos resultados da deteção de objetos

Figura 6.12: *Interface* de deteção de objetos

Assim o elemento principal é a matriz representativa do estado do jogo, sendo atualizada após cada jogada, mostrando assim os quadrados vazios e os ocupados por "X"s e "O"s.

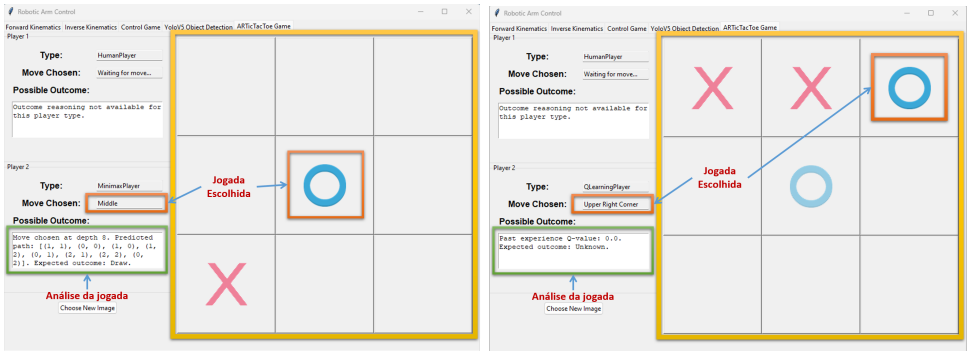
Os restantes elementos deste separador apresentam o tipo de jogador selecionado, a jogada que ele efetuou e uma breve justificação ou raciocínio por detrás desta, consoante o processo de tomada de decisão:

- **Jogador Humano** – Mensagem predefinida, "*Outcome reasoning not available for this player type.*"
- **Jogador Aleatório** – Mensagem predefinida, "*Move chosen randomly. No outcome predicted.*"
- **Jogador *Minimax*** – Mensagem dinâmica que demonstra a profundidade analisada, os melhores movimentos e respostas para o estado atual e o resultado esperado para a linha de movimentos analisada.
- **Jogador *Q-Learning*** – Mensagem dinâmica que demonstra o "Q-Value" para o estado atual e a previsão imediata do jogo.

Isto permite que os utilizadores compreendam porque é que certas jogadas foram feitas e como é que o algoritmo chegou ao movimento jogado, dando uma visão clara da progressão do jogo, do desempenho da IA e do processo de tomada de decisões em tempo real.

6.5 *Flow* do Jogo

O *backend* deste sistema funciona através de uma série estruturada de passos, assegurando transições suaves entre as diferentes fases do jogo. Desde a configuração inicial, passando por cada turno, até aos resultados finais, a lógica de controlo assegura que todas as eventualidades estão consideradas. Os principais componentes



(a) Jogo contra o *MiniMax* (b) Jogo contra o *Q-Learning*

Figura 6.13: *Interface* do estado do jogo

desta incluem a configuração do jogo, a mudança dos turnos dos jogadores, a integração das deteções YOLO e o controlo dos movimentos físicos do robô.

Quando se pressiona o botão para jogar no terceiro separador da *interface* gráfica, a função "start_game" é acionada. Nesta função, o sistema lê os dados configurados pelo utilizador que especificam o tipo de jogadores e os hiperparâmetros dos algoritmos e aplica estes parâmetros para configurar o ambiente e preparar o jogo para arrancar.

Depois do jogo estar configurado, o *backend* inicia o *loop* principal, ilustrado na Figura 6.14, alternando turnos entre o jogador 1 e o jogador 2 e atualizando o estado do tabuleiro de jogo, após cada jogada. Ao mesmo tempo, o sistema verifica se existe um vencedor ou uma condição de empate, atualizando o fluxo do jogo em conformidade.

Para um jogador humano, o *loop* principal espera que o modelo de YOLO realize as suas deteções e lhe envie o dicionário com as deteções através do botão "continue" no quarto separador. A partir das deteções o sistema calcula o valor de IoU com o intuito de descobrir as células que se encontram ocupadas e as posições de cada símbolo. O processamento das deteções é crucial, uma vez que pode acontecer que o modelo detete objetos repetidos nas mesmas coordenadas, como se apresenta na Figura 6.15. Assim, é necessário remover as deteções repetidas no mesmo espaço e as deteções repetidas de certas classes que apresentem um grau de confiança inferior (não podem existir 2 instâncias da classe "Field" na mesma imagem ou de alguma das células), garantindo que o tabuleiro de jogo no sistema não encontra nenhum erro e corresponde efetivamente ao estado físico do tabuleiro.

Em seguida, junta-se esta informação às coordenadas de cada deteção num dicionário novo, o mapa de deteções, utilizado para atualizar o estado do jogo para permitir que o sistema consiga, no turno de um jogador não humano, controlar o braço robótico para as posições detetadas e jogar na vida real.

Pelo contrário, para os jogadores não humanos, o sistema determina imediatamente a próxima jogada com base no algoritmo de cada jogador para o estado do

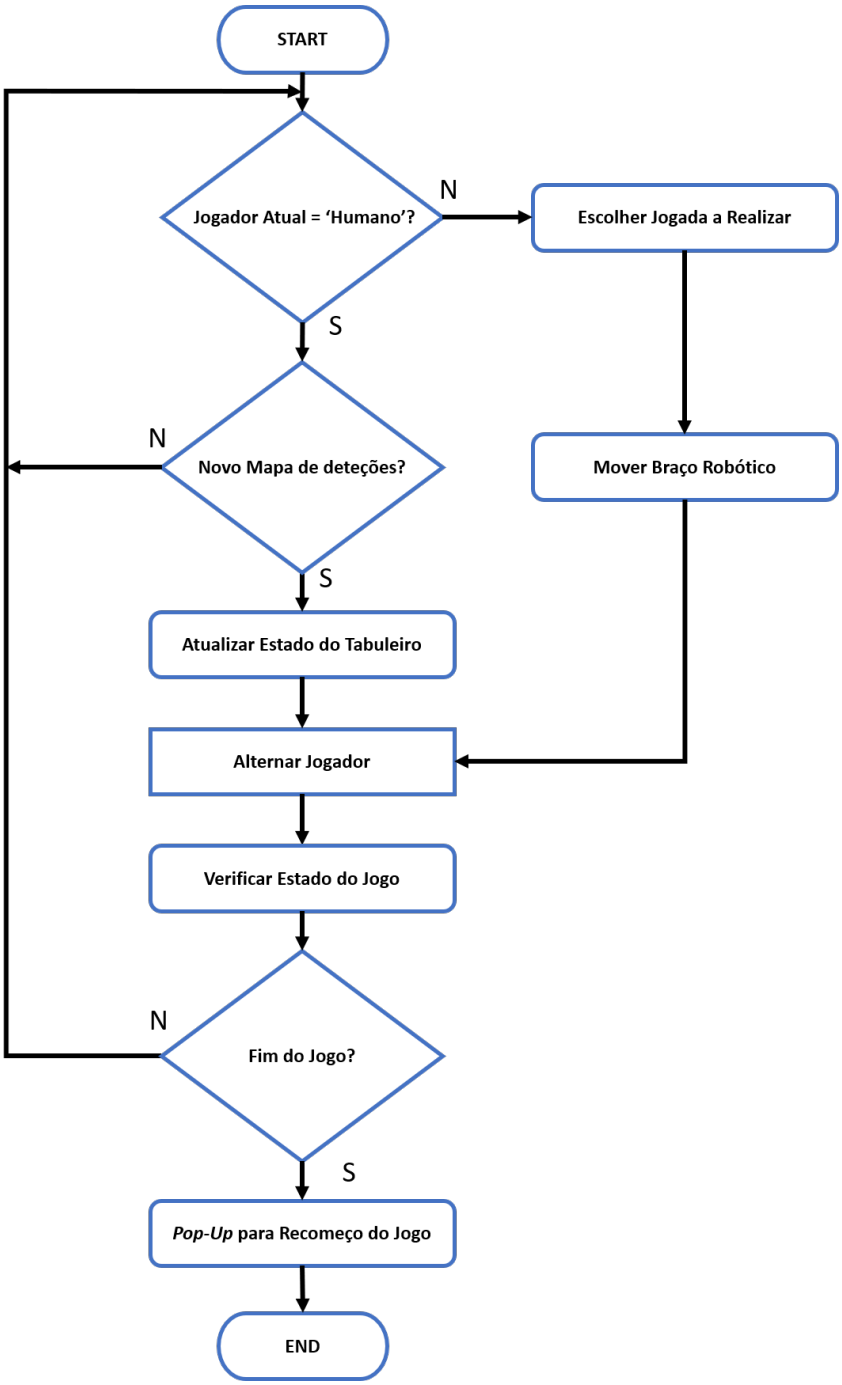


Figura 6.14: Lógica do *loop* principal do jogo

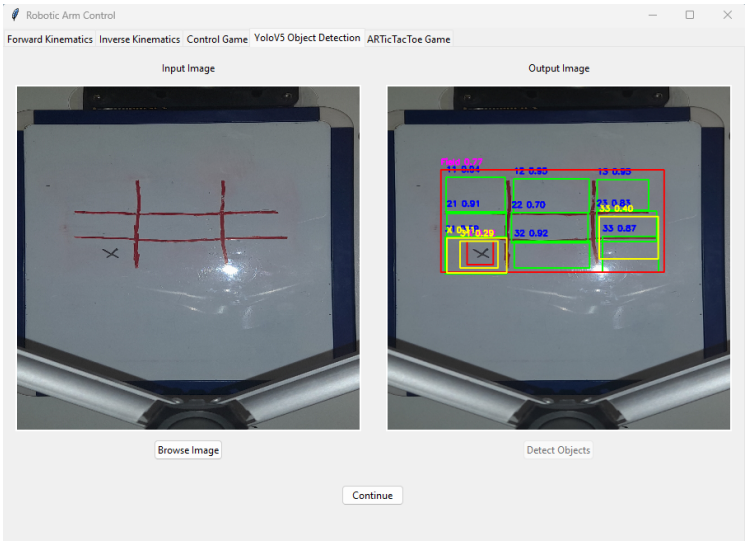


Figura 6.15: Resultados do YOLO com deteções repetidas

tabuleiro e a lógica de movimento do robô entra em ação.

Os movimentos do braço robótico dividem-se em três etapas:

1. Pairar sobre o objeto detetado – o atuador final move-se para o centro X e Y da deteção com um Z elevado predefinido;

2. Baixar o atuador final – o atuador final permanece sobre centro X e Y da deteção, mas diminui o Z para marcar a célula onde vai jogar;

3. Voltar à base – o atuador final volta à posição predefinida inicial.

Os ângulos que permitem os movimentos de pairar e de baixar o atuador final são calculados segundo as coordenadas de cada deteção e a cinemática inversa do sistema. Assim, tenta-se assegurar que o atuador final se encontre sobre o centro da deteção para estes movimentos. Para isto é necessário converter as coordenadas de cada deteção de pixéis da imagem para centímetros.

