INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia Eletrotécnica Energia e Automação

JOGO DO GALO COM O ROBOT

RÚBEN ALEXANDRE DE GONÇALVES ORVALHO

(Licenciado em Engenharia Eletrotécnica)

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

Orientadora:
Professora Doutora Maria da Graça Vieira de Brito Almeida (ISEL)

Júri:
Presidente:
Professor Doutor Filipe André de Sousa Figueira Barata (ISEL)
Vogais:
Professor Doutor Armando José Leitão Cordeiro (ISEL)
Professora Doutora Maria da Graça Vieira de Brito Almeida (ISEL)

Fevereiro de 2023

RESUMO

A dissertação apresentada foi desenvolvida com o intuito de aliar tecnologia de processamento de imagem com mecanismos de decisão e cinemática. Desta forma, utilizando um ambiente de interação com o utilizador pode-se dar resposta ao mesmo de forma autónoma.

O ambiente escolhido é o tradicional jogo do galo, este é composto por um tabuleiro com 9 casas e destinado a 2 jogadores. O jogo começa com a escolha da marca a utilizar por cada jogador, *X* ou *O* em seguida, alternadamente, cada jogador deve preencher uma casa do tabuleiro com a sua marca sendo a vitória atribuída ao jogador que conseguir colocar três peças em linha com a sua marca, esta pode ser vertical, horizontal ou diagonal.

Para a interação do computador com o jogador vai ser utilizado um manipulador para traduzir fisicamente a resposta calculada pelo computador à jogada do utilizador. Neste âmbito entram em jogo duas disciplinas, muito abrangentes e cada vez mais importantes no desenvolvimento tecnológico mundial, robótica e processamento de imagem.

No campo do processamento de imagem tem-se como tarefas: identificação do estado do ambiente de interação, processamento da jogada do utilizador e recolha da posição física das peças colocadas em jogo. Aliado a este processo existe um algoritmo que com base no estado do jogo irá calcular a melhor resposta a efetuar pelo manipulador.

De forma a conseguir uma recolha correta do ambiente de jogo, foram utilizadas rotinas de processamento de imagem com métodos para reduzir o ruído que possa ser introduzido por variáveis externas ou alterações no ambiente e corretamente identificar a peça introduzida em jogo.

Após a correta análise do estado atual do tabuleiro de jogo e com base na jogada calculada para o computador é determinado o movimento a executar pelo manipulador com recurso a cinemática. Com a informação necessária para realizar o movimento da peça a mesma é traduzida com recurso a um sistema dedicado ao seu tratamento e compatibilização com o manipulador.

Em seguida os movimentos de cada junta previamente determinados são convertidos para grandezas analógicas através dum processador Arduíno e enviados para o manipulador.

Palavras-chave: Interação Humano-Computador, Processamento de Imagem, Arduino, Cinemática.

ABSTRACT

The dissertation presented was developed with the aim of combining computational vision technology with decision-making mechanisms and kinematics. In this way, using an interaction environment with the user, we will be able to respond autonomously.

The environment chosen is the traditional tic-tac-toe game, it consists of a board with 9 fields and intended for 2 players. The game begins with the choice of the mark that will used by each player, X or O then alternately, each player must fill a field on the board with their mark, with the victory being attributed to the player who manages to place three pieces in line with their mark, this line can be vertical, horizontal or diagonal.

For the interaction between the computer and the player, a manipulator will be used to physically translate the response calculated by the computer. In this context, two disciplines come into play both being very comprehensive and increasingly important in the current worlds' technological development, robotics and image processing by means of computational algorithms.

In the field of computational vision, we have the following tasks: identifying the state of the interaction environment, processing the user's move and collecting the physical position of the pieces placed in play. Allied to this process, there is an algorithm that based on the state of the game will calculate the best answer for the computer.

To obtain a correct collection of the game environment, image processing routines were used with methods to filter the noise that can be introduced by external variables or ambient changes and correctly identify the piece introduced in game.

After the correct analysis of the current state of the board, based on the calculated move for the computer the movement to be performed by the manipulator is determined using kinematics, after this the information needed to move the piece is translated using a system dedicated to its treatment and compatibility with the manipulator.

Then the previously determined movements of each joint are converted to analog values through an Arduino processor and sent to the manipulator.

Keywords: Human-Machine Interaction, Computational Vision, Arduino, Kinematics.

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer ao ISEL, pela importância que teve no meu desenvolvimento e pelas oportunidades que me foram dadas. A todos os professores e colegas, pelas histórias que se criaram e que foram partilhadas, pela aprendizagem e pelas amizades que acredito perdurarão muito para além do curto período em que compartilhámos este espaço.

À professora Doutora Maria da Graça Vieira de Brito Almeida, pela paciência, pela orientação e pelo tempo que despendeu durante o desenvolvimento deste trabalho.

Em particular ao meu colega Santo de Freitas, que me acompanhou durante parte do meu percurso enquanto aluno desta instituição, que ainda me acompanha enquanto profissional e que teve um papel muito importante na minha força motriz para desenvolver este projeto. Agradeço ainda o facto de ter colocado à disposição o manipulador utilizado para este trabalho.

Por último, aos meus avós que me proporcionaram as oportunidades necessárias para chegar a este ponto, que sempre me motivaram e tornaram todo este percurso possível, os meus eternos agradecimentos.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS

RGB – Sistema de cores *Red* (vermelho) *Green* (verde) *Blue* (azul)

HSV - Sistema de cores *Hue* (matiz), *Saturation* (saturação) e *Value* (valor)

Gdl – Graus de Liberdade

BUS – Barramento de ligação

CPU – *Central Processing Unit* (unidade de processamento central)

PWM – *Pulse Width Modulation* (Modulação por largura de impulso)

EPROM – *Eraseable PROM*

RAM – *Random Access Memory* (memória de leitura/escrita)

DRAM – *Dynamic RAM (*memória de leitura/escrita dinâmica)

SRAM – *Static RAM* (memória de leitura/escrita estática)

ROM – *Read Only Memory* (memória de leitura)

PROM – *Programmable ROM* (memória de leitura, permite programar quando criada)

EEPROM – *Eraseable PROM* (memória de leitura, permite apagar e reprogramar)

A/D – Analógico para Digital

D-H - *Denavit-Hartenberg*

URDF - *Unified Robot Description Format* (Formato Unificado de Descrição de Robôs)

CSV - Comma-Separated Values (valores separados por vírgula)

HMI – *Human Machine Interface* (Interface Homem-Máquina)

SIMBOLOGIA

m – Metros

Mpx – Megapixel

Px – Pixel

º - Graus

1. INTRODUÇÃO

1.1. Motivação

Desde o início da investigação científica a recolha visual de informação desempenha um papel crucial. No entanto, a mesma era inicialmente guardada através de descrições verbais ou documentos escritos. Com o aparecimento da fotografia tornou possível armazenar dados de forma objetiva. Como exemplo tem-se a astronomia e a justiça. No campo da astronomia foi possível medir a posição e dimensão de estrelas e outros corpos celestes. Na justiça sob a forma de prova o aparecimento de dados fotográficos permitiu a documentação de informação crucial de forma objetiva.

O processamento de imagem é extremamente importante por várias razões, seja de uma forma mais primária sendo o principal meio de recolha de informação utilizado por grande parte dos seres vivos, como meio de armazenamento de informação através de meios físicos ou digitais, através de fotografia ou vídeo, ou até como ferramenta de aquisição em vários sistemas de automação e controlo, tornou-se atualmente uma ferramenta essencial em diversas áreas de estudo.

A sua aplicação em diversas áreas permitiu tornar autónomos processos que até então necessitavam de intervenção humana para tratar a informação obtida. Esta ferramenta é atualmente crucial em diversos sistemas automatizados tendo um papel essencial para o seu funcionamento. No entanto, de forma a se conseguir aplicar este tipo de tecnologia é preciso ter sistemas altamente fiáveis e controlados de forma a blindar o sistema contra eventos que possam comprometer o seu funcionamento, uma vez que uma ação errada pode ter graves consequências. Para tornar o tratamento visual totalmente desprovido de interação Humana tem que se assegurar a segurança do meio em que este é aplicado.

O processamento de imagem adiciona uma mais-valia importante a processos automatizados uma vez que introduz flexibilidade e um ambiente de recolha muito superior a qualquer outro método de controlo, possibilitando a monitorização de eventos com carácter distinto através do mesmo ponto de recolha.

Aliando esta tecnologia a um robô manipulador é possível interagir com o ambiente através de controlo cinemático. Com recurso a este elemento é possível adicionar à rotina de recolha de informação diversas respostas com base em alterações ocorridas no sistema no decorrer do tempo do processo.

Desta forma com a realização do presente projeto conseguiram-se aliar diversas disciplinas cruciais no desenvolvimento de soluções de automação, flexíveis e eficientes, tecnologia de processamento de imagem com mecanismos de decisão e cinemática, aproveitando desta forma para aprofundar conhecimento em áreas extremamente importantes na indústria atual.

1.2. Objetivos

Com o desenvolvimento deste trabalho pretende-se aprofundar os conhecimentos obtidos nas unidades curriculares de sistemas robóticos e circuitos eletrónicos embebidos.

Este trabalho irá interligar diversas ferramentas o que permitirá verificar a interação entre as mesmas, nomeadamente o desenvolvimento de um sistema capaz de interagir com um utilizador através de processamento de imagem e manipulação da área de jogo através de um manipulador robótico.

Para conseguir este propósito será utilizada uma câmara para fazer a captura da área de jogo, a captura será posteriormente processada de forma a ser reduzido o ruído que possa ser introduzido por diversas fontes e também permite, em tempo real, obter informação do ambiente de trabalho. Após o processamento será utilizado um algoritmo de forma a calcular a melhor resposta ao movimento apresentado.

Para além disso será utilizado um microprocessador Arduíno que será o responsável pelo controlo do robô. Outro objetivo será consolidar ainda mais os conhecimentos na plataforma MATLAB.

Será utilizada a plataforma de programação MATLAB e integrada com todo o sistema, este elemento de *software* irá reunir as funções necessárias para o processamento de imagem, a cinemática, a comunicação com o microprocessador e o algoritmo de decisão. Através deste *software* será criada uma interface que permitirá ao utilizador introduzir alguns parâmetros necessários para o funcionamento bem como a visualização e interação durante o jogo.

Deste modo através de diversas áreas do conhecimento será possível criar um sistema que integra *software* com *hardware* e permite a interação com o utilizador de forma interativa. Atualmente é imprescindível a programação aliada aos diversos componentes e soluções disponíveis no mercado.

1.3. Organização do trabalho

Este trabalho encontra-se organizado em 5 secções.

Primeiro, na secção número 1, são apresentados os objetivos e organização do trabalho

Na secção seguinte, número 2, é apresentado o estado da arte onde se exploram as diversas áreas relacionadas com este trabalho.

O desenvolvimento da solução e os resultados deste trabalho são apresentados nas secções 3 e 4 respetivamente.