Através da função "map_pixel_to_real_life" calcula-se, em primeiro lugar, o centro da *bounding box* do objeto em pixéis e de seguida multiplica-se pela escala da imagem, convertendo 640x640 px para 25,5x33,5 cm.

Posteriormente, aplicando a classe "Servo", enviam-se as ordens dos ângulos calculados e as *threads* iniciam o movimento do braço robótico para a posição definida.

À medida que o jogo avança, o *loop* de *backend* verifica constantemente se o jogo apresenta uma condição terminal e, quando esta condição é satisfeita, o jogo termina. Decide-se o vencedor e um *pop-up* permite recomeçar um novo jogo, com a reposição do tabuleiro e a preparação do sistema para a próxima ronda.

Capítulo 7

Principais Resultados

Neste capítulo, apresentam-se os principais resultados obtidos, exemplificando o processo do sistema durante um jogo de teste realizado entre um jogador humano e o jogador de *Q-Learning* treinado. Posteriormente explica-se um pouco dos problemas e dificuldades na execução do projeto e apresentam-se algumas das soluções encontradas.

7.1 Resultados do Sistema

Durante os testes finais, o sistema obteve resultados bastante positivos, conseguindo cumprir o objetivo proposto e jogar efetivamente o jogo do galo. Nas Figuras 7.1 a 7.10 apresentam-se os resultados de um jogo de teste entre um Jogador Humano e o Jogador *Q-Learning*.

- Início do jogo:

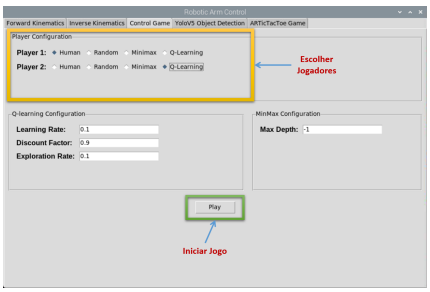
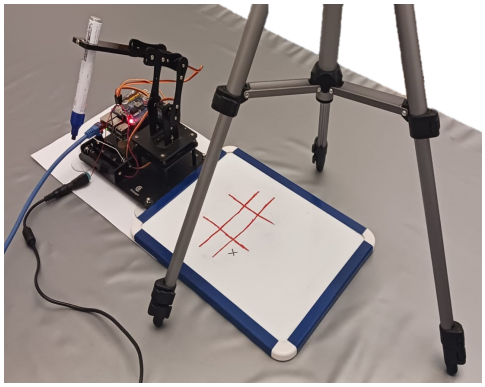
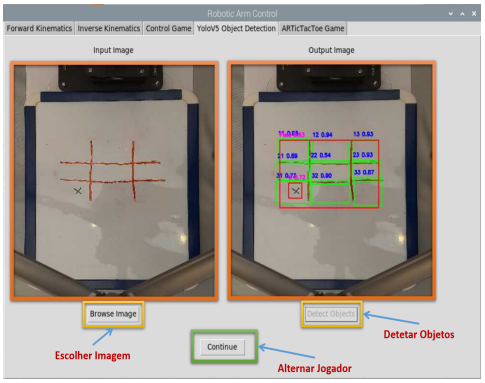


Figura 7.1: Início do jogo

- Primeiro turno:
  - Jogador 1:



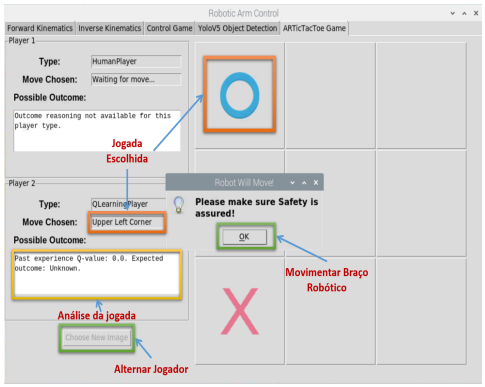
(a) Jogada do jogador 1



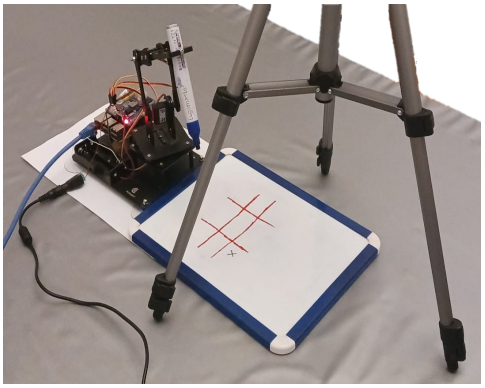
(b) Análise do estado do tabuleiro

Figura 7.2: Primeiro turno do jogador humano

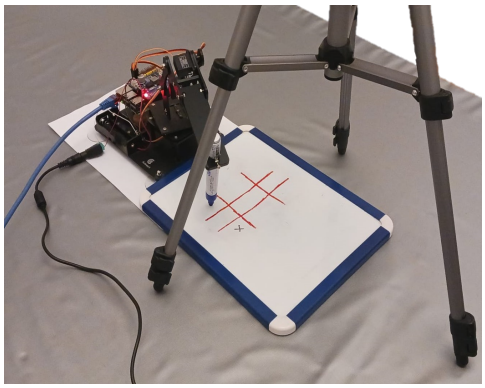
  - Jogador 2:



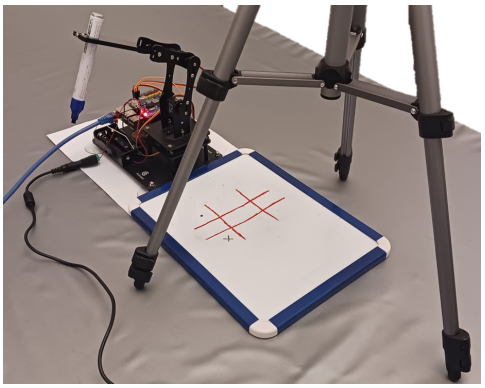
(a) Escolha da jogada do jogador 2



(b) Primeiro movimento do braço robótico



(c) Segundo movimento do braço robótico

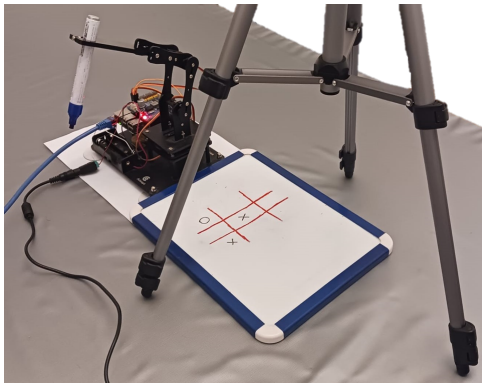


(d) Terceiro movimento do braço robótico

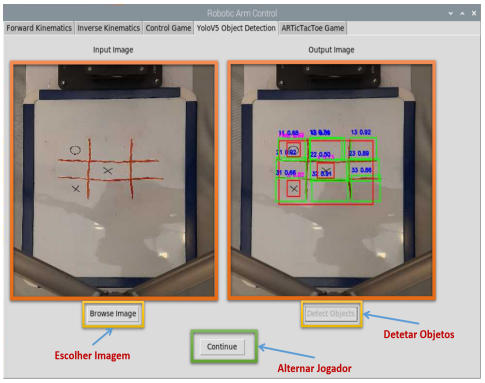
Figura 7.3: Primeiro turno do jogador *Q-Learning*

- Segundo turno:

– Jogador 1:



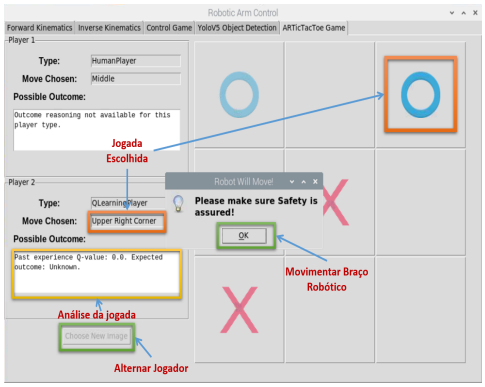
(a) Jogada do jogador 1



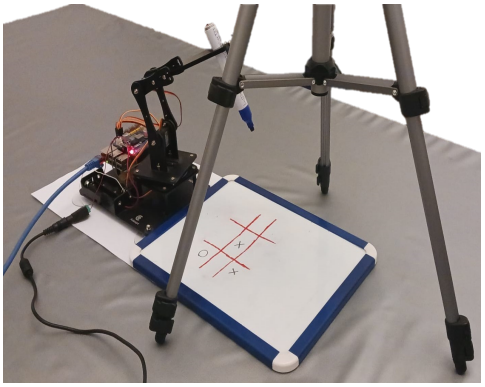
(b) Análise do estado do tabuleiro

Figura 7.4: Segundo turno do jogador humano

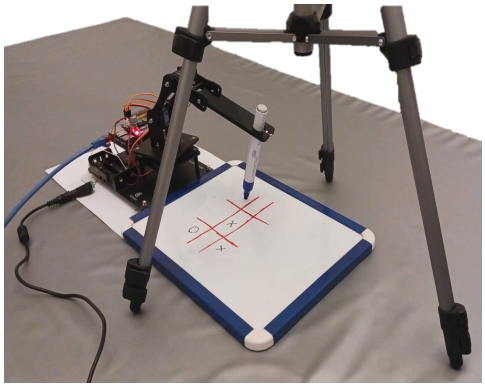
– Jogador 2:



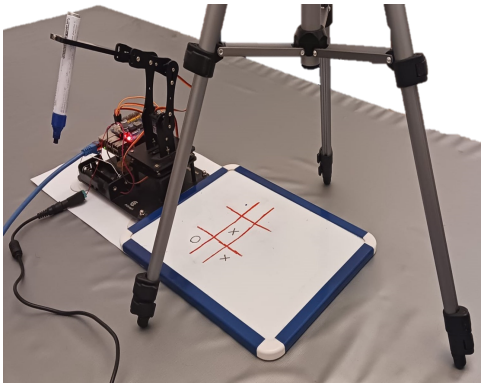
(a) Escolha da jogada do jogador 2



(b) Primeiro movimento do braço robótico



(c) Segundo movimento do braço robótico

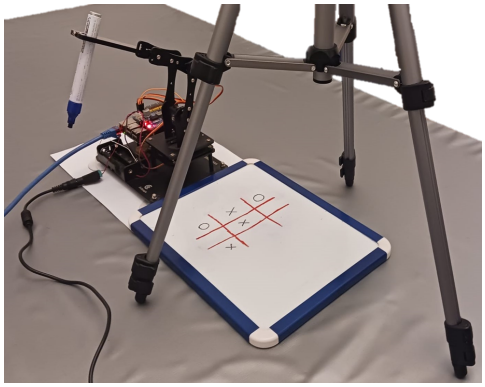


(d) Terceiro movimento do braço robótico

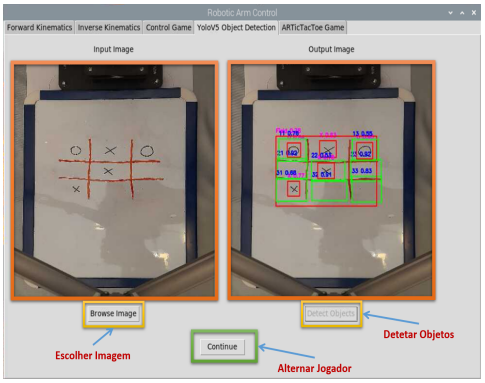
Figura 7.5: Segundo turno do jogador *Q-Learning*

- Terceiro turno:

– Jogador 1:



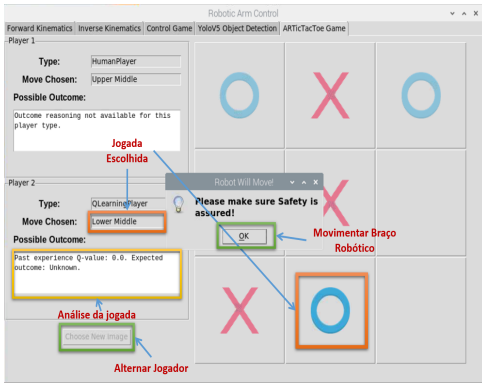
(a) Jogada do jogador 1



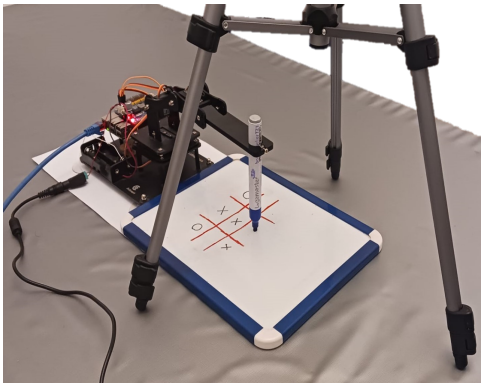
(b) Análise do estado do tabuleiro

Figura 7.6: Terceiro turno do jogador humano

– Jogador 2:



(a) Escolha da jogada do jogador 2

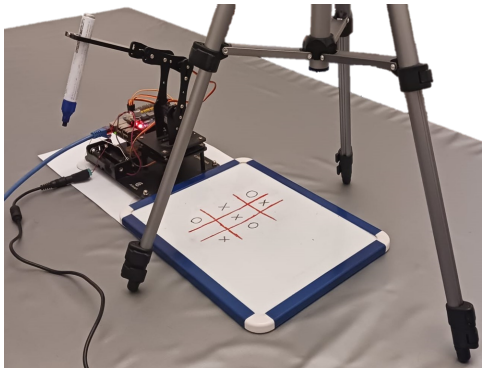


(b) Segundo movimento do braço robótico

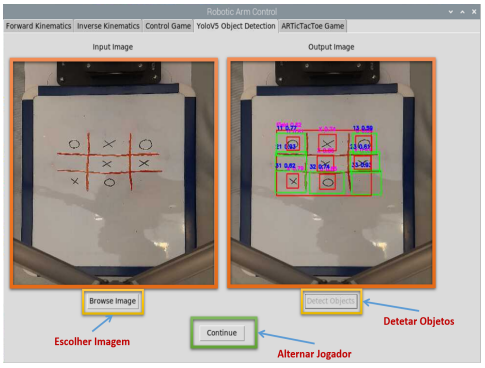
Figura 7.7: Terceiro turno do jogador *Q-Learning*

- Quarto turno:

– Jogador 1:



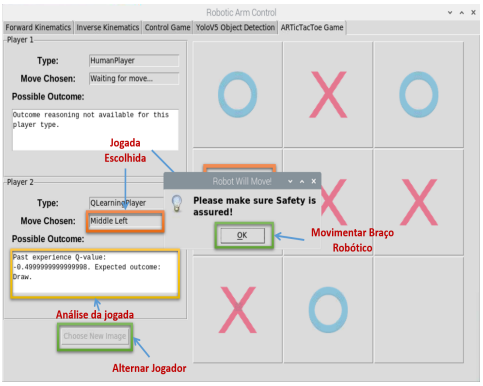
(a) Jogada do jogador 1



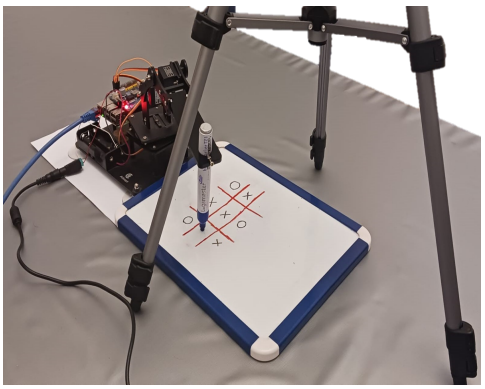
(b) Análise do estado do tabuleiro

Figura 7.8: Quarto turno do jogador humano

– Jogador 2:



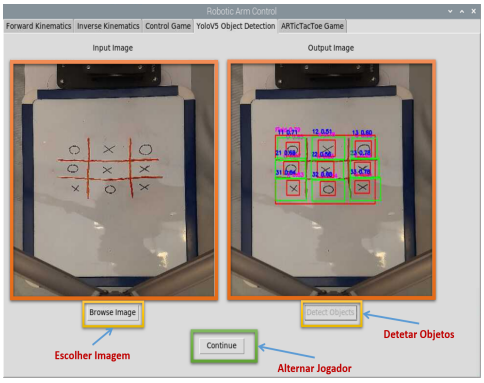
(a) Escolha da jogada do jogador 2



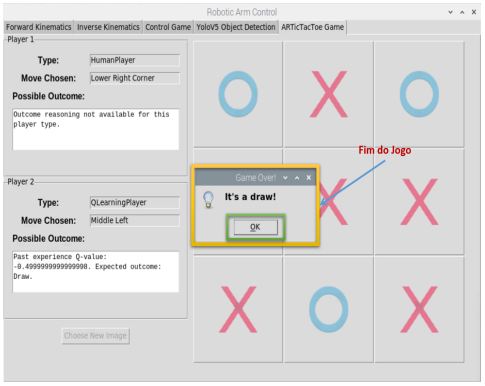
(b) Segundo movimento do braço robótico

Figura 7.9: Quarto turno do jogador *Q-Learning*

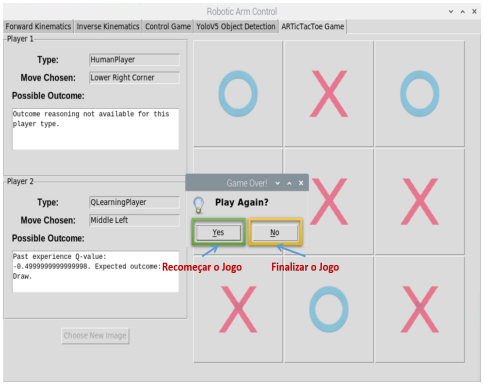
- Fim do jogo:



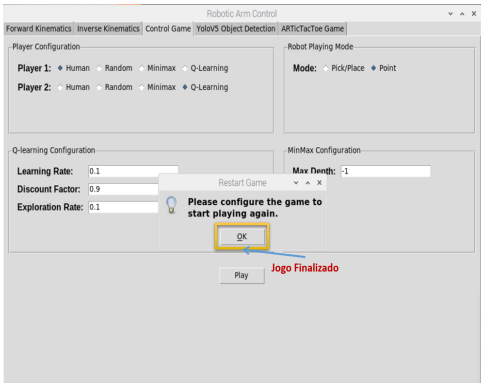
(a) Estado do tabuleiro



(b) Resultado do jogo



(c) *Pop-Up* de recomeço do jogo



(d) Finalização do jogo

Figura 7.10: Último turno e final do jogo

A eficácia do algoritmo de *Q-learning* na tomada de decisões e sua capacidade de adaptação ao contexto da tarefa, levou o sistema a tentar não perder no jogo,

procurando sempre o resultado que lhe desse uma maior recompensa, uma vitória ou, no pior dos casos, um empate.

Na mesma medida, o módulo de visão computacional, o modelo de YOLO treinado, permitiu a interação do braço robótico com o ambiente envolvente e a execução física das jogadas, produzindo erros de posicionamento mínimos e com desvios dentro dos limites aceitáveis para a tarefa (até 1 cm).

No entanto, até se conseguir chegar a estes resultados, foi necessário adaptar, por diversas vezes, a implementação primordial deste sistema, de modo a resolver os vários problemas que apareceram, desde os resultados menos precisos do modelo YOLO, passando pela elevada taxa de derrota do algoritmo de *Q-Learning*, até aos problemas mecânicos do braço robótico.

7.2 Resolução de Problemas

Com o intuito de resolver o problema dos resultados do modelo de YOLO foi necessário voltar a treinar várias vezes o modelo com parâmetros diferentes até se obter deteções precisas consistentes. O método de treino descrito na Subsecção 6.2.2 é o produto final das várias iterações feitas até se descobrir as verdadeiras causas do problema.

A primeira causa eram as anotações incompletas dos conjuntos de dados utilizados, o que implicou inspecionar e corrigir manualmente algumas das etiquetas para garantir que correspondiam aos objetos específicos que o modelo iria detetar.

Nos primeiros treinos do modelo também não se considerava a alteração do ficheiro de configuração dos hiperparâmetros de treino. Estes parâmetros ao aplicarem transformações nas imagens de treino estavam a provocar a confusão de classes. O facto de aplicar inversões nas imagens, por exemplo, invertia também a ordem das células do campo de jogo, o que leva as anotações a não possuir a classe certa, ou seja, a célula que correspondia ao canto superior esquerdo (classe '11'), após uma inversão horizontal estava colocada na posição do canto superior direito (classe '13') sem haver uma alteração da sua classe. Na mesma medida, as junções e cortes de imagens fazia com que o tabuleiro de jogo não se apresentasse completo na imagem, mas fosse alvo de recortes.

Estes factos levavam ao crescimento de classificações erradas, diminuindo a precisão do modelo e à confusão de classes, daí ser necessário ir contra as recomendações da equipa desenvolvedora do YOLO e editar o ficheiro dos hiperparâmetros.