Por fim, na última secção, são apresentadas as conclusões e possíveis alterações e possíveis desenvolvimentos futuros a este trabalho

2. ESTADO DA ARTE

2.1. Introdução

No presente capítulo serão abordadas as principais áreas envolvidas nos assuntos associados ao desenvolvimento deste trabalho.

Como primeiro tema será abordada a área da captura e processamento de imagem, aqui o estudo será orientado para as técnicas relacionadas ao trabalho apresentado, uma vez este ser um tema muito abrangente e consoante a aplicação muitas soluções poderão ser utilizadas, será abordada a área da perceção visual humana como introdução ao tema do processamento na vertente computacional, bem como os métodos utilizados para a representação e condicionamento de imagem de forma digital sendo estes cruciais para a interpretação e desenvolvimento de programas e rotinas de deteção e processamento. Será também importante relacionar a imagem adquirida que dispõe de informação com duas dimensões e como introduzir a mesma num sistema de controlo com três dimensões.

Em seguida serão explicados alguns conceitos e modelos utilizados na robótica, caracterizando os robôs e enumerando as diferentes montagens e possibilidades de movimento para os mesmos.

Seguidamente será abordado o tema da cinemática, este será o veículo usado para integrar o processamento de imagem e a robótica de forma a compor um sistema interativo e com a capacidade de responder de forma autónoma ao utilizador, explicando de forma breve alguns métodos utilizados para determinar as coordenadas de posição que permitirão traduzir fisicamente a resposta do sistema.

Irá ainda ser explorado o tema dos microcontroladores com foco na plataforma Arduíno. Esta tecnologia será utilizada como meio de comunicação entre a estação de processamento de informação e o manipulador.

Por fim será introduzido o *software* utilizado como veículo de interligação entre todas as disciplinas, este permitirá relacionar o processamento de imagem, a cinemática, a comunicação com o microprocessador e o algoritmo de decisão, permitirá ainda o desenvolvimento de uma interface *HMI* (*Human Machine Interface)*, esta possibilitará a interação com o ambiente desenvolvido bem como a inserção de parâmetros necessários para o funcionamento. O *software* utilizado será o MATLAB, como principal fator para a sua utilização está a sua ampla biblioteca de funções e interoperabilidade, permitindo assim a ligação entre todos os elementos de forma eficaz.

2.2. Processamento de imagem

Grande parte das impressões do mundo e memórias são baseadas em informações obtidas através da visão. São obtidas através dum trabalho conjunto dos olhos e do encéfalo, desta forma a perceção atual é de que este é um processo ativo e criativo, ou seja, para além da informação que se recebe na retina existem mais processos envolvidos até à chegada da informação ao cérebro, Figura 1.

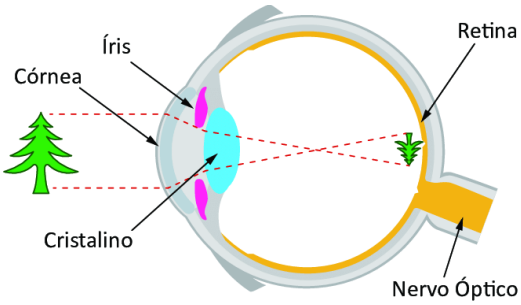


Figura 1 - Olho humano vs recolha de imagem [1]

O processamento de imagem na vertente computacional refere-se à manipulação e operação de imagens. A este tipo de tratamento e transformação de imagens aplicado à computação de forma inteligente, interpretativa e de modo a extrair informações, novas representações e conclusões a partir dos dados da imagem, dá-se o nome de visão computacional. Esta tecnologia é atualmente importante em diversas áreas da ciência, desde sistemas de reconhecimento de padrões, movimentos e expressões faciais, sistemas de realidade aumentada, até aplicações interdisciplinares abrangendo áreas como a medicina e a biologia.

Utilizando a recolha de informação por imagens, obtém-se um sinal bidimensional, isto é, são sinais no domínio espacial que variam no espaço e que podem ou não variar no tempo. Desta forma, as imagens digitais podem ser definidas como funções bidimensionais compostas por um conjunto de elementos finitos, normalmente denominados por *pixels*, estes possuem localizações e valores particulares. Para este trabalho são de especial interesse as imagens fotográficas, sendo também as mais semelhantes à visão humana, mas existem outras aplicações que utilizam imagens sintéticas, omnidirecionais, térmicas (ou que representem luz fora do espectro visível ao ser humano, entre outras).

2.2.1. Interpretação Digital de Imagens

Quando se obtém uma fotografia através de uma câmara digital, a imagem é durante este processo convertida do formato analógico para o formato digital, sendo este o formato visualizado nos computadores e aparelhos digitais. O sinal analógico será a luz incidente no obturador da câmara, que para ser lida por um aparelho digital necessita de ser convertido para uma sequência de valores binários ou *bits*. Este processo de conversão A/D (analógico para digital) é composto geralmente por três etapas: em primeiro lugar o sinal analógico é amostrado e o valor capturado é retido na forma de um sinal discreto, é nesta etapa que se define a resolução das imagens e, portanto, a quantidade de *pixels* a apresentar [2]. De seguida o valor é quantificado, aqui os valores obtidos são aproximados para um conjunto ou escala de valores finitos, determinando assim a profundidade da imagem digital resultante, ou qual o valor individual de cada *pixel*, sendo um exemplo deste processo as imagens em tons cinza, que utilizam 8 *bits* o que resulta em 256 níveis de cinza para cada *pixel* (ou 0 a 255), sendo os extremos as cores preto e branco. As imagens binárias, ou vulgarmente, preto e branco, estas apenas utilizam dois níveis, para a correta apresentação de imagens desta forma é necessário existir um padrão ou algoritmo de comparação que identifique as cores representadas e lhes atribua o valor adequado.

Existem ainda imagens com um espectro de cor mais alargado, que procuram representar de forma fiel a imagem obtida pelo estímulo causado no sistema visual humano em resposta à incidência de radiação eletromagnética visível, luz. Atualmente é utilizado um espaço de 24 *bits*, este permite a representação de cores diferentes.

Em qualquer das situações com vários níveis de cor, o sistema poderá ser representado em diversos formatos de cor:

- **RGB**: Este será possivelmente o espaço de cores mais comum. Baseia-se na visão humana e foi determinado através da utilização de três tipos diferentes de células foto-recetoras, cada uma com sensibilidade a diferentes frequências de luz. Cada uma destas células responde de forma mais intensa às frequências que representam as cores vermelho (*Red*), verde (*Green*) e azul (*Blue*), daí a denominação do sistema, RGB (*Red – Green – Blue*). Este utiliza cada um dos 3 *bytes* que constituem o *pixel* para determinar a intensidade da respetiva componente, Figura 2.

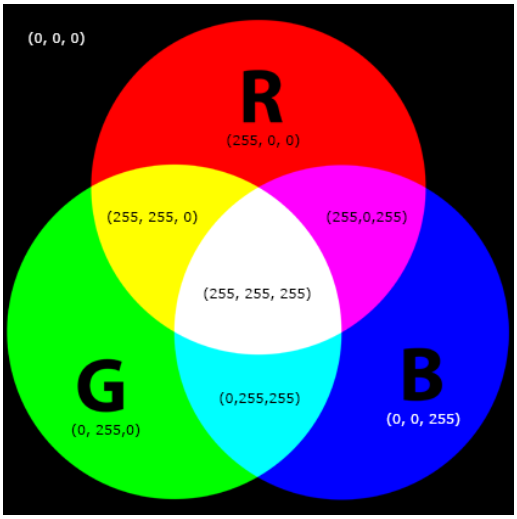


Figura 2 - Exemplo de demonstração de cor em sistema RGB [3]

• **HSV**: neste sistema o espaço de cores foi dividido nas componentes matiz (*hue*), saturação (*saturation*) e valor (*value*). A componente matiz representa a informação básica da cor, esta está compreendida entre 0 e 100%, e procura abranger todas as cores do espectro visível, a saturação introduz o tom cinza na imagem, desta forma escurece a mesma, com valores compreendidos entre 0 e 100% sendo 0 a cor na sua representação mais fiel, enquanto o valor estabelece o brilho da cor, com valores entre 0 e 100% os seus extremos representam as cores preto e branco.

Finalmente a imagem é codificada num determinado número de *pixels* correspondentes à resolução pretendida, Figura 3.

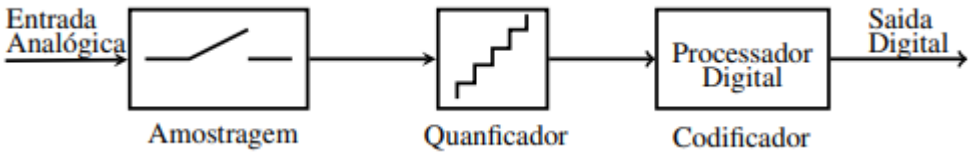


Figura 3- Arquitetura de um conversor A/D genérico

Não existe um método ideal para o tratamento da informação de forma a extrair informações de uma imagem, cada aplicação requer uma abordagem específica e contextualizada, à semelhança da visão humana. Para o trabalho realizado existem diversas técnicas interessantes para o desenvolvimento do mesmo. Sendo uma delas a aplicação de um tipo de operações designadas por operações espaciais. Operações espaciais referem-se a operações realizadas diretamente nos *pixels* de uma dada imagem, trabalhando assim no domínio espacial. Dentro destas técnicas existem operadores pontuais, realizados *pixel* a *pixel* e operadores não pontuais entre eles os estatísticos, onde usam funções de densidade e probabilidade.

Operadores pontuais:

Dentro dos operadores pontuais, pode-se citar algumas operações com histogramas de imagens, como limiarização (*tresholding*), Figura 4. Este é um método que através da segmentação de imagens se baseia na diferenciação dos níveis de cinza, desta forma recebe uma imagem composta por dois grupos de *pixels* um grupo composto pelos pixels com um nível de cinza acima do limiar e outro com nível inferior, sendo que todos os elementos de um mesmo grupo recebem um mesmo valor fixo.

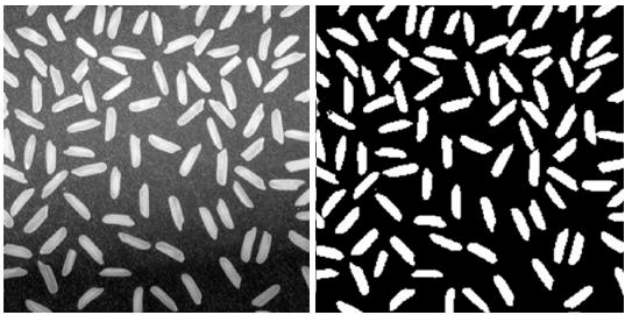


Figura 4 - Exemplo de aplicação da técnica de tresholding [4]

Operadores não pontuais:

Entre os operadores não pontuais existem operadores estatísticos, morfológicos, entre outros. Existem imensas possibilidades dentro deste tipo de operadores, desde transformações geométricas como rotação, escala e translação, a interpolação, dilatação. filtros de média e mediana que são muito usados para a harmonização da imagem ou remoção de ruído respetivamente. Como exemplo verifica-se na Figura 5 a aplicação do filtro de mediana com o intuito de remover o ruído existente na imagem original, este produz uma matriz com a vizinhança pretendida para o filtro, com a mediana dos valores dos elementos presentes nesta.