A partir do momento em que se solucionaram estas causas, a precisão do modelo de YOLO aumentou drasticamente, não confundindo classes como anteriormente, demonstrando que o problema estava resolvido.

Relativamente à elevada taxa de derrota do algoritmo de *Q-Learning*, foi necessário realizar treinos mais longos, do que o que tinha sido inicialmente pensado.

Os treinos iniciais de 10000 jogos no total contra os diversos adversários, não eram suficientes para o algoritmo explorar jogadas suficientes e descobrir aquelas que proporcionavam as melhores pontuações. Assim sendo, treinou-se o algoritmo durante os 300000 jogos de modo a explorar um espaço suficientemente alargado do contexto do jogo, equilibrando a exploração realizada com o aproveitamento de jogadas já conhecidas.

No entanto, o algoritmo continua a apresentar uma taxa de derrota superior ao que seria esperado quando se encontra como jogador 2 ('O'). Todavia, este facto pode ser explicado pela facilidade que o jogador 1 tem para ganhar no jogo do galo. Como o jogador 1 é o primeiro a realizar o seu movimento, este decide como é que o jogo se vai desenrolar, sendo que o jogador 2 na maioria das vezes, encontra-se apenas a responder aos avanços do adversário. Também é de salientar que o jogador 1 tem um turno extra que segundo jogador (em 9 células, o jogador 1 pode jogar 5 vezes e o jogador 2 apenas 4). Fatores que podem explicar a taxa de derrota reduzida do jogador 1.

Em relação aos problemas mecânicos, a falta de binário dos motores e a não existência de mecanismos de *feedback* de posição foram as principais dificuldades e implicaram a alteração e adaptação do sistema idealizado.

Como os servo-motores constituintes do módulo de braço robótico não possuíam nenhum sistema de *feedback* de posição, foi necessário refazer o controlo do braço robótico inicial, uma vez que a rotina de movimentos suaves não funciona sem o conhecimento da posição do sistema.

A solução encontrada foi o movimento rápido e repentino dos servo-motores para uma posição determinada quando se inicia a aplicação principal. A partir deste ponto, é possível ler o último valor enviado para placa de Arm HAT e assim consegue-se ter uma ideia da última posição de cada servo-motor. As desvantagens deste método são inúmeras, como por exemplo as perdas de posição, contudo, é um método que na sua essência consegue atingir o objetivo proposto.

A falta de "força" dos motores foi a principal dificuldade deste projeto, levando o sistema proposto a ser adaptado. Os servo-motores, AD002 apenas exercem 2 kg.cm de binário, não sendo suficiente para executar o controlo efetivo do braço robótico e manter a posição dos elos. Um exemplo, é o servo-motor que controla a garra que não apresenta força suficiente para a abrir ou fechar, assim, o sistema não consegue manipular/pegar em objetos.

Desta forma, retiraram-se os dois últimos servo-motores (rotação do pulso e abertura da garra), reduzindo o sistema a um 3-DOF. Tendo isto em conta, em vez do sistema manipular objetos para efetuar as suas jogadas no jogo do galo na vida real, o sistema aponta com o marcador a posição onde eventualmente iria realizar a sua jogada.

Capítulo 8

Conclusões

A investigação e aprofundamento dos conhecimentos sobre a área de Inteligência Artificial e de Robótica foram os objetivos iniciais deste projeto. Assim, deliberou-se desenvolver um sistema que integrasse visão computacional e algoritmos de tomada de decisão para permitir que um braço robótico conseguisse executar uma simples tarefa interativa: jogar o jogo do galo de forma autónoma.

Ao longo de vários meses, vários componentes deste sistema foram sendo desenvolvidos e aperfeiçoados até serem integrados com sucesso no sistema principal. O módulo de visão computacional aplica um modelo YOLOv5 treinado para detetar e localizar com precisão os símbolos no tabuleiro de jogo. A tomada de decisão, um algoritmo de *Q-Learning*, permite que o robô selecione as melhores jogadas a realizar com base no estado atual. Por fim, o controlo do braço robótico converte as decisões do sistema em movimentos, permitindo que o robô interaja com o tabuleiro do jogo em tempo real.

Os resultados obtidos são bastante promissores e demonstram que o sistema desenvolvido é, efetivamente, capaz de jogar o jogo de forma autónoma (Figura 8.1).

Em retrospetiva, este projeto ambicioso revelou-se extremamente interessante e desafiante. Não só se pôde desenvolver conhecimentos nas áreas de estudo, como também se descobriu como combinar técnicas de visão computacional com *Reinforcement Learning* para a tomada de decisões autónomas em cenários reais.

A longo prazo, o sistema desenvolvido pode ser expandido a fim de executar outras tarefas para além do jogo do galo, nomeadamente jogos de tabuleiro mais

complexos ou aplicações industriais que exijam precisão e adaptabilidade. A capacidade do sistema para aprender e para se adaptar através das experiências passadas introduz inúmeras oportunidades futuras.

No entanto, apesar do êxito alcançado, o sistema desenvolvido apresenta algumas limitações.

8.1 Trabalho Futuro

Como trabalho futuro, propõe-se o aperfeiçoamento de vários aspetos do sistema.

A precisão do modelo de visão computacional, embora elevada, é ocasionalmente afetada pelas condições da imagem. Propõe-se a otimização deste componente para o tornar mais robusto, potencialmente através da implementação de diferentes modelos com uma maior capacidade.

Além disso, o algoritmo de *Q-Learning*, embora eficaz em contextos simples, pode não se adaptar bem a ambientes mais complexos sem modificações da sua lógica base. A integração de algoritmos de tomada de decisão mais sofisticados de modo a permitir ao sistema lidar com outro tipo de tarefas, é um dos aspetos a promover.

Por fim, as limitações físicas do braço robótico, incluindo a capacidade dos seus servo-motores, colocaram restrições à eficiência do sistema. Sugere-se um incremento do binário dos motores do braço robótico, para elevar a *performance* geral do sistema.

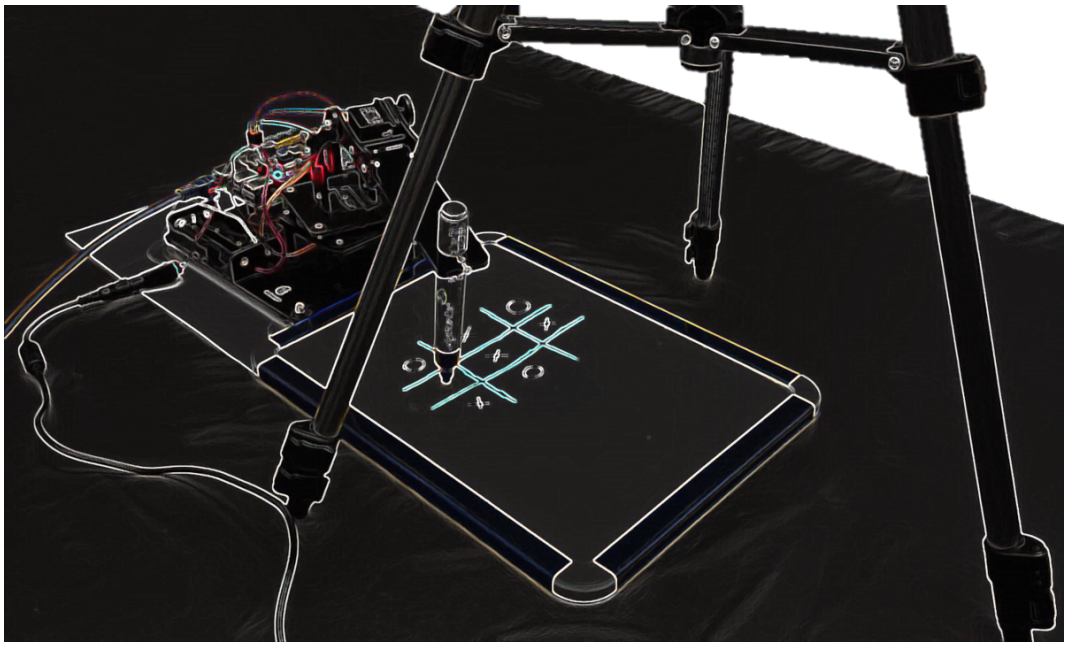


Figura 8.1: Sistema desenvolvido

Referências

[1] A. Buetti-Dinh, V. Galli, S. Bellenberg, O. Ilie, M. Herold, S. Christel, M. Boretska, I. V. Pivkin, P. Wilmes, W. Sand, M. Vera, and M. Dopson, "Deep neural networks outperform human expert's capacity in characterizing bioleaching bacterial biofilm composition," *Biotechnology Reports*, vol. 22, p. e00321, 2019. [Citado na página 2]

[2] M. E. Moran, "Evolution of robotic arms," *Journal of Robotic Surgery*, vol. 1, pp. 103–111, 5 2007. [Citado nas páginas 2 e 9]

[3] R. W. Lebling, "Robots of ages past," *AramcoWorld*, pp. 30–37, 11 2019. [Citado na página 8]

[4] A. Mayor, *Gods and Robots: Myths, Machines, and Ancient Dreams of Technology*. Princeton University Press, 2018. [Citado na página 8]

[5] M. Li, A. Milojević, and H. Handroos, "Robotics in manufacturing-the past and the present," *Technical, Economic and Societal Effects of Manufacturing 4.0: Automation, Adaption and Manufacturing in Finland and Beyond*, pp. 85–95, 1 2020. [Citado nas páginas ix, 8 e 9]

[6] S. Kalan, S. Chauhan, R. F. Coelho, M. A. Orvieto, I. R. Camacho, K. J. Palmer, and V. R. Patel, "History of robotic surgery," *Journal of Robotic Surgery*, vol. 4, pp. 141–147, 7 2010. [Citado na página 8]

[7] V. Kumar, "Introduction to robotics." Available at `http://www.universelle-automation.de/1978_Cincinatti.pdf`, 9 1998. (Last accessed in 17/03/2024). [Citado na página 8]

[8] I. Zamalloa, R. Kojcev, A. Hernández, I. Muguruza, L. Usategui, A. Bilbao, and V. Mayoral, "Dissecting robotics-historical overview and future perspectives," *Acutronic Robotics*, 2017. [Citado nas páginas 8 e 9]

[9] N. G. Hockstein, C. G. Gourin, R. A. Faust, and D. J. Terris, "A history of robots: From science fiction to surgical robotics," *Journal of Robotic Surgery*, vol. 1, pp. 113–118, 3 2007. [Citado nas páginas ix, 8 e 9]

[10] L. G. A. and Scalera, "From the unimate to the delta robot: The early decades of industrial robotics," in *Explorations in the History and Heritage of Machines*

and Mechanisms (M. Z. Baichun and Ceccarelli, eds.), pp. 284–295, Springer International Publishing, 2019. [Citado nas páginas ix e 9]