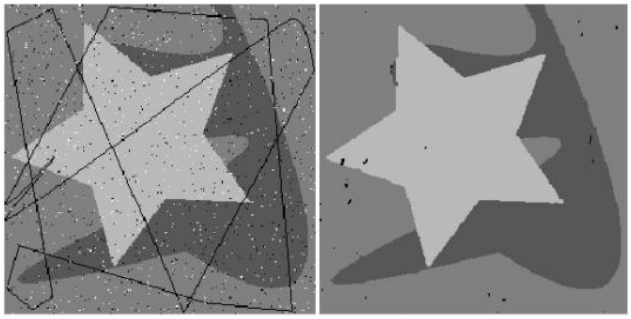


Figura 5 - Exemplo de aplicação do filtro de mediana [4]

Operadores de primeira e segunda ordem:

Existe ainda outro tipo de tratamento que se pode utilizar para extrair informações de uma imagem, pode-se por exemplo analisar as derivadas de sinal da imagem, de forma a determinar variações de gradiente ou contraste numa imagem, com isto consegue-se detetar extremos entre diversas regiões numa imagem, com este tipo de processamento consegue-se detetar bordas e cantos em imagem, onde uma variação no contraste implica uma derivada de valor elevado naquela região.

Utilizando este tipo de operadores de primeira e segunda ordem consegue-se detetar contornos e limites, sendo estes de grande interesse para a visão computacional aplicada à robótica. Como exemplo de métodos de deteção de limites com derivadas de primeira ordem, encontra-se o método de *Canny*, Figura 6 [5]. Existem também de forma similar outros métodos utilizados como o *Harris Corner Detector*, Figura 7, este método foi introduzido em 1988 por *Chris Harris* e *Mike Stephens* [6] como um melhoramento ao método já existente *Moravec's Corner Detector*, aqui são utilizadas derivadas para determinar vizinhanças onde exista mais do que um limite nas imediações.

Figura 6 – Demonstração de aplicação do método de Canny

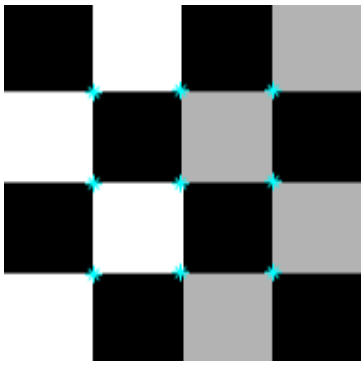


Figura 7 - Demonstração do método Harris Corner Detector [4]

Deteção de formas:

Existem ainda métodos utilizados para deteção de formas conhecidas como retas, círculos, elipses, quadrados, entre outras. Entre estes métodos a aplicação da transformada de *Hough*, Figura 8, é uma técnica que calcula a função de um conjunto acumulado de pontos previamente selecionados, como contornos de uma imagem, e que com esta informação permite determinar se se trata de uma linha ou um círculo

Figura 8 - Exemplo de aplicação da transformada de Hough [4]

Este método apesar de ter sido desenvolvido em 1959 com base na transformada de *Hough* apenas começou a ser aprofundado e utilizado a partir dos anos 2000 [3].

Pode-se também com base neste tipo de metodologias introduzir métodos atuais, que utilizam características genéticas e redes neuronais para desenvolver métodos de reconhecimento e deteção através de câmaras, estas tecnologias são muito utilizadas para proteção de privacidade, como exemplo de deteção e remoção ou pixelização de rostos e outro tipo de informação sensível que não se pretende que seja reconhecida ou até sistemas de autenticação que utilizem características físicas(biométrica) do utilizador para detetar a sua identidade, este tipo de sistemas podem utilizar métodos de treino de reconhecimento para melhorar a sua eficácia.

Assim, através do processamento de imagem é possível extrair informação de uma imagem e com base nessa informação controlar sistemas variados. Atualmente o processamento de imagem está interligado com outros sistemas como por exemplo sistemas de automação e robótica.

Aplicação em sistemas atuais:

Como exemplo de aplicação deste tipo de tecnologia no mercado atual a solução *OIT HL System* [7] utiliza um sistema de visão integrado num processo de produção. Esta permite verificar durante o processo de fabrico a qualidade do componente inspecionado, neste caso iluminação automóvel, sendo assim possível inspecionar e detetar qualquer defeito existente de forma célere e providenciar as ações necessárias para a correção das anomalias detetadas.

Na Figura 9 é possível verificar a célula onde está aplicado este sistema, trata-se de uma estação de injeção onde foram providenciadas as medidas necessárias para ser realizado o processo de inspeção visual.

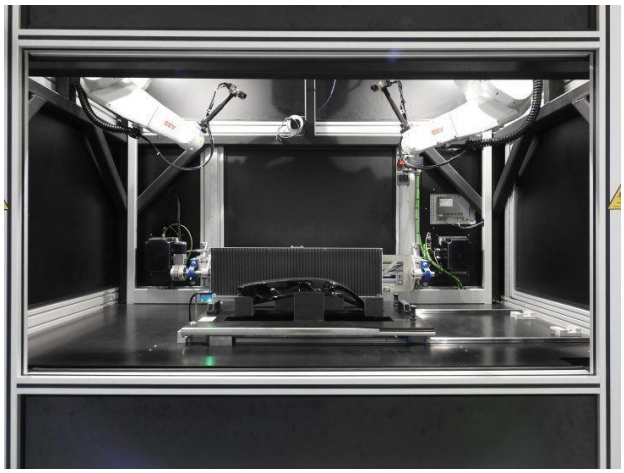


Figura 9 - Sistema OIT HL System [7]

2.3. Robótica e Automação Industrial

Com a evolução tecnológica, a Humanidade para além da inovação procurou também otimizar e automatizar processos que até então eram completamente dependentes de ação humana, esta transformação permite a supressão do erro humano no decorrer da tarefa e também a mobilização de esforços para outras áreas. A base desta metodologia assenta em reproduzir ações previamente realizadas por mão humana através de ferramentas ou máquinas desenvolvidas para realizar as mesmas de forma sistemática conseguindo assim eliminar o fator do erro humano no processo e associando tempos de processo muito inferiores aos previamente conhecidos. Assim é possível atingir índices de produtividade muito superiores, com custos inferiores e controlados, com níveis de segurança superiores e com uma redução muito significativa da mão de obra.

A introdução da automação nos processos industriais permitiu integrar diversas tecnologias de forma a otimizar cadeias de produção já implementadas, possibilitou a integração de sistemas eletrónicos, tanto a nível de *software* como *hardware*, elétricos e mecânicos.

O aparecimento do robô industrial deu-se no final dos anos setenta e teve como impulsionador *George Devol* com o robô *Unimate*, este é atualmente conhecido como o pai da robótica moderna tendo sido o *Unimate* o primeiro braço robótico programável reconhecido e que abriu portas para grandes avanços na indústria da robótica [8].

Em termos gerais, um robô manipulador é construído para mecanicamente conseguir posicionar uma garra ou ferramenta que se encontra no seu órgão terminal.

Particularmente um manipulador é normalmente constituído por um conjunto de corpos rígidos ligados por meio de juntas que podem ser rotativas ou prismáticas, formando uma cadeia cinemática. Desta forma a sua estrutura é composta por: **corpo**, **braço** e **punho**. Sendo as juntas os elementos que permitem a mobilidade do manipulador e o principal elemento diferenciador deste tipo de máquina. Pode-se definir as mesmas em 4 tipos essenciais, Figura 10:

- **Esférica (E):** permite uma rotação em torno de três eixos em simultâneo;
- **Torção (T):** O elo de saída possui um eixo de rotação paralelo a ambos os eixos e desenvolve um movimento linear;
- **Linear (L):** A orientação do movimento no elo de saída é igual à do elo de entrada;
- **Rotacionais (R):** O elo de saída permite uma rotação num eixo paralelo a ambos os elos;

Para além destas existem ainda dois tipos de utilização comum que se tratam de uma composição dos tipos anteriores, sendo elas:

- **Planar (P):** O elo de saída efetua um movimento num plano definido pelo elo de entrada e em duas direções, sendo composta por duas juntas prismáticas;

- **Cilíndrica(C):** O elo de saída efetua um movimento linear e rotacional, ambos com orientação e eixo do movimento paralelos ao elo de entrada, desta faz parte uma junta prismática e uma rotacional.

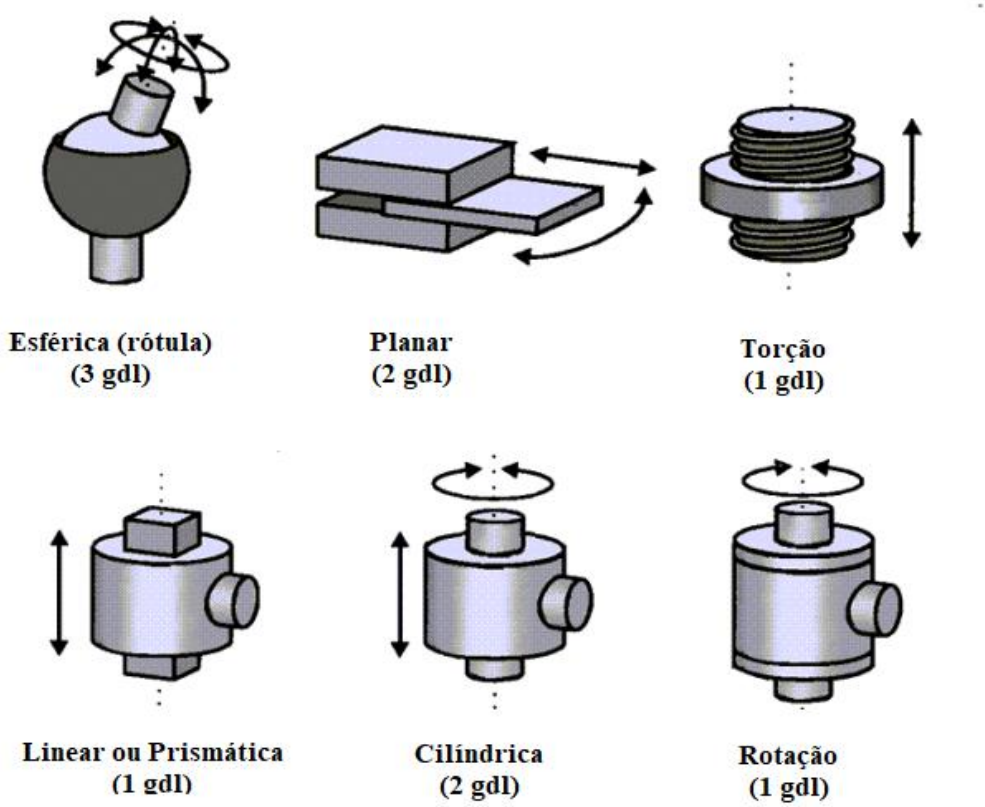


Figura 10- Tipo de juntas usadas em robôs manipuladores [9]

Como base para suportar os movimentos a executar a base que conjuntamente com o **corpo** do manipulador formam a estrutura de fixação do manipulador, sendo o seu papel principal garantir a estabilidade da máquina. Em seguida o **braço** e o **punho.** Consoante o número e tipo de juntas atribuídas ao manipulador, assim variar o seu número de **graus de liberdade** e **graus de movimento**.

Existem também movimentos associados ao punho do manipulador, estes movimentos podem ser equiparados aos eixos associados aos movimentos em aeronaves, dessa forma utilizam a mesma nomenclatura, Figura 11:

- Movimento "*Roll*" (Rotacional): o punho possui um eixo de rotação paralelo ao do braço, possibilitando um movimento de rotação;
- Movimento "*Pitch*" (Inclinação): neste caso o eixo de rotação é perpendicular ao braço do robô e orientado na horizontal, permite assim um movimento de inclinação vertical;

- Eixo "*Yaw*" (Orientação): este movimento possui características semelhantes ao movimento "*Pitch*" sendo também perpendicular braço do robô mas orientado na vertical, desta forma permitindo um movimento de inclinação horizontal.

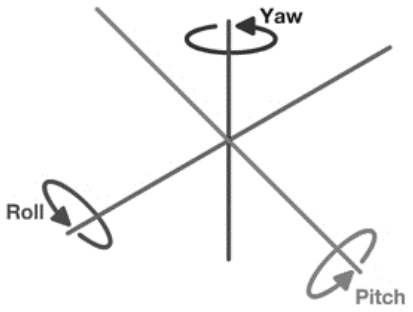


Figura 11 - Descrição movimentos punho [10]

Assim os Gdl, indicam as possibilidades de posicionamento e orientação no espaço, existem por exemplo 6 graus de liberdade no seu órgão terminal se tiver 3 graus em posicionamento e 3 graus em orientação 3D, Figura 12, esta será a configuração mais complexa. Relativamente aos seus graus de movimento estes existem sempre em número superior ou igual ao número de graus de liberdade, pode-se ter duas juntas a efetuar o mesmo movimento em posições distintas do manipulador atribuindo assim um número superior de graus de movimento ao manipulador em relação aos seus graus de liberdade. Será com base nos requisitos da aplicação para a qual está dimensionado o órgão terminal que se irá definir a relação entre graus de liberdade e de movimento para o manipulador.

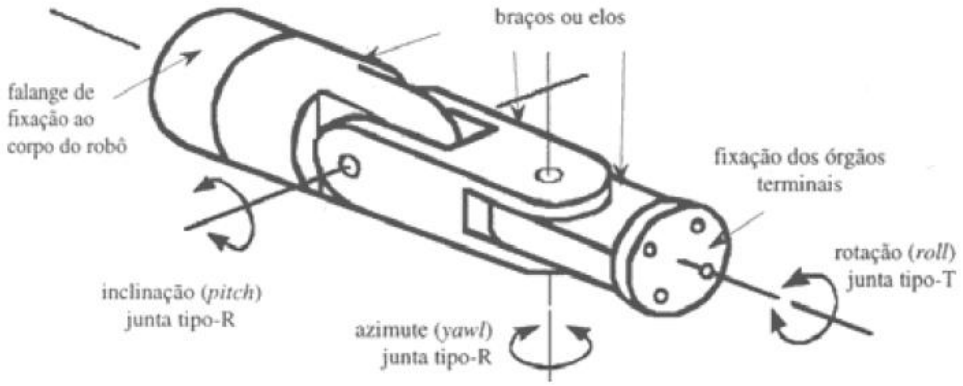


Figura 12 – Exemplo de punho com 3 graus de liberdade [11]

2.3.1. O Robô Industrial

Com a introdução da robótica em ambiente industrial apareceram como resposta soluções com diversas formas, configurações, tamanhos e controlo. Estes podem ser classificados quanto à sua estrutura mecânica, sendo os principais:

- Robô de Coordenadas Cartesianas/Pórtico;
- Robô de Coordenadas Cilíndricas;
- Robô de Coordenadas Esféricas;
- Robô SCARA;
- Robô Articulado ou Antropomórfico;

Robô de Coordenadas Cartesianas

É composto por três juntas lineares, desta forma o seu movimento é resultado da deslocação da base em três eixos. Assim, obtém-se um sistema de coordenadas cartesiano coincidente com os eixos de movimento onde o volume de trabalho gerado pelo robô é um paralelepípedo, Figura 13.

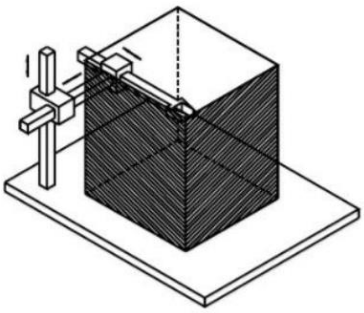


Figura 13 - Robô de Coordenadas Cartesianas/Pórtico [10]

Robô de Coordenadas Cilíndricas

É composto por duas juntas lineares e uma de rotação, assim o seu movimento pode ser decomposto em dois movimentos lineares e um de rotação. Com esta base de movimento obtém-se um sistema de coordenadas de referência cilíndrica, onde o volume de trabalho será um cilindro, Figura 14.

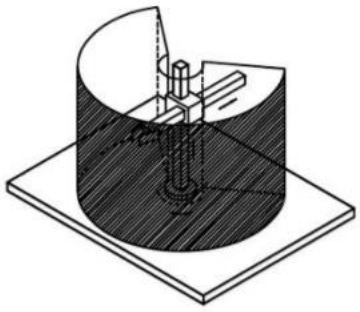


Figura 14 - Robô de Coordenadas Cilíndricas [10]

Robô de Coordenadas Esféricas

É composto por duas juntas de rotação com eixos distintos e uma junta linear, compreende assim um movimento linear e duas rotações. Desta forma o seu movimento pode ser compreendido num sistema de coordenadas de referência cilíndrica. Com esta constituição consegue-se um volume de trabalho esférico, Figura 15.

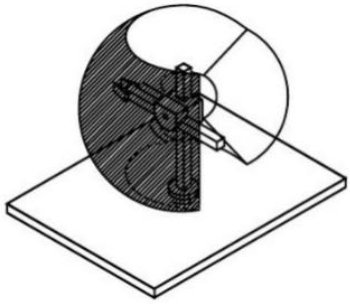


Figura 15 - Robô de Coordenadas Esféricas [10]

Robô SCARA ("*Selective Compliance Assembly Robot Arm*"**)**

É composto por duas juntas de rotação paralelas, que desta forma trabalham no mesmo plano, e uma junta linear perpendicular a ambas. Compreende assim um movimento linear e duas rotações, possuindo assim um sistema de coordenadas de referência cilíndrica. Possui um volume de trabalho cilíndrico.

Este tipo de robôs é muito utilizado em sistemas de processamento automatizado, devido à sua configuração tornam-se muito versáteis bem como geralmente mais rápidos e compactos que os que utilizam sistemas de coordenadas cartesianas, Figura 16.

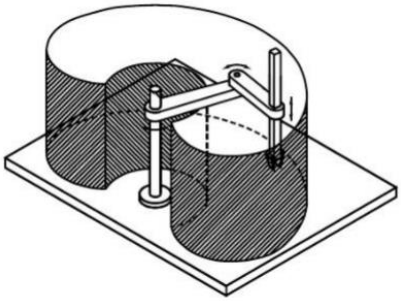


Figura 16 – Robô SCARA [10]

Robô Articulado ou Antropomórfico

É composto por pelo menos três juntas de rotação, utilizando exclusivamente este tipo de juntas e sendo a junta da base ortogonal às restantes juntas que compõem o braço, estas são simétricas entre si o que permite uma maior mobilidade ao robô. Possui um volume de trabalho variável e com geometria complexa, Figura 17.

Este tipo de robôs apesar do seu grau de complexidade mais elevado, possuem a maior flexibilidade em termos de movimento, conseguindo também elevada velocidade de operação apesar do número de juntas ser tipicamente superior aos restantes. Possibilitam também a utilização de estruturas cinemáticas abertas ou fechadas.

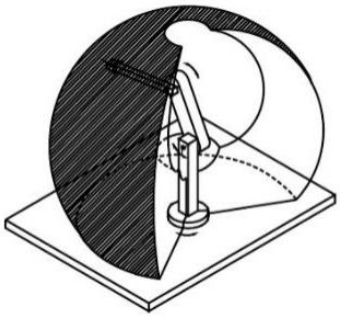


Figura 17 - Robô Articulado ou Antropomórfico [10]

Robô Delta

Existem ainda algumas configurações desenvolvidas com intuito de otimizar processos de produção existentes, estas são especificamente concebidas com a tarefa a desempenhar em mente. Como exemplo o robô de estrutura **Delta** *DR-3iB/8L* **(**Figura 18**)** utiliza elementos mecânicos com atuadores que trabalham em paralelo, tipicamente os fabricantes que optam por este tipo de configuração utilizam juntas rotativas ou prismáticas. Ao utilizar este tipo de configuração consegue-se:

- aumentar a capacidade de carga,
- aumentar a rigidez estrutural e construir um robô mais compacto,
- diminuir a sua inércia,
- melhorar a capacidade de posicionamento,

Figura 18 - Robô Delta FANUC - DR-3iB/8L [12]

2.3.2. Cinemática

Para integrar a robótica em processos automatizados é preciso considerar um aspeto muito importante, o movimento. Como fazer com que o robô se mova de acordo com a necessidade.

De encontro à necessidade de estudar o movimento dos corpos surgiu na física a mecânica, esta procura respostas para diversos temas cruciais associados a este tema, "*Como colocar um corpo em movimento? Como provocar a alteração do movimento do um corpo? O que está envolvido neste processo?"*, entre outras [8]. Desta forma surgiram três grandes disciplinas na mecânica que fundamentalmente respondem às três leis de Newton, a estática, a dinâmica e a cinemática. Como disciplinas que se complementam pode-se resumir cada uma delas da forma seguinte:

- Estática**:** se a resultante das forças que atuam num sistema for nula o sistema está em equilíbrio, assim a soma das forças a atuar no interior do sistema é igual à das que se encontram a atuar no seu exterior;
- Dinâmica**:** a taxa de variação da quantidade de movimento de um corpo no tempo é igual à soma das forças aplicadas nesse corpo;
- Cinemática**:** com base na posição de referência e nas dimensões associadas ao movimento consegue-se em qualquer instante determinar a posição de um ponto do corpo, a sua orientação em torno de um eixo fixo e a rotação em torno desse eixo.