[11] M. J. Matarić, *The Robotics Primer*. Massachusetts Institute of Technology, 2007. [Citado nas páginas ix, 10, 11, 12, 13, 17, 18, 19, 20 e 21]

[12] Protolabs, "Materials that command the robotics industry." Available at `https://www.protolabs.com/en-gb/resources/blog/materials-that-command-the-robotics-industry/`. (Last accessed in 22/03/2024). [Citado na página 10]

[13] A. Robotics, "Exploring the anatomy of a robot - understanding the materials used in robotics design." Available at `https://www.awerobotics.com/exploring-the-anatomy-of-a-robot-understanding-the-materials-used-in-robotics-design/`. (Last accessed in 22/03/2024). [Citado na página 10]

[14] V. Patidar and R. Tiwari, "Survey of robotic arm and parameters," *2016 International Conference on Computer Communication and Informatics, ICCCI 2016*, 5 2016. [Citado na página 11]

[15] G. D. C. D. M. and Considine, *Robot Technology Fundamentals*, pp. 262–320. Springer US, 1986. [Citado na página 11]

[16] P. Li and X. Liu, "Common sensors in industrial robots: A review," *Journal of Physics: Conference Series*, p. 10, 2019. [Citado nas páginas 12, 14, 15 e 16]

[17] J. Luo, X. Zhou, C. Zeng, Y. Jiang, W. Qi, K. Xiang, M. Pang, and B. Tang, "Robotics perception and control: Key technologies and applications," *Micromachines*, vol. 15, 4 2024. [Citado nas páginas 12 e 15]

[18] J. Wang and D. Herath, "What makes robots? sensors, actuators, and algorithms," *Foundations of Robotics*, pp. 177–203, 2022. [Citado nas páginas ix, 12, 13, 14, 16, 17, 18, 20, 21 e 22]

[19] A. Navigation, "Inertial measurement unit (imu) | an introduction." Available at `https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/`, 2024. (Last accessed in 23/03/2024). [Citado nas páginas ix e 12]

[20] G. Benet, F. Blanes, J. E. Simó, and P. Pérez, "Using infrared sensors for distance measurement in mobile robots," *Robotics and Autonomous Systems*, vol. 1006, pp. 1–12, 2002. [Citado na página 14]

[21] H. Utama, Kevin, and H. Tanudjaja, "Smart street lighting system with data monitoring," *IOP Conference Series: Materials Science and Engineering*, vol. 1007, p. 12148, 9 2020. [Citado nas páginas ix e 14]

[22] R. N. Albustanji, S. Elmanaseer, and A. A. Alkhatib, "Robotics: Five senses plus one—an overview," *Robotics 2023, Vol. 12, Page 68*, vol. 12, p. 68, 5 2023. [Citado na página 15]

[23] K. Kim, J. Kim, X. Jiang, and T. Kim, "Static force measurement using piezoelectric sensors," *Journal of Sensors*, vol. 2021, pp. 1–8, 9 2021. [Citado nas páginas ix e 15]

[24] C. Benedek, A. Majdik, B. Nagy, Z. Rozsa, and T. Sziranyi, "Positioning and perception in lidar point clouds," *Digital Signal Processing*, vol. 119, p. 103193, 12 2021. [Citado nas páginas ix e 16]

[25] Synopsys, "What is lidar and how does it work?." Available at `https://www.synopsys.com/glossary/what-is-lidar.html`. (Last accessed in 22/03/2024). [Citado nas páginas ix e 16]

[26] D. Sebai, "Depth in focus: D component of rgb-d images and videos," *Electrical and Electronic Engineering*, 10 2023. [Citado na página 17]

[27] Editorial, "What are the different types of robot control systems?." Available at `https://roboticsbiz.com/what-are-the-different-types-of-robot-control-systems/`, 2023. (Last accessed in 26/03/2024). [Citado nas páginas 18, 19 e 20]

[28] A. Robotics, "Most common types of controllers in robotics." Available at `https://medium.com/@admin_71435/most-common-types-of-controllers-in-robotics-766e80eb0169`, 7 2023. (Last accessed in 25/03/2024). [Citado nas páginas 18 e 19]

[29] R. C. Arkin, "Reactive robotic systems," *Georgia Institute of Techonology*, 1998. [Citado na página 19]

[30] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artificial Intelligence*, vol. 247, pp. 10–44, 6 2017. [Citado na página 20]

[31] I. Q. Search, "Dc motor: What is it? how does it work? types, uses." Available at `https://www.iqsdirectory.com/articles/electric-motor/dc-motors.html`. (Last accessed in 24/03/2024). [Citado nas páginas ix, 20 e 21]

[32] O. Motor, "Stepper motor basics." Available at https://www.orientalmotor.com/stepper-motors/technology/stepper-motor-basics.html. (Last accessed in 24/03/2024). [Citado na página 21]

[33] C. Fiore, "Stepper motors: Types, uses and working principle | article | mps." Available at https://www.monolithicpower.com/en/learning/resources/stepper-motors-basics-types-uses. (Last accessed in 24/03/2024). [Citado nas páginas ix, 21 e 22]

[34] W. Mitianiec, *Modern Pneumatic and Combustion Hybrid Engines*, pp. 129–159. IntechOpen, 2017. [Citado na página 22]

[35] S. Sunena, "Pneumatic systems for aircraft," *Journal of Aeronautics and Aerospace Engineering*, vol. 11, pp. 1–2, 1 2022. [Citado na página 22]

[36] I. Moir and A. Seabridge, "Aircraft pneumatic subsystems," *Encyclopedia of Aerospace Engineering*, pp. 1–18, 9 2013. [Citado na página 22]

[37] Maverick, "Hydraulic actuators 101 | what is it, types and comparison." Available at https://www.maverickmachine.ca/news/hydraulic-actuators-101/. (Last accessed in 26/03/2024). [Citado na página 22]

[38] H. H. Poole, "Fundamentals of robotics engineering," *Fundamentals of Robotics Engineering*, 1989. [Citado na página 22]

[39] IFR, "Industrial robots definition - iso 8373:2012." Available at https://ifr.org/img/office/Industrial_Robots_2016_Chapter_1_2.pdf. (Last accessed in 18/03/2024). [Citado nas páginas ix, 22 e 26]

[40] M. Hägele, K. Nilsson, J. N. Pires, and R. Bischoff, "Industrial robotics," *Springer Handbooks*, pp. 1385–1422, 2016. [Citado na página 23]

[41] Telefónica, "Industrial robots: what they are, how they work, and what types exist -." Available at https://www.telefonica.com/en/communication-room/blog/industrial-robots-what-how-work-types/. (Last accessed in 16/03/2024). [Citado na página 23]

[42] M. Hägele, K. Nilsson, and J. N. Pires, "Industrial robotics," *Springer Handbook of Robotics*, pp. 963–986, 2008. [Citado na página 23]

[43] J. Automation, "How the manufacturing industry uses robotics and automation." Available at https://www.jrautomation.com/resources/manufacturing-robotics-automation. (Last accessed in 16/03/2024). [Citado nas páginas ix e 23]

[44] B. Becher, "What are industrial robots?." Available at `https://builtin.com/robotics/what-are-industrial-robots`. (Last accessed in 16/03/2024). [Citado nas páginas 23 e 25]

[45] D. Collins, "What is a cartesian robot?." Available at `https://www.linearmotiontips.com/what-is-a-cartesian-robot/`. (Last accessed in 17/03/2024). [Citado nas páginas ix, 23 e 24]

[46] J. Worth, "What is a cartesian robot (gantry) and how is it used in automation?." Available at `https://toolbox.igus.com/4397/cartesian-robots-for-automation`. (Last accessed in 17/03/2024). [Citado na página 23]

[47] A. Devasia, "What are cartesian robots?." Available at `https://control.com/technical-articles/what-are-cartesian-robots/`, 2021. (Last accessed in 17/03/2024). [Citado na página 24]

[48] M. E. Systems, "Cylindrical robots." Available at `https://www.mwes.com/types-of-industrial-robots/cylindrical-robots/`. (Last accessed in 19/03/2024). [Citado na página 24]

[49] A. Alsabbagh, "Data analysis and force control of a 6-dof industrial robot," Master's thesis, University of Debrecen Faculty of Engineering Mechatronics, 6 2019. [Citado nas páginas ix e 24]

[50] I. Process Solutions, "What are the different types of industrial robots and their applications?." Available at `https://processsolutions.com/what-are-the-different-types-of-industrial-robots-and-their-applications/`. (Last accessed in 19/03/2024). [Citado nas páginas ix e 25]

[51] Fanuc, "Scara robots." Available at `https://www.fanuc.eu/it/en/robots/robot-filter-page/scara-series/selection-support`. (Last accessed in 21/03/2024). [Citado na página 25]

[52] A. Owen-Hill, "What is a scara robot? the background and benefits." Available at `https://robodk.com/blog/what-is-a-scara-robot/`, 3 2023. (Last accessed in 22/03/2024). [Citado na página 25]

[53] R. Rao, "What are articulated robots? anatomy, control systems, advantages, selection criteria, and applications." Available at `https://www.wevolver.com/article/articulated-robots`, 3 2023. (Last accessed in 21/03/2024). [Citado na página 25]

[54] A. H. Chiang and S. Trimi, "Impacts of service robots on service quality," *Service Business*, vol. 14, p. 439, 9 2020. [Citado nas páginas ix, 26 e 27]

[55] I. Lee, J. M. Corchado, S. Kollias, and J. Taheri, "Service robots: A systematic literature review," *Electronics 2021, Vol. 10, Page 2658*, vol. 10, p. 2658, 10 2021. [Citado na página 26]

[56] M. Söderlund, "Service robots with (perceived) theory of mind: An examination of humans' reactions," *Journal of Retailing and Consumer Services*, vol. 67, p. 102999, 7 2022. [Citado na página 26]

[57] P. Asgharian, A. M. Panchea, and F. Ferland, "A review on the use of mobile service robots in elderly care," *Robotics 2022, Vol. 11, Page 127*, vol. 11, p. 127, 11 2022. [Citado nas páginas ix, 26 e 27]

[58] M. Chen, X. Wang, R. Law, and M. Zhang, "Research on the frontier and prospect of service robots in the tourism and hospitality industry based on international core journals: A review," *Behavioral Sciences*, vol. 13, 7 2023. [Citado na página 27]

[59] S. Automotive, "The 6 levels of vehicle autonomy explained." Available at `https://www.synopsys.com/automotive/autonomous-driving-levels.html`. (Last accessed in 20/03/2024). [Citado nas páginas ix e 28]