Assim, para se poder aplicar movimento ao robô é preciso de saber a sua posição e para onde se pretende que o mesmo se mova. Para isso é preciso então aplicar conceitos de cinemática ao sistema.

Com isto em mente existem na cinemática dois métodos que se podem aplicar, a cinemática direta onde para uma determinada posição e orientação das juntas do robô ir-se-á determinar a posição e orientação do respetivo punho, e a cinemática inversa onde se fará o caminho contrário, ou seja com base na posição e orientação do punho determinar-se-á a posição e orientação das juntas respetivas.

Cinemática Direta:

Sendo um manipulador composto por uma sequência de corpos rígidos (elos) ligados por juntas, para se poder aplicar estes métodos é necessário começar por enumerar os pares junta-elo do manipulador (juntas ***1 a n*** e elos ***0 a n***).

Para o processo de cinemática direta começar-se-á com a base, é o elemento fixo do sistema e assim será a origem (elo ***0***) e o punho será a junta de maior índice (***n***). [11]

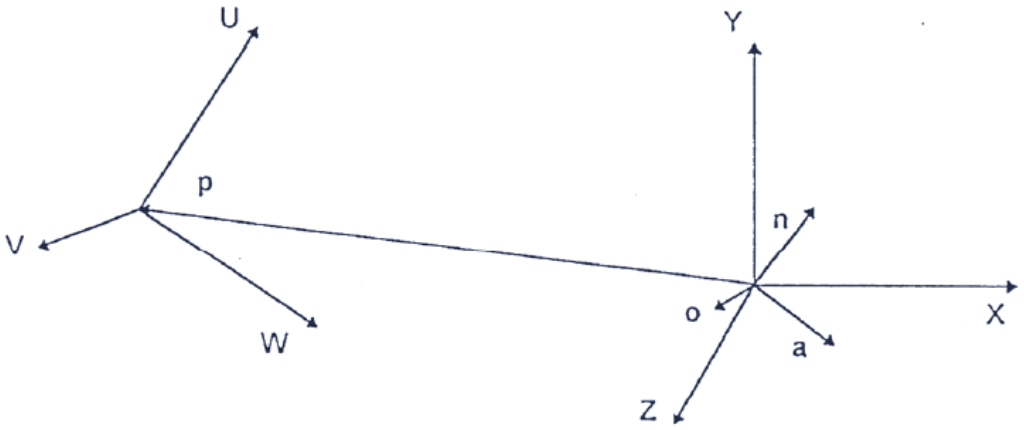


Figura 19 – Demonstração de referencial fixo OXYZ e referencial móvel OUVW [11]

Com base no descrito ir-se-á utilizar um sistema de coordenadas $O_nX_nY_nZ_n$, este será o referencial fixo. Relativamente a este referencial posicionar-se-ão os referenciais móveis $O_nU_nV_nW_n$, estes serão colocados nos restantes elementos do corpo do robô, Figura 19. A posição e orientação de cada elemento deste sistema será determinada com base nas coordenadas do elemento de índice *n-1*, para isso serão utilizadas matrizes de transformação homogéneas, estas podem ser definidas como as matrizes de transformação de um sistema de coordenadas $O_nU_nV_nW_n$ para $O_nX_nY_nZ_n$.

$$P_{XYZ} = HP_{UVW} \qquad (1)$$

Onde,

$$P_{XYZ} = [p_X\ p_Y\ p_Z]^T \qquad (2)$$

$$H = \begin{bmatrix} n_X & o_X & a_X & p_X \\ n_Y & o_Y & a_Y & p_Y \\ n_Z & o_Z & a_Z & p_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3)$$

Em que:

- **n**= representa um vetor perpendicular ao órgão terminal.
- **o** = representa um vetor que aponta na direção do movimento de ação da ferramenta do órgão terminal.
- **a** = representa um vetor paralelo ao eixo do órgão terminal.
- **p** = vetor da base do referencial para a origem do órgão terminal.

E,

$$P_{UVW} = [p_U \; p_V \; p_W]^T \qquad (4)$$

Decompondo a matriz homogénea apresentada em (3) obtém-se: nas 3 primeiras linhas e colunas, o versor o_U, o versor o_V, o versor o_W e a origem do sistema de coordenadas O_{UVW} respetivamente no sistema de coordenadas O_{XYZ} ; em seguida na última linha obtêm-se nos primeiros 3 elementos a relação de transformação de perspetiva entre os sistemas de coordenadas, e por último o fator de escala entre os sistemas.

Existe ainda uma transformação a considerar, se o referencial $O_n U_n V_n W_n$ sofrer uma rotação de α segundo o eixo O_X então o ponto no referencial O_{UVW} segundo O_{XYZ} terá coordenadas diferentes, para contemplar esta rotação ter-se-á de aplicar uma matriz de rotação $R_{X,\alpha}$. Desta forma tem-se que:

$$P_{XYZ} = R_{X,\alpha}[p_U \; p_V \; p_W]^T \qquad (5)$$

Com a rotação segundo O_X , e onde $(i_X, j_Y, k_Z)\; e\; (i_U, j_V, k_W)$ representam vetores unitários segundo os eixos O_{xyz} e O_{uvw} respetivamente, obtém-se:

$$R_{X,\alpha} = \begin{bmatrix} i_X.i_U & i_X.j_V & i_X.k_W \\ j_Y.i_U & j_Y.j_V & j_Y.k_W \\ k_Z.i_U & k_Z.j_V & k_Z.k_W \end{bmatrix} \qquad (6)$$

com $i_X = i_U$:

$$R_{X,\alpha} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \qquad (7)$$

É possível de modo semelhante obter as equações de rotação segundo $O_Y (i_Y = i_V)$ para um ângulo φ e O_Z $(i_Z = i_W)$ para um ângulo θ:

$$=R_{Y,\varphi}=\begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix} \quad (8)$$

$$=R_{Z,\theta}=\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (9)$$

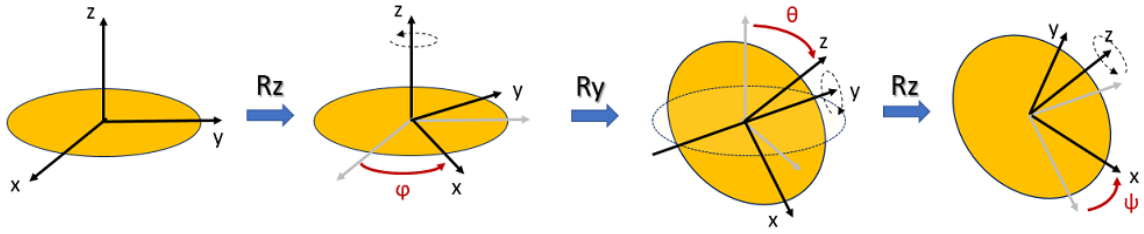


Figura 20 – Exemplo de aplicação de rotações sucessivas [13]

Em suma pode-se enumerar algumas propriedades das matrizes referentes à rotação de um corpo, Figura 20, segundo a cinemática direta:

- as colunas da matriz de rotação aplicada representam os eixos do referencial móvel com referência no referencial fixo do sistema, e as linhas de forma inversa representam os eixos do referencial fixo com referência no referencial móvel do sistema;
- sendo cada linha e coluna representada por vetores unitários (módulo igual a um) o determinante desta matriz será também unitário;
- o produto interno de duas linhas ou duas colunas pertencentes a este tipo de matriz é nulo;
- a inversa desta é igual à sua transposta.
-

Consegue-se desta forma partir da posição base do sistema, chegar à extremidade do mesmo. Para descrever as relações que existem de translação e rotação entre dois elos adjacentes o método *Denavit-Hartenberg*, criado em 1955 estabelece um conjunto de regras para orientação dos referenciais das juntas ao longo da cadeia cinemática [11].

De forma simplificada este baseia-se no facto de que para determinar a posição relativa de duas retas no espaço, apenas são necessários 2 parâmetros, em que um é a distância medida ao longo da normal comum entre as duas retas e o seguinte é o ângulo em torno da mesma que uma das retas deve girar de forma a ficar paralela à outra.

Assim sendo, se para definir a posição relativa de duas retas no espaço se utilizam dois parâmetros, para definir a posição relativa de dois sistemas de coordenadas ir-se-á necessitar

de quatro. Pode-se chegar a esta conclusão da seguinte forma: para dois sistemas de três eixos, para determinar a posição relativa entre dois dos eixos de coincidentes de cada um ir-se-ão utilizar dois parâmetros, após conhecer estes e pelas condições de ortogonalidade e regra da mão direita consegue-se imediatamente definir o terceiro. Com isto para descrever a posição relativa entre dois sistemas de coordenadas é preciso de definir a posição relativa de dois dos seus eixos.

Em suma vai-se ter por cada junta quatro parâmetros que irão fazer parte da tabela de parâmetros *Denavit-Hartenberg*, correspondentes aos ângulos e distâncias entre os respetivos referenciais que foram estabelecidos, Figura 21.

Este método utiliza uma representação baseada em transformações homogéneas, onde que cada uma exprime um referencial em relação ao anterior, sendo que cada referencial está associado a um elo.

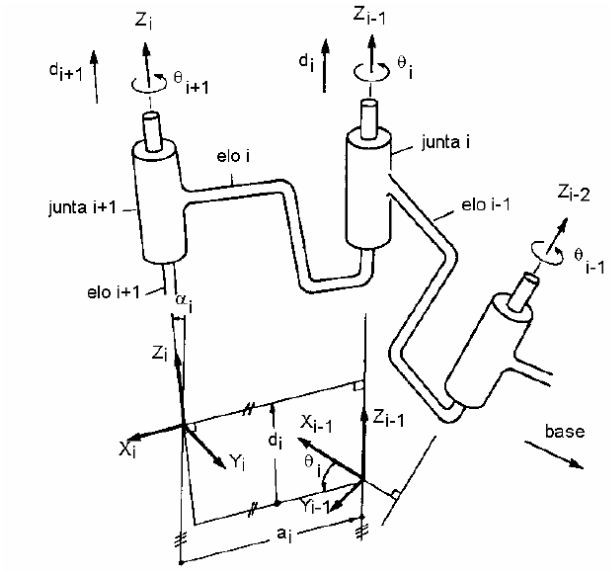


Figura 21 –Sistema de Coordenadas para Cinemática Direta [11]

Ao multiplicar todas as matrizes que fazem parte do sistema será possível obter a relação necessária para determinar a posição do órgão terminal expressa em relação ao sistema de eixos da base, o qual pode constituir o referencial de inércia do sistema.