[60] N. Barla, "Self-driving cars with convolutional neural networks (cnn)." Available at `https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn`, 8 2021. (Last accessed in 30/03/2024). [Citado na página 28]

[61] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, pp. 712–733, 2 2021. [Citado na página 28]

[62] D. Parekh, N. Poddar, A. Rajpurkar, M. Chahal, N. Kumar, G. P. Joshi, and W. Cho, "A review on autonomous vehicles: Progress, methods and challenges," *Electronics 2022, Vol. 11, Page 2162*, vol. 11, p. 2162, 7 2022. [Citado nas páginas ix e 29]

[63] G. Lefranc, I. Lopez-Juarez, R. Osorio-Comparán, and M. Peña-Cabrera, "Impact of cobots on automation," *Procedia Computer Science*, vol. 214, pp. 71–78, 1 2022. [Citado nas páginas 29 e 30]

[64] J. marc Buchert, "Cobot vs industrial robot: Differences and comparison." Available at `https://manplusmachines.com/differences-robots-cobots/`. (Last accessed in 30/03/2024). [Citado na página 29]

[65] B. Vanderborght, J. Berte, G. Coppel, I. E. Makrini, S. Elprama, C. Jewell, A.-J. Knevels, J. Potargent, I. Ravyse, K. Schroyen, F. Stals, J. den Bergh, T. der Auwermeulen, T. Waegeman, and A. Jacobs, "Towards an acceptable socially collaborative robot for the manufacturing industry," 9 2017. [Citado nas páginas ix e 30]

[66] R. Forghani, "Machine learning and other artificial intelligence applications," *Neuroimaging Clinics*, vol. 30, p. i, 11 2020. doi: 10.1016/S1052-5149(20)30067-8. [Citado nas páginas 32 e 33]

[67] Z.-X. Cai, L. Liu, B. Chen, and Y. Wang, *Introduction*, pp. 1–31. World Scientific, 6 2021. [Citado nas páginas 32 e 33]

[68] J. Veisdal, "The birthplace of ai." Available at `https://www.cantorsparadise.com/the-birthplace-of-ai-9ab7d4e5fb00`, 9 2019. (Last accessed in 07/09/2024). [Citado nas páginas ix e 32]

[69] N. Narsinghani, "The timeline of artificial intelligence - from the 1940s to the 2020s." Available at `https://verloop.io/blog/the-timeline-of-artificial-intelligence-from-the-1940s/`, 2024. (Last accessed in 07/09/2024). [Citado nas páginas ix, 32 e 33]

[70] Tableau, "What is the history of artificial intelligence (ai)?." Available at `https://www.tableau.com/data-insights/ai/history`. (Last accessed in 07/09/2024). [Citado nas páginas 32 e 33]

[71] G. Matos, "Demystifying artificial intelligence: Understanding its evolution, key areas, and fundamentals." Available at `https://share.atelie.software/demystifying-artificial-intelligence-understanding-its-evolution-key-areas-and-fundamentals-2dd875364ea3`, 9 2023. (Last accessed in 10/09/2024). [Citado nas páginas ix e 34]

[72] S. Brown, "Machine learning, explained." Available at `https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained`, 4 2021. (Last accessed in 10/09/2024). [Citado nas páginas 35 e 37]

[73] M. Nishadini, "Introduction to machine learning. these days, machine learning(ml) has... | by malsha nishadini | medium." Available at `https://medium.com/@malsha_nishadini/introduction-to-machine-learning-19539f54d145`, 4 2019. (Last accessed in 09/09/2024). [Citado nas páginas ix, 35 e 38]

[74] R. Pugliese, S. Regondi, and R. Marini, "Machine learning-based approach: global trends, research directions, and regulatory standpoints," *Data Science and Management*, vol. 4, pp. 19–29, 12 2021. [Citado nas páginas x e 36]

[75] Priyadharshini, "What is machine learning and types of machine learning." Available at https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-machine-learning, 3 2023. (Last accessed in 08/09/2024). [Citado nas páginas 36 e 38]

[76] N. Agarwal and D. Yadav, "A comprehensive analysis of classical machine learning and modern deep learning methodologies," *International Journal of Engineering Research and*, vol. 13, 9 2024. [Citado nas páginas x e 36]

[77] R. I. Muhamedyev, "Machine learning methods: An overview," *COMPUTER MODELLING and NEW TECHNOLOGIES*, pp. 14–29, 2015. [Citado na página 37]

[78] V. Kanade, "What is machine learning? understanding types and applications." Available at https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/, 8 2022. (Last accessed in 10/09/2024). [Citado na página 37]

[79] S. Doshi, *Commonsense Validation and Reasoning using Natural Language Processing*. PhD thesis, Cork Institute of Techonlogy, 9 2020. [Citado nas páginas x e 37]

[80] J. Alzubi, A. Nayyar, and A. Kumar, "Machine learning from theory to algorithms: An overview," *Journal of Physics: Conference Series*, vol. 1142, p. 012012, 11 2018. [Citado na página 38]

[81] G. Sharma, "A gentle introduction to semi supervised learning." Available at https://medium.com/@gayatri_sharma/a-gentle-introduction-to-semi-supervised-learning-7afa5539beea. (Last accessed in 27/09/2024). [Citado nas páginas x e 38]

[82] D. Fumo, "Types of machine learning algorithms you should know." Available at https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861, 6 2017. (Last accessed in 10/09/2024). [Citado nas páginas x e 38]

[83] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, pp. 1140–1144, 12 2018. [Citado na página 39]

[84] M. Reynolds, "Deepmind's ai beats world's best go player in latest face-off." Available at https://www.newscientist.com/article/2132086-deepminds-

ai-beats-worlds-best-go-player-in-latest-face-off/, 5 2017. (Last accessed in 10/09/2024). [Citado nas páginas x e 39]

[85] O. A. M. López, A. M. López, and J. Crossa, "Fundamentals of artificial neural networks and deep learning," *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, pp. 379–425, 2022. [Citado nas páginas x, 39 e 40]

[86] Q. Bi, K. E. Goodman, J. Kaminsky, and J. Lessler, "What is machine learning? a primer for the epidemiologist," *American Journal of Epidemiology*, vol. 188, pp. 2222–2239, 12 2019. [Citado nas páginas 39, 47, 50 e 51]

[87] S. A. Marhon, C. J. Cameron, and S. C. Kremer, "Recurrent neural networks," *Intelligent Systems Reference Library*, vol. 49, pp. 29–65, 2013. [Citado na página 40]

[88] S. Ma, B. Xiao, R. Hong, B. Addissie, Z. Drikas, T. Antonsen, E. Ott, and S. Anlage, "Classification and prediction of wave chaotic systems with machine learning techniques," *Acta Physica Polonica A*, 11 2019. [Citado nas páginas x e 40]

[89] O. Orojo, J. Tepper, T. M. McGinnity, and M. Mahmud, "The multi-recurrent neural network for state-of-the-art time-series processing," *Procedia Computer Science*, vol. 222, pp. 488–498, 1 2023. [Citado na página 40]

[90] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Citado nas páginas x, 41 e 42]

[91] S. Indolia, A. K. Goswami, S. P. Mishra, and P. Asopa, "Conceptual understanding of convolutional neural network- a deep learning approach," *Procedia Computer Science*, vol. 132, pp. 679–688, 1 2018. [Citado na página 41]

[92] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature 2015 521:7553*, vol. 521, pp. 436–444, 5 2015. [Citado na página 41]

[93] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, p. 611, 8 2018. [Citado na página 41]

[94] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, pp. 1–43, 4 2024. [Citado nas páginas x e 41]

[95] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, pp. 1–20, 11 2021. [Citado na página 41]

[96] J. Brownlee, "What is deep learning?." Available at `https://machinelearningmastery.com/what-is-deep-learning/`, 8 2019. (Last accessed in 11/09/2024). [Citado nas páginas x e 42]

[97] I. Zeroual and A. Lakhouaja, "Data science in light of natural language processing: An overview," *Procedia Computer Science*, vol. 127, pp. 82–91, 1 2018. [Citado na página 43]

[98] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, pp. 3713–3744, 1 2023. [Citado na página 43]

[99] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association : JAMIA*, vol. 18, p. 544, 9 2011. [Citado na página 43]

[100] R. Dass, "The essential guide to how nlp works." Available at `https://medium.com/@ritidass29/the-essential-guide-to-how-nlp-works-4d3bb23faf76`, 9 2018. (Last accessed in 11/09/2024). [Citado na página 43]

[101] Turing, "What are the applications of quantum nlp for translation?." Available at `https://www.turing.com/kb/applications-of-quantum-nlp-for-translation`. (Last accessed in 11/09/2024). [Citado nas páginas x, 43 e 44]

[102] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer-Verlag, 1st ed., 2010. [Citado na página 44]

[103] I. Mihajlovic, "Everything you ever wanted to know about computer vision.." Available at `https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e`, 4 2019. (Last accessed in 11/09/2024). [Citado na página 44]

[104] R. M. Hussien, K. Q. Al-Jubouri, M. A. Gburi, A. G. H. Qahtan, and A. H. D. Jaafar, "Computer vision and image processing the challenges and opportunities for new technologies approach: A paper review," *Journal of Physics: Conference Series*, vol. 1973, p. 012002, 8 2021. [Citado na página 44]

[105] A. I. Khan and S. Al-Habsi, "Machine learning in computer vision," *Procedia Computer Science*, vol. 167, pp. 1444–1451, 1 2020. [Citado nas páginas x, 44 e 45]

[106] J. E. T. Akinsola, A. Jet, and H. J. O, "Supervised machine learning algorithms: Classification and comparison," *International Journal of Computer Trends and Technology*, vol. 48, 2017. [Citado nas páginas 45, 47, 50 e 51]

[107] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, pp. 1–21, 5 2021. [Citado nas páginas 45, 46, 47, 48, 49, 50 e 52]

[108] J. Wilber, "Linear regression." Available at `https://mlu-explain.github.io/linear-regression/`, 9 2022. (Last accessed in 12/09/2024). [Citado na página 45]

[109] Javatpoint, "Linear regression in machine learning." Available at `https://www.javatpoint.com/linear-regression-in-machine-learning`. (Last accessed in 12/09/2024). [Citado nas páginas x e 46]

[110] V. Nasteski, "An overview of the supervised machine learning methods," *HORIZONS.B*, vol. 4, 12 2017. [Citado nas páginas 45, 46 e 47]

[111] M. Gupta, "Linear regression in machine learning." Available at `https://www.geeksforgeeks.org/ml-linear-regression/`, 9 2024. (Last accessed in 12/09/2024). [Citado na página 46]

[112] GeeksforGeeks, "Logistic regression in machine learning." Available at `https://www.geeksforgeeks.org/understanding-logistic-regression/`, 6 2024. (Last accessed in 12/09/2024). [Citado na página 46]