Assim, o método *Denavit-Hartenberg (D-H)*, pode ser aplicado da seguinte forma [11]:

1. *Estabelecer o referencial base* (X_0, Y_0, Z_0). Este deverá ser colocado na base de suporte do robô, sendo o eixo Z_0 coincidente com o eixo da junta da base. Os restantes eixos são colocados de acordo com a "regra da mão direita" ou outra, caso seja mais conveniente, mantêm sim independentemente da sua colocação uma relação de perpendicularidade com Z_0;
2. *Estabelecer o eixo* Z_i. De acordo com o número de juntas presentes ir-se-á fazer coincidir o eixo Z_i com o eixo da junta $i + 1$;

3. *Estabelecer o referencial i*. Colocar na intersecção dos eixos Z_i e Z_{i-1} ou na intersecção perpendicular aos mesmos a origem do referencial i;

4. *Estabelecer o eixo* X_i. De acordo com $X_i = \pm\frac{(Z_{i-1}\times Z_i)}{\|(Z_{i-1}\times Z_i)\|}$, ou caso Z_i e Z_{i-1} sejam paralelos o eixo deverá ser colocado segundo a perpendicular comum entre os mesmos;

5. *Estabelecer o eixo* Y_i. De acordo com $Y_i = \frac{(Z_{i-1}\times Z_i)}{\|(Z_{i-1}\times Z_i)\|}$, este deverá completar o referencial da forma mais conveniente ou de acordo com a "regra da mão direita";

6. *Estabelecer o referencial terminal.* Colocar X_n de modo que seja perpendicular a Z_{n-1}, caso a última junta seja uma junta rotativa os eixos Z_n e Z_{n-1} deverão ser coincidentes. Assim o eixo Y_n deverá completar o referencial da forma mais conveniente ou de acordo com a "regra da mão direita";

7. *Determinar* d_i. Para as juntas de i_1 a i_{n-1} calcular segundo Z_{i-1} a distância da origem do referencial $i-1$ até à intersecção dos eixos Z_{i-1} com X_i. No caso de se estar perante uma junta linear este valor é variável;

8. *Determinar* a_i. Para as juntas de i_1 a i_{n-1} calcular segundo X_i a distância desde a intersecção dos eixos Z_{i-1} com X_i;

9. *Determinar* θ_i. Para as juntas de i_1 a i_{n-1} calcular segundo Z_{i-1} o ângulo entre X_{i-1} e X_i. No caso de se estar perante uma junta rotativa o valor é variável;

10. *Determinar* α_i. Para as juntas de i_1 a i_{n-1} calcular segundo X_i o ângulo entre Z_{i-1} e Z_i.

11. *Estabelecer a tabela de parâmetros D-H.*

Findo este processo serão, para cada uma das juntas, extraídos os 4 parâmetros que farão parte da tabela de parâmetros *D-H*, Tabela 1 – Exemplo tabela de parâmetros D-H.

Tabela 1 – Exemplo tabela de parâmetros D-H

Junta i	$\boldsymbol{\theta_i}$ (º)	$\boldsymbol{\alpha_i}$ (º)	$\boldsymbol{a_i}$**(mm)**	$\boldsymbol{d_i}$**(mm)**
J_1	θ_1	α_1	a_1	d_1
J_{n-1}	θ_{n-1}	α_{n-1}	a_{n-1}	d_{n-1}

Com a utilização destes parâmetros é possível obter a matriz homogénea que relaciona um referencial com o seu adjacente através da fórmula (10).

$$^{i-1}A_i = R(Z_{i-1},\theta_i).T(0,0,d_i).T(a_i,0,0).R(X_i,\alpha_i) = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

A escolha da origem do referencial base apenas tem que garantir que o eixo $\boldsymbol{Z_0}$ coincide com o eixo da primeira junta, de forma semelhante o último referencial pode ser colocado em qualquer ponto do órgão terminal, tendo apenas que assegurar a perpendicularidade entre $\boldsymbol{X_n}$ e $\boldsymbol{Z_{n-1}}$.

Equações da Cinemática Direta:

Com a tabela de parâmetros *D-H* preenchida e as matrizes de transformação homogénea determinadas pode-se então finalmente determinar a matriz $^{0}\boldsymbol{T_i}$. Esta será a responsável por especificar a localização do referencial *i* em relação ao referencial da base.:

$$^{0}\boldsymbol{T_i} = {}^{0}\boldsymbol{A_1}.{}^{1}\boldsymbol{A_2} \ldots {}^{i-1}\boldsymbol{A_i} \quad (11)$$

$$^{0}\boldsymbol{T_i} = \prod_{j=1}^{i} \boldsymbol{A_j} \text{ , para } \boldsymbol{i = 1, 2, \ldots, n} \quad (12)$$

$$= {}^{0}\boldsymbol{T_i} = \begin{bmatrix} X_i & Y_i & Z_i & p_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

Onde:

- $[X_i \quad Y_i \quad Z_i]$ = matriz que representa a orientação do referencial *i*, em relação à base.
- $[p_i]$ = vetor com origem no referencial base que aponta para a origem do referencial *i*.

Assim, consoante o número de Gdl do manipulador, a solução das equações de cinemática direta é conseguida pelo cálculo da matriz $\boldsymbol{T = {}^{0}A_i}$. É, no entanto, importante perceber que a matriz $\boldsymbol{T}$ é única para um dado sistema de referenciais, e que este sistema deve ser estabelecido com base do algoritmo ***D-H***.

Para uma utilização eficiente da matriz $\boldsymbol{T}$ em tempo real é conveniente encontrarem-se métodos de cálculo computacionais otimizados para o processo.

Cinemática Inversa:

Partindo do princípio da lógica da cinemática direta, a cinemática inversa procura calcular a posição em que vão estar as juntas do robô ao ser conhecida uma determinada posição. Este envolve, no entanto, cálculos consideravelmente mais complexos. Esta diferença deve-se principalmente:

- pode-se ter mais que uma solução para o mesmo manipulador, sendo que o número das mesmas aumenta com o número de graus de liberdade do manipulador;
- as equações envolvidas no processo não são lineares, pelo que as resoluções nem sempre permitem uma conclusão com processos analíticos;
- caso a posição especificada se encontre foram do espaço de trabalho pode não se conseguir determinar uma solução;

Por estes motivos existem diversas formas para solucionar este problema, sendo que para isso pode-se utilizar dois tipos de métodos, analíticos ou iterativos.

Quando possível a utilização de métodos analíticos torna o processo mais simples, no entanto só permitem a resolução de problemas de natureza mais simples, nomeadamente:

- eixos com três juntas rotativas consecutivas e que se intersectam;
- eixos com três juntas rotativas paralelas.

Os métodos iterativos que apesar de abordarem problemas mais complexos apenas permitem encontrar uma das possíveis soluções.

Dificuldades dos métodos utilizados para cinemática inversa:

Os métodos utilizados para resolução dos problemas apresentados requerem sempre uma análise da sua aplicação, sendo o principal motivo a existência de diversas soluções para cada caso.

Assim, depois de calculadas as soluções podem ser tomadas algumas decisões:

- se apenas algumas ou alguma das soluções calculadas se apresentarem dentro do intervalo de variação, a dimensão do problema é reduzida e mantêm-se apenas as que cumprem os limites;
- se todas as soluções calculadas se apresentarem dentro dos limites de variação, então o problema mantém a dimensão inicial calculada;
- se todas as soluções calculadas se apresentarem fora dos limites de variação, então a posição escolhida para mover o manipulador encontra-se fora do espaço de trabalho do mesmo.

Estando perante um caso que apresente solução para a posição pretendida, a escolha tenderá sempre para a que for mais vantajosa para o problema, isto é, que minimize o erro entre o vetor de coordenadas do espaço de juntas atual e o da próxima solução.

2.4. Microcontroladores

A tecnologia apresentada tem como raiz do seu desenvolvimento os circuitos integrados, estes foram essenciais para o desenvolvimento dos microprocessadores uma vez que a sua aparição tornou possível a coexistência de milhares de transístores num único chip. Desta forma, para compreender o funcionamento de um microcontrolador é necessário introduzir um elemento crucial no seu desenvolvimento, o microprocessador.

O aparecimento do microprocessador é potenciado por uma equipa de engenheiros japoneses pertencentes à empresa BUSICOM, estes propuseram em 1969 à INTEL uma encomenda de circuitos integrados para implementar em calculadoras. Esta ao chegar às mãos do responsável pela mesma, *Marcian Hoff*, tomou um rumo diferente do sugerido, isto pois com a experiência que o mesmo já possuía de trabalho com uma solução de um computador (*PDP-8*), surgiu a ideia de armazenar no circuito integrado o programa que iria determinar a sua função. Desta forma a solução tornar-se ia mais simples apesar de necessitar de muito mais memória que a solução sugerida pela equipa japonesa. A equipa japonesa acabou por se render à solução apresentada por *Marcian Hoff* e foi assim que nasceu o primeiro microprocessador [14], Figura 22.

Figura 22- Marcian Hoff, inventor do microprocessador [15]

Com conhecimento do potencial da sua descoberta a INTEL comprou a licença de venda da sua solução à BUSICOM. Pouco tempo depois, em 1971, aparece no mercado o primeiro microprocessador, com o desígnio 4004. Este foi o primeiro da sua história, possuía 4 bits e era capaz de 6000 operações por segundo [14].

No entanto, sendo o microprocessador fundamental para o funcionamento de um microcontrolador este não depende apenas da sua capacidade de processamento, assim a diferença mais aparente entre os dois é a inexistência de componentes periféricos num sistema microcontrolador, isto é, para desempenhar a sua função este não necessita de qualquer outro componente, o microcontrolador possui todas as funcionalidades necessárias em si mesmo.

Atualmente a arquitetura mais popular utilizada no desenvolvimento de microcontroladores é a arquitetura de *Harvard* [16], esta utiliza dois **BUS** dados distintos para transferências de memória, um para instruções e outro para dados permitindo assim a operação simultânea destes.

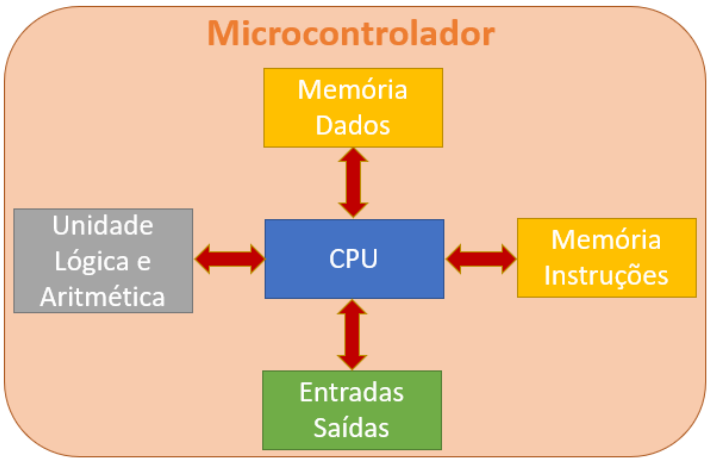


Figura 23 - Exemplo segundo Arquitectura de Harvard para microcontrolador

Ao analisar as soluções existentes no mercado atual verifica-se que existem imensos fabricantes a apostar nesta tecnologia, devido ao seu potencial e flexibilidade. Ainda assim é possível identificar as soluções *Arduino* como principal escolha neste tipo de sistemas. Sendo a arquitetura mais comum o *Arduino Uno* Figura 24.