[113] E. Bisong, "Logistic regression," *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 243–250, 2019. [Citado nas páginas x e 47]

[114] P. Gupta, "Decision trees in machine learning." Available at `https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052`, 5 2017. (Last accessed in 12/09/2024). [Citado nas páginas 47 e 48]

[115] DataCamp, "Python decision tree classification tutorial: Scikit-learn decisiontreeclassifier." Available at `https://www.datacamp.com/tutorial/decision-tree-classification-python`, 2 2023. (Last accessed in 12/09/2024). [Citado nas páginas x e 47]

[116] X. Wu, Y. Gao, and D. Jiao, "Multi-label classification based on random forest algorithm for non-intrusive load monitoring system," *Processes 2019, Vol. 7, Page 337*, vol. 7, p. 337, 6 2019. [Citado na página 48]

[117] Shiksha, "Random forest algorithm: Python code." Available at `https://www.shiksha.com/misc-courses/articles/random-forest-algorithm-python-code/`. (Last accessed in 12/09/2024). [Citado nas páginas x e 48]

[118] R. Shah, "Introduction to k-nearest neighbors (knn) algorithm." Available at https://ai.plainenglish.io/introduction-to-k-nearest-neighbors-knn-algorithm-e8617a448fa8, 3 2021. (Last accessed in 13/09/2024). [Citado na página 49]

[119] H. K. Gianey and R. Choudhary, "Comprehensive review on supervised machine learning algorithms," *Proceedings - 2017 International Conference on Machine Learning and Data Science, MLDS 2017*, vol. 2018-January, pp. 38–43, 7 2017. [Citado na página 49]

[120] IBM, "What is the k-nearest neighbors algorithm?." Available at https://www.ibm.com/topics/knn. (Last accessed in 13/09/2024). [Citado nas páginas x e 50]

[121] GeeksforGeeks, "K-nearest neighbor(knn) algorithm." Available at https://www.geeksforgeeks.org/k-nearest-neighbours/, 7 2024. (Last accessed in 13/09/2024). [Citado na página 49]

[122] E. Academy, "What are some limitations of the k nearest neighbors algorithm in terms of scalability and training process?," 8 2023. (Last accessed in 13/09/2024). [Citado na página 49]

[123] A. M. Deris, A. M. Zain, and R. Sallehuddin, "Overview of support vector machine in modeling machining performances," *Procedia Engineering*, vol. 24, pp. 308–312, 1 2011. [Citado na página 50]

[124] D. J. Armaghani, P. G. Asteris, B. Askarian, M. Hasanipanah, R. Tarinejad, and V. V. Huynh, "Examining hybrid and single svm models with different kernels to predict rock brittleness," *Sustainability 2020, Vol. 12, Page 2229*, vol. 12, p. 2229, 3 2020. [Citado nas páginas x e 50]

[125] S. Huang, C. A. Nianguang, P. P. Pacheco, S. Narandes, Y. Wang, and X. U. Wayne, "Applications of support vector machine (svm) learning in cancer genomics," *Cancer Genomics and Proteomics*, vol. 15, p. 41, 1 2018. [Citado na página 50]

[126] R. Guido, S. Ferrisi, D. Lofaro, and D. Conforti, "An overview on the advancements of support vector machine models in healthcare applications: A review," *Information 2024, Vol. 15, Page 235*, vol. 15, p. 235, 4 2024. [Citado nas páginas x e 51]

[127] E. U. Oti, M. O. Olusola, F. C. Eze, and S. U. Enogwe, "Comprehensive review of k-means clustering algorithms," *International Journal of Advances in Scientific Research and Engineering*, vol. 07, pp. 64–69, 2021. [Citado na página 51]

[128] Y. Li and H. Wu, "A clustering method based on k-means algorithm," *Physics Procedia*, vol. 25, pp. 1104–1109, 1 2012. [Citado na página 51]

[129] X. Jin and J. Han, "K-means clustering," *Encyclopedia of Machine Learning*, pp. 563–564, 2011. [Citado nas páginas x e 51]

[130] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics 2020, Vol. 9, Page 1295*, vol. 9, p. 1295, 8 2020. [Citado na página 52]

[131] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning 1992 8:3*, vol. 8, pp. 279–292, 5 1992. [Citado na página 52]

[132] B. Jang, M. Kim, G. Harerimana, and J. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. PP, p. 1, 9 2019. [Citado nas páginas 52 e 53]

[133] A. Lamba, "An introduction to q-learning: reinforcement learning." Available at `https://medium.com/free-code-camp/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc`, 9 2018. (Last accessed in 15/09/2024). [Citado nas páginas x e 53]

[134] K. Sivamayil, E. Rajasekar, B. Aljafari, S. Nikolovski, S. Vairavasundaram, and I. Vairavasundaram, "A systematic study on reinforcement learning based applications," *Energies 2023, Vol. 16, Page 1512*, vol. 16, p. 1512, 2 2023. [Citado na página 52]

[135] S. Corecco, G. Adorni, and L. M. Gambardella, "Proximal policy optimization-based reinforcement learning and hybrid approaches to explore the cross array task optimal solution," *Machine Learning and Knowledge Extraction 2023, Vol. 5, Pages 1660-1679*, vol. 5, pp. 1660–1679, 11 2023. [Citado na página 53]

[136] GeeksforGeeks, "A brief introduction to proximal policy optimization." Available at `https://www.geeksforgeeks.org/a-brief-introduction-to-proximal-policy-optimization/`, 2 2022. (Last accessed in 15/09/2024). [Citado na página 53]

[137] T. Team, "Proximal policy optimization." Available at `https://toloka.ai/blog/proximal-policy-optimization/`, 2 2024. (Last accessed in 15/09/2024). [Citado na página 54]

[138] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, "Proximal policy optimization algorithms," *ArXiv*, 2017. [Citado na página 54]

[139] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cognitive Robotics*, vol. 3, pp. 54–70, 1 2023. [Citado na página 54]

[140] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object detection using yolo: challenges, architectural successors, datasets and applications," *Multimedia Tools and Applications*, vol. 82, p. 9243, 3 2023. [Citado na página 54]

[141] M. Hussain, "Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection," *Machines 2023, Vol. 11, Page 677*, vol. 11, p. 677, 6 2023. [Citado na página 54]

[142] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 6 2016. [Citado nas páginas x, xii, 54, 55 e 145]

[143] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*, p. 21–37. Springer International Publishing, 2016. [Citado nas páginas x, 55 e 56]

[144] S. H. Kang and J. S. Park, "Aligned matching: Improving small object detection in ssd," *Sensors (Basel, Switzerland)*, vol. 23, 3 2023. [Citado na página 56]

[145] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 11 2013. [Citado nas páginas x e 57]

[146] P. Potrimba, "What is r-cnn?." Available at `https://blog.roboflow.com/what-is-r-cnn/`. (Last accessed in 16/09/2024). [Citado na página 57]

[147] R. Gandhi, "R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms." Available at `https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e`, 7 2018. (Last accessed in 16/09/2024). [Citado na página 57]

[148] R. Divya and J. Peter, "Smart healthcare system-a brain-like computing approach for analyzing the performance of detectron2 and posenet models for anomalous action detection in aged people with movement impairments," *Complex and Intelligent Systems*, vol. 8, 9 2021. [Citado nas páginas x e 58]

[149] Ultralytics, "Ultralytics github." Available at `https://github.com/ultralytics/ultralytics`. (Last accessed in 28/09/2024). [Citado na página 58]

[150] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, "Yolov10: Real-time end-to-end object detection," *ArXiv*, 5 2024. [Citado na página 58]

[151] Adeept, "Adeept rasparm 4-dof robotic arm kit for raspberry pi 4/3 model b/b+/2b | robot arm kit with python pc software and remote control." Available at `https://www.adeept.com/rasparm_p0126.html`. (Last accessed in 05/09/2024). [Citado nas páginas x e 61]

[152] Keyestudio, "Keyestudio 4df mechanical ps2 joystick metallic robot arm learning starter kit v2.0 for arduino diy." Available at `https://www.keyestudio.com/products/keyestudio-4df-mechanicals2-joystick-metallic-robot-arm-learning-starter-kit-v20-for-arduino-diy`. (Last accessed in 05/09/2024). [Citado nas páginas x, xi, 62 e 69]

[153] Adeept, "Adeept 5-dof robotic arm kit for raspberry pi 4 b 3 b+ b a+, programmable diy coding stem educational 5 axis robot arm with python code and tutorials(." Available at `https://www.adeept.com/robotic-arm-kit-rpi-black_p0368.html`. (Last accessed in 23/09/2024). [Citado nas páginas x, xi, 62, 64, 71, 75 e 78]

[154] Hiwonder, "Hiwonder armpi mini 4dof vision robotic arm powered by raspberry pi." Available at `https://www.hiwonder.com/products/armpi-mini`. (Last accessed in 05/09/2024). [Citado nas páginas x, xi, 62, 66 e 72]

[155] Makerfabs, "6 dof robot arm with raspberry pi pico | makerfabs." Available at `https://www.makerfabs.com/raspberry-pi-pico-6-dof-robot-arm.html`. (Last accessed in 05/09/2024). [Citado nas páginas x e 62]

[156] Adafruit, "Standard servo - towerpro sg-5010." Available at `https://www.adafruit.com/product/155`. (Last accessed in 17/09/2024). [Citado nas páginas x, 64 e 65]

[157] ThinkRobotics, "Standard metal servo (mg946r)." Available at `https://thinkrobotics.com/products/standard-metal-servo-mg946r-1`. (Last accessed in 17/09/2024). [Citado nas páginas x e 65]

[158] Components101, "Mg90s servo, metal gear with one bearing." Available at `https://components101.com/sites/default/files/component_datasheet/MG90S-Datasheet.pdf`. (Last accessed in 17/09/2024). [Citado na página 65]

[159] Adeept, "Ad002 servo motor." Available at `https://www.adeept.com/ad002-servo-motorx2_p0274.html`. (Last accessed in 17/09/2024). [Citado na página 66]

[160] G. Electronic, "Ld-1501mg digital servo." Available at https://www.gie.com.my/shop.php?action=robotics/motors/LD_1501MG. (Last accessed in 17/09/2024). [Citado na página 66]

[161] Hiwonder, "Ldx-218 full metal gear digital servo." Available at https://www.hiwonder.com/products/ldx-218. (Last accessed in 17/09/2024). [Citado na página 66]

[162] Hiwonder, "Lfd-01m robot servo 6v 14g micro servo." Available at https://www.hiwonder.com/products/lfd-01m. (Last accessed in 17/09/2024). [Citado na página 66]