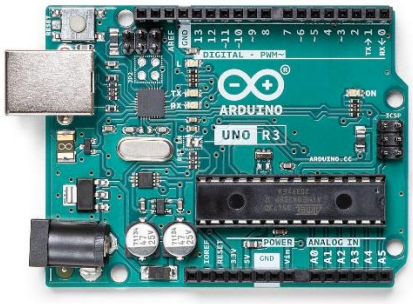


Figura 24 - Arduino Uno [17]

Este microcontrolador com o seu tamanho extremamente compacto possui, entre outras características, 14 portas binárias configuráveis, onde é possível escolher de forma independente se vão ser utilizadas como entradas ou saídas, 6 entradas analógicas e uma porta de comunicação USB.

2.4.1. Processamento

Para realizar operações com a informação disponível, é necessário um elemento capaz de processar esses mesmos blocos de dados. Para isso é necessária uma unidade de processamento central, esta é designada por **CPU** (*Central Processing Unit)*, Figura 25. Esta está constantemente a seguir as instruções que recebe e executa as mesmas. Assim, o CPU possui memória, no entanto esta é apenas utilizada para armazenar informação para a execução dos processos a decorrer, estas unidades de memória contidas no CPU são designadas por registos.

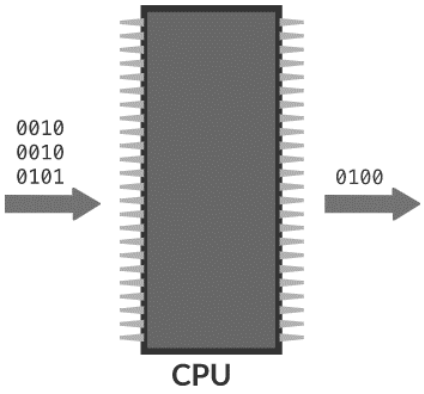


Figura 25 - Processamento em unidade CPU [18]

2.4.2. Módulo de entradas e saídas

No entanto, a unidade possui também a possibilidade de trabalhar com informação exterior. Enviar ou receber informação do exterior, sendo estas de outras unidades ou componentes. Assim, existem localizações dedicadas a estas funções, cada uma com uma função diferente e associada ao tipo de entrada ou saída. Podem-se enumerar os seus diferentes tipos da seguinte forma:

Entradas binárias:

Estas podem ser caracterizadas de forma simples por uma entrada capaz de identificar o valor de tensão aplicado na mesma. Correspondente ao nível lógico "1" ou "0", "1" se o valor de tensão na entrada estiver compreendido na sua banda limite superior e "0" se estiver na banda inferior, Figura 26.

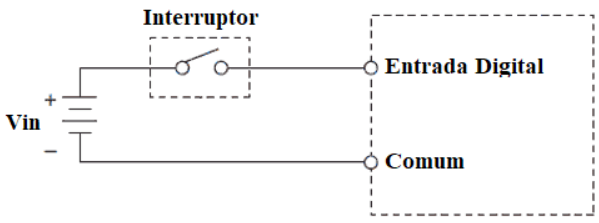


Figura 26 - Exemplo de Entrada Digital

Saídas binárias:

Estas de forma semelhante a uma entrada binária atuam com base no seu valor lógico. Comutadas tipicamente por um sistema de interruptor, seja por exemplo um transístor ou um relé. Este irá abrir ou fechar o seu circuito associado dependendo do estado lógico atribuído à saída, Figura 27.

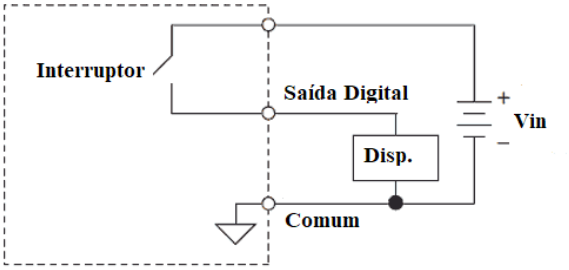


Figura 27- Exemplo de Saída Digital

Entradas analógicas:

Os sinais analógicos caracterizam-se por serem mensuráveis e possuírem um valor definido entre uma gama de valores, Figura 28. Uma vez que o microcontrolador trabalha com dados digitais o sinal analógico recebido terá que ser convertido, para isso é utilizado um conversor analógico/digital. Este será responsável pela conversão de todos os sinais de origem analógica para valores binários, permitindo assim o processamento pelo microcontrolador

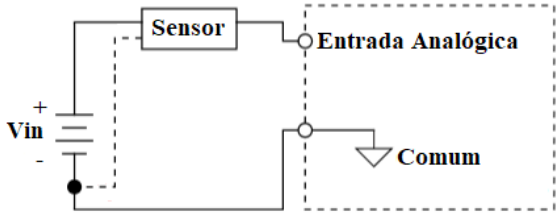


Figura 28- Exemplo de Entrada Analógica

Saídas analógicas:

As saídas analógicas são por definição sinais analógicos mensuráveis que utilizam uma gama previamente dimensionada e gerados pelo controlador, Figura 29. Estas saídas, no entanto, são obtidas no microcontrolador através de variações no valor de uma saída binária, utilizando modulação por largura de pulso (*PWM).*

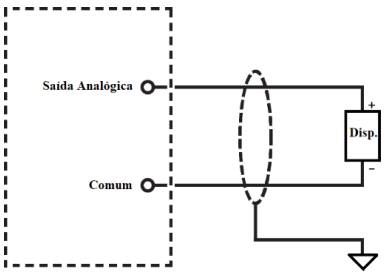


Figura 29-Exemplo de Saída Analógica

PWM (Pulse Width Modulation):

Os sinais PWM são baseados no conceito de distribuir um valor analógico definido numa base de tempo fixo e em quantidades de 0% a 100% da base de tempo utilizada (***Duty Cicle***). Desta forma consegue-se reduzir a potência entregue pelo sinal à carga mantendo o sinal de saída desejado, Figura 30.

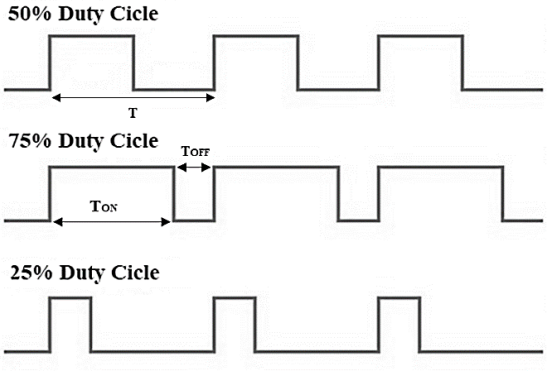


Figura 30 - Exemplo Sinais PWM

Entradas/Saídas específicas:

Contadores de impulsos:

Estas são entradas com o intuito de contabilizar o número de impulsos com determinada característica na entrada utilizada. São comummente utilizadas em contadores de caudal, onde o número de pulsos que se recebe na entrada corresponde à quantidade de caudal medido pela turbina, com uma relação do número de pulsos por unidade de tempo, sendo o número de pulsos proporcional à taxa de caudal utilizada. Utilizando a mesma metodologia existem outras aplicações para este tipo de entradas, Figura 31.

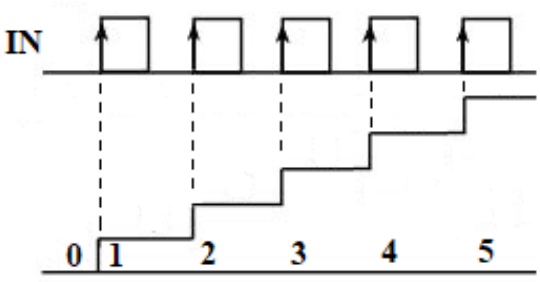
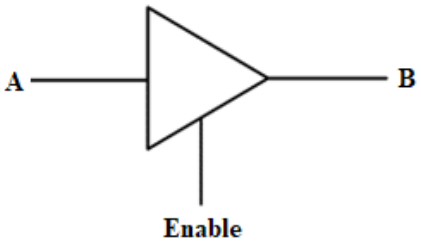


Figura 31 - Exemplo Contador Pulsos

Sinais com três estados:

Estes sinais consistem na utilização de dois sinais digitais. Com esta metodologia consegue-se utilizando dois sinais com uma característica binária fornecer três comandos a um dispositivo a controlar, adicionando assim um estado de alta impedância (Z), correspondente ao estado desabilitado da porta lógica. Estes são muito utilizados para controlar de forma modelada atuadores de válvulas. Desta forma consegue-se controlar os movimentos em ambas as direções do atuador de forma independente bem como a sua paragem, Figura 32.



Enable	A	B
0	0	Z
0	1	Z
1	0	0
1	1	1

Figura 32 - Exemplo Sinal de Três Estados

2.4.3. Comunicação

No seguimento da introdução das diferentes possibilidades de comunicação com o exterior é possível identificar um dos seus inconvenientes, a transmissão dos mesmos. A nível de economia e arquitetura do próprio sistema seria incomportável utilizar uma linha para cada dispositivo inserido, para resolver este problema são utilizados protocolos de comunicação, sendo o mais usual a utilização do mesmo número de linhas para todos os dispositivos. Ainda assim, existem algumas regras a implementar, de forma a garantir uma comunicação fiável e sem colisão de dados, ao conjunto das mesmas chama-se protocolo, sendo uma das regras principais a **codificação.**

Como exemplo de um tipo de codificação utilizada para comunicação existe a **NRZ** (*Non Return to Zero*), Figura 33. Esta utiliza o nível lógico "1" como estado de espera para que inicie a transmissão, sendo a condição para início da transmissão a permanência da linha no nível lógico "0" durante um período definido, assim, sempre que este nível lógico for detetado na linha durante o período definido sabe-se que existem dados a ser transmitidos na linha e é iniciada a receção dos mesmos. Do lado do emissor ir-se-á após o envio da mensagem colocar no último bit da mensagem, no caso de uma mensagem de 8 bits no oitavo bit, o nível lógico de espera, o nível "1", que simboliza o fim da transmissão.

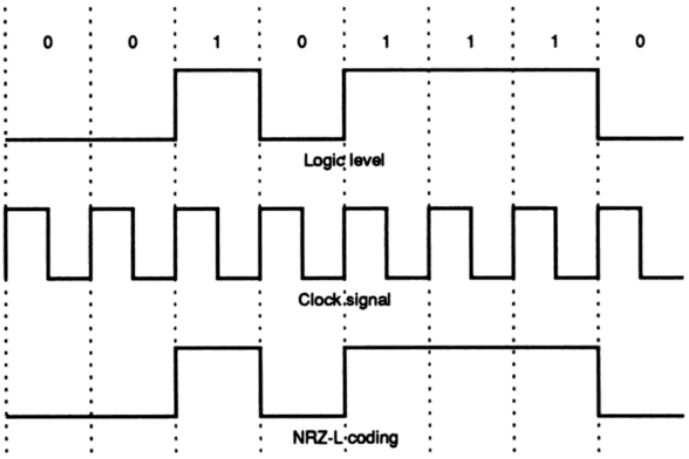


Figura 33- Exemplo de mensagem em codificação NRZ L [19]

Esta técnica de codificação é utilizada na comunicação do tipo **série**, isto é, os bits são enviados de forma sequencial no mesmo canal de comunicação. Este é o tipo de comunicação mais utilizada atualmente uma vez que diminui o custo dos canais de transmissão, podendo ser utilizado o mesmo canal de transmissão para todos os elementos da linha, facilitando desta forma a sincronização entre equipamentos bem como a introdução de novos elementos na linha. Como exemplo de protocolos bastante utilizados o **RS-232**, **UART**, entre outros.