[163] E. Caldas, "Mg996r high torque metal gear dual ball bearing servo." Available at https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf. (Last accessed in 17/09/2024). [Citado na página 67]

[164] Amazon, "Yf-6125mg metal digital servo 25kg high torque." Available at https://www.amazon.com/Replacement-YF-6125MG-YF6130MG-Waterproof-Anti-Burn/dp/B09YKNG4PN. (Last accessed in 17/09/2024). [Citado nas páginas xi e 67]

[165] G. Dandabathula, D. Agrawal, S. Lohiya, and S. Sitiraju, "Design and implementation of lab grade interplanetary rover for educational purpose," tech. rep., National Remote Sensing Centre, 9 2019. [Citado nas páginas xi e 70]

[166] R. Pi, "Raspberry pi 4 model b." Available at https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. (Last accessed in 18/09/2024). [Citado nas páginas xi e 70]

[167] Adafruit, "Pca9685." Available at https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf. (Last accessed in 18/09/2024). [Citado na página 71]

[168] STMicroelectronics, "L298 dual full-bridge driver." Available at https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf, 2000. (Last accessed in 18/09/2024). [Citado na página 71]

[169] R. Pi, "Pico-series microcontrollers." Available at https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html. (Last accessed in 18/09/2024). [Citado nas páginas xi e 73]

[170] A. Nesmiyanova, "Python by the numbers: How strong is the demand for python on the software development market in 2021?." Available at https://steelkiwi.com/blog/python-demand-on-development-market/, 2021. (Last accessed in 20/09/2024). [Citado na página 91]

[171] S. Lumholt and G. van Rossum, "tkinter." Available at `https://docs.python.org/3/library/tkinter.html`. (Last accessed in 20/09/2024). [Citado na página 91]

[172] F. AI, "Pytorch." Available at `https://pytorch.org/`. (Last accessed in 20/09/2024). [Citado na página 91]

[173] W. G. Intel and Itseez, "Opencv - open computer vision library." Available at `https://opencv.org/`. (Last accessed in 20/09/2024). [Citado na página 91]

[174] K.-P. Lindegaard, "smbus2 · pypi." Available at `https://pypi.org/project/smbus2/`. (Last accessed in 20/09/2024). [Citado na página 92]

[175] Microsoft, "Visual studio code - code editing. redefined." Available at `https://code.visualstudio.com/`. (Last accessed in 20/09/2024). [Citado na página 92]

[176] J. Nelson, "Roboflow: Computer vision tools for developers and enterprises." Available at `https://roboflow.com/`. (Last accessed in 20/09/2024). [Citado na página 92]

[177] Google, "Kaggle." Available at `https://www.kaggle.com/`. (Last accessed in 20/09/2024). [Citado na página 92]

[178] YOLO, "Tic tac toe dataset." Available at `https://universe.roboflow.com/yolo-tbs7b/tic-tac-toe-gdqgd`, jun 2022. (Last accessed in 24/09/2024). [Citado na página 95]

[179] carlostumo, "cell-detection dataset." Available at `https://universe.roboflow.com/carlostumo/cell-detection-5hlob`, feb 2024. (Last accessed in 24/09/2024). [Citado na página 95]

[180] Y. dataset, "Tris dataset." Available at `https://universe.roboflow.com/yolo-dataset-rwndt/tris`, mar 2023. (Last accessed in 24/09/2024). [Citado na página 95]

[181] tictactoe, "tictactoe dataset." Available at `https://universe.roboflow.com/tictactoe/tictactoe-qsz5b`, mar 2024. (Last accessed in 24/09/2024). [Citado na página 95]

[182] TE, "tictactoe dataset." Available at `https://universe.roboflow.com/te-xkrgm/tictactoe-kpr4t`, jul 2024. (Last accessed in 24/09/2024). [Citado na página 95]

[183] U. Team, "Ultralytics yolo docs." Available at `https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/`. (Last accessed in 24/09/2024). [Citado nas páginas xi e 98]

[184] B. Jorgensen, "Minimax." Available at `https://beej.us/blog/data/minimax/`. (Last accessed in 25/09/2024). [Citado nas páginas xi e 105]

[185] R. Xu, H. Lin, K. Lu, L. Cao, and Y. Liu, "A forest fire detection system based on ensemble learning," *Forests*, vol. 12, p. 217, 9 2021. [Citado nas páginas xii e 146]

Anexo A

Classe de Controlo "PCA9685"

A.1 Configuração da Frequência de PWM

Para definir a frequência de PWM é necessário editar os registos da placa segundo o seu *datasheet*. Desta forma, para que a frequência de controlo dos motores seja de 50 Hz, é necessário calcular o valor para dar ao *prescaler*:

$$\begin{aligned} \text{Prescaler Value} &= round\left(\frac{\text{Osc. Clock}}{4096 \times \text{Update Rate}}\right) - 1 \\ &= round\left(\frac{25\text{ MHz}}{4096 \times 50\text{ Hz}}\right) - 1 \\ &= 121 \end{aligned} \tag{A.1}$$

Sabendo que o valor de *clock* é de 25 MHz, o valor a atribuir ao *prescaler*, que é configurado no registo 254, é de 121. Contudo, para poder modificar este registo é necessário mudar o modo de operação da placa no registo 0. Para isto, é necessário seguir uma ordem bem definida:

1. Ler e guardar o modo de operação atual;
2. Enviar o novo modo certificando-se de que apenas se altera o último bit para 0 e o quarto bit para 1, de forma a garantir que a placa não se encontra a gerar sinais e está disposta a ser configurada;
3. Escrever o valor do *prescaler* no registo 254;

4. Enviar o modo de operação anterior;
5. Esperar no mínimo 500 microssegundos para o oscilador estabilizar;
6. Reescrever o último bit do registo 0 para reiniciar os canais de PWM a fim de operar com as novas definições.

A.2 Controlo de um Servo-motor

O IC 9685 permite controlar um servo-motor específico, tendo em conta o canal em que este se encontra ligado. Deste modo, é necessário escrever nos dois registos de 12 bits de cada canal os valores de PWM calculados.

Após vários testes, descobriu-se que um ângulo de 0° corresponde a um valor de 80 ciclos *off* em 4096 ciclos de PWM e 180° a 520, perfazendo, assim, 1.95 % e 12.7 %, de *Duty Cycle*, respetivamente.

Anexo B

Modelo YOLO

B.1 Estrutura do Modelo YOLOv1

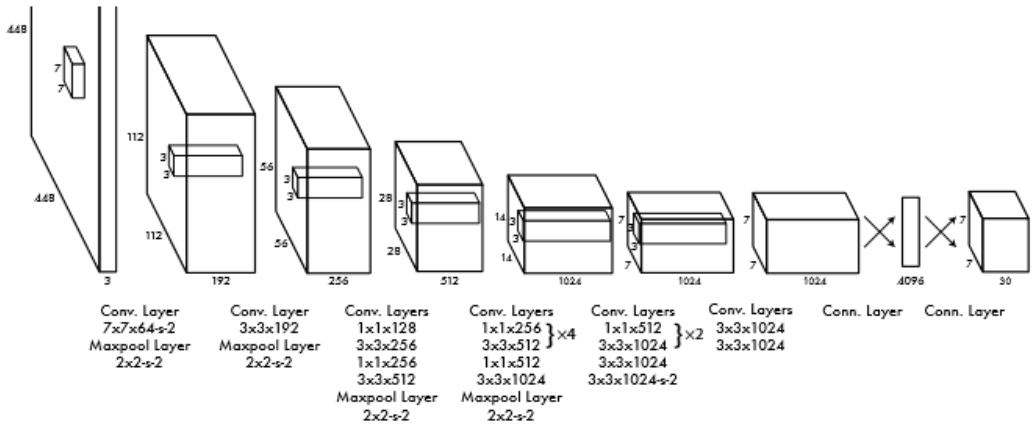


Figura B.1: Estrutura do modelo YOLO [142]

B.2 Estrutura do Modelo YOLOv5

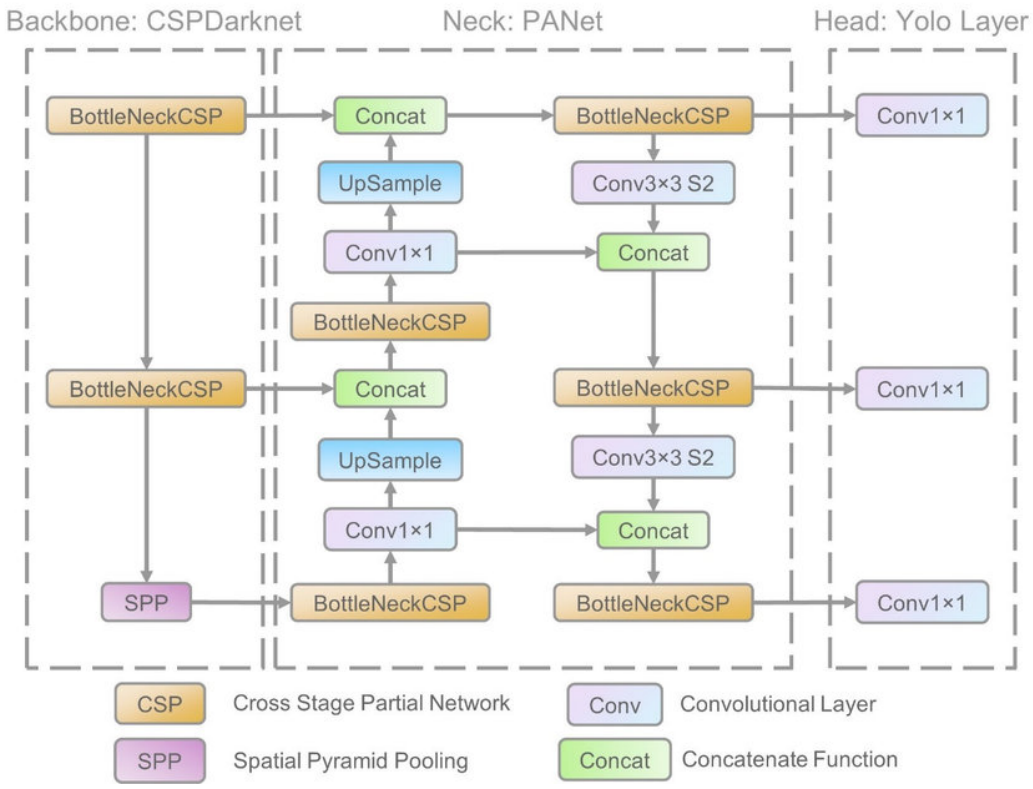


Figura B.2: Estrutura do modelo YOLOv5 [185]

Anexo C

Código Desenvolvido

Disponível em: Afonso Timóteo - *Robotic Arm with Reinforcement Learning*