Contudo, existe um elemento crucial para ser possível a temporização e sincronização dos elementos associados ao microprocessador, a unidade de **temporização**, este é na realidade um contador que irá dar informações sobre a hora, duração, entre outros elementos associados a diversos processos que contenham elementos temporais. Desta forma é responsável pela introdução de uma dimensão real a todos os processos a decorrer, o **tempo**.

2.4.4. Watchdog

Uma condição essencial para o programa será a correta execução e fluência do mesmo, desta forma será necessário perceber se a execução está a ocorrer de forma planeada ou se houve algum erro ou imprevisto e o mesmo entrou num estado imprevisto ou de falha.

Como responsável desta função é utilizado um **watchdog**, este é de forma simples um contador com uma característica particular, será colocado a zero sempre que o programa efetuar um novo ciclo. Desta forma, sempre que o contador não for colocado a zero irá atingir o seu valor máximo e colocar uma ***flag*** no programa, esta será usada para enviar o programa para a sua rotina de recuperação. Isto irá permitir ao programa efetuar o seu próprio ***reset*** sem intervenção humana.

2.4.5. Memória

Como já descrito, o microcontrolador possui a capacidade de armazenar o programa e outras funcionalidades inerentes ao desempenho das suas funções. Estes dados são armazenados na memória do dispositivo.

A memória de um dispositivo é o local onde são armazenados todos os dados do mesmo, assim, é natural também que seja o centro de toda a atividade no microcontrolador. Esta pode ser dividida quanto à sua funcionalidade em dois tipos: **RAM** (*Random Acess Memory*) e **ROM** (*Read Only Memory*), a principal diferença entre as duas está principalmente no facto de a **RAM** permitir leitura e escrita dos dados e a **ROM** apenas permitir a leitura dos mesmos. Estas podem ainda ser decompostas em vários tipos, **DRAM** (*Dinamic RAM)*, **SRAM** (*Static RAM)*, **PROM** *(Programmable ROM)*, **EPROM** *(Eraseable PROM)* [18], entre outras.

De forma semelhante à arquitetura de um computador, o **BUS** de um microcontrolador é o componente responsável pela transmissão de dados entre componentes. Qualquer componente presente no dispositivo irá comunicar através deste sistema, este pode ser caracterizado essencialmente como um barramento de comunicação de dados, quanto mais *bits* permitir transmitir melhor será o seu desempenho.

Para concretizar a transmissão este utiliza um grupo de condutores, o seu número é variável e depende diretamente da quantidade de informação que irá ser transferida pelo mesmo. Assim, ter-se-á dois **BUS**, um de dados que conterá a informação a transmitir e outro correspondente ao endereço que identifica o seu correspondente.

Assim, a memória pode ser acedida utilizando o endereçamento existente, isto é, para se aceder a um determinado conteúdo é utilizado o endereço onde está presente a informação, desta forma consegue-se controlo sob a informação que se está a ler e assim que a informação seja apresentada corretamente. Após saber o endereço pode-se então determinar se se pretende apenas ler o conteúdo ou também escrever no endereço selecionado.

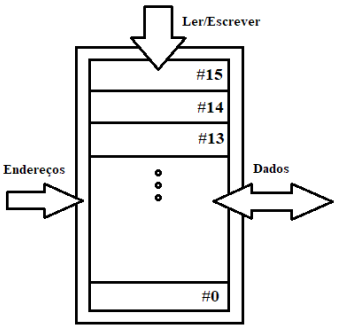


Figura 34 - Modelo unidade de memória

Com isto consegue-se aceder, manipular e armazenar a informação necessária para o processo, sendo este um aspeto muito importante no desenvolvimento de qualquer solução, desta forma a memória desempenha um papel essencial na estrutura do microcontrolador.

2.5. MATLAB

O MATLAB é um software de cálculo numérico de alta performance, que tem sido uma das ferramentas mais populares para análise e solução de problemas complexos em engenharia, matemática e ciência.

Este teve origem na década de 1970, quando o seu criador, *Cleve Moler*, era um estudante de doutoramento na Universidade do Novo México [20].

Durante esse período trabalhava num projeto de pesquisa que envolvia análise numérica e modelos matemáticos. Devido às dificuldades que enfrentava na realização dessas tarefas utilizando as ferramentas disponíveis na época começou a desenvolver uma linguagem de programação de alto nível que seria mais fácil e intuitiva para realizar as tarefas que enfrentava.

Em 1978, *Moler* e seus colegas lançaram a primeira versão do MATLAB, que se chamava inicialmente MATRIX. Esta versão era uma ferramenta de processamento de matrizes baseada em Fortran que foi rapidamente melhorada para incluir outras funcionalidades, como programação de gráficos e processamento de sinais.

Com o tempo, a popularidade do MATLAB cresceu rapidamente, e em 1984 *Moler* fundou a *MathWorks*, uma empresa especializada em software dedicado para aplicações que envolvam cálculo numérico. A empresa continuou a desenvolver o MATLAB, expandindo as suas funcionalidades e melhorando o seu desempenho.

Nos últimos anos, o MATLAB tem continuado a evoluir e melhorar, com a introdução de novas ferramentas e funcionalidades para análise e solução de problemas. Além disso, a sua compatibilidade com outras linguagens de programação, como o Python e o C++, tornou-o uma escolha popular para quem trabalha em múltiplas plataformas e procura aplicações versáteis que envolvam tecnologias diferentes. Este é conhecido pela sua rápida introdução e adaptação bem como ampla a sua biblioteca de ferramentas para resolução de problemas.

Outra das suas vantagens é a possibilidade de criar gráficos e visualizações. Podem ser criados gráficos de alta qualidade que facilitam a compreensão e interação com aplicações bem como tornam mais fácil visualizar e analisar dados complexos.

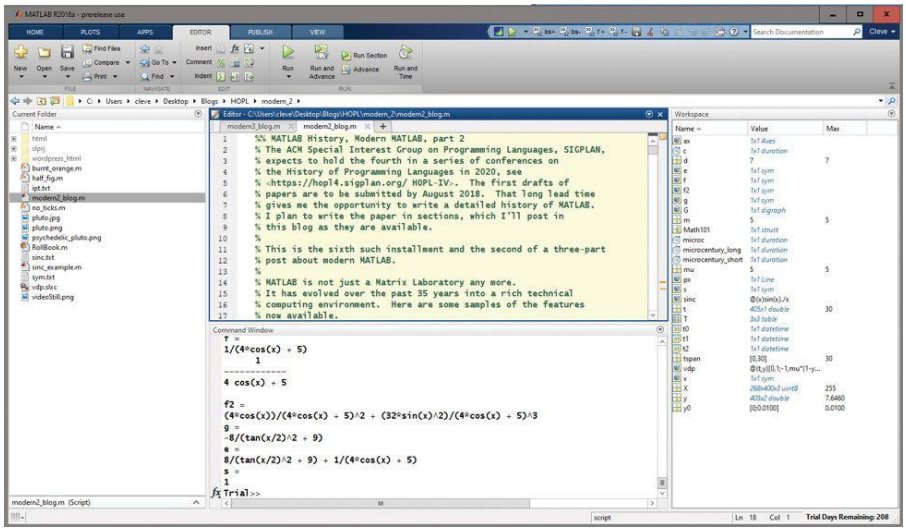


Figura 35 - Ambiente de desenvolvimento MATLAB [20]

Possui ainda uma vasta biblioteca de funções e ferramentas. Esta inclui funções para matrizes, cálculo numérico, processamento de sinais, otimização, estatística e muito mais. Além disso, o MATLAB é compatível com outras bibliotecas populares, como o NumPy e o Python.

Resumindo, o MATLAB é um software poderoso e facilmente aplicável em soluções de cálculo numérico e com interoperabilidade. Com a sua extensa biblioteca de funções e versatilidade permite criar aplicações complexas que envolvam diversas disciplinas dispondo de recursos que tornam a sua implementação uma mais-valia extremamente importante.

3. DESENVOLVIMENTO DA SOLUÇÃO

3.1. Introdução

Após a introdução feita nos capítulos anteriores em assuntos cruciais ao desenvolvimento do projeto da presente dissertação de mestrado, este capítulo aborda os temas envolvidos no desenvolvimento e execução do trabalho, bem como a relação entre os mesmos e qual o papel que desempenham no processo. Os diversos temas serão introduzidos com base no seu envolvimento na cadeia de ação, começando pela aquisição e processamento de imagem e terminando no movimento do robô.

Para a análise do ambiente de jogo e interação com o utilizador é importante a correta identificação da jogada efetuada. Desta forma para além da aquisição da imagem, o seu processamento desempenha um papel central no desenvolvimento da solução. Serão demonstrados os métodos utilizados para tratamento desta informação.

Em seguida com recurso a um algoritmo de decisão será calculada a melhor resposta para a jogada observada. Com esta informação e com recurso a cinemática irão ser calculados os parâmetros de cada junta para colocar o manipulador na posição adequada de forma a colocar a próxima peça na casa pertencente à resposta calculada.

Para enviar os parâmetros calculados anteriormente para os motores de cada junta foi utilizado um controlador *Arduíno*, este permite controlar cada junta de forma independente com recurso a uma placa *Braccio Shield V1*. Este serve ainda de ponte de comunicação entre o software desenvolvido e o sistema apresentado.

Por último será apresentado o software desenvolvido, este é o elemento agregador dos pontos introduzidos anteriormente e permite a interação com o utilizador. Como elemento principal está a janela de interação com o utilizador onde será mostrado o estado atual do tabuleiro bem como das juntas do manipulador. Existem ainda duas janelas secundárias, uma que permite a configuração de alguns parâmetros do ambiente de jogo, bem como a dimensão do tabuleiro, o seu posicionamento e as portas utilizadas para os equipamentos associados, nomeadamente o *Arduino* e a câmara, e outra janela que permite o controlo das juntas do manipulador de forma independente em modo de teste.

O software utilizado será o *MATLAB*, este dispõe de uma biblioteca de ferramentas muito extensa e adaptada a diversas áreas do conhecimento, sendo que para o trabalho apresentado serão integradas algumas funções de processamento de imagem, de comunicação com o microcontrolador *Arduino* e ainda de cinemática.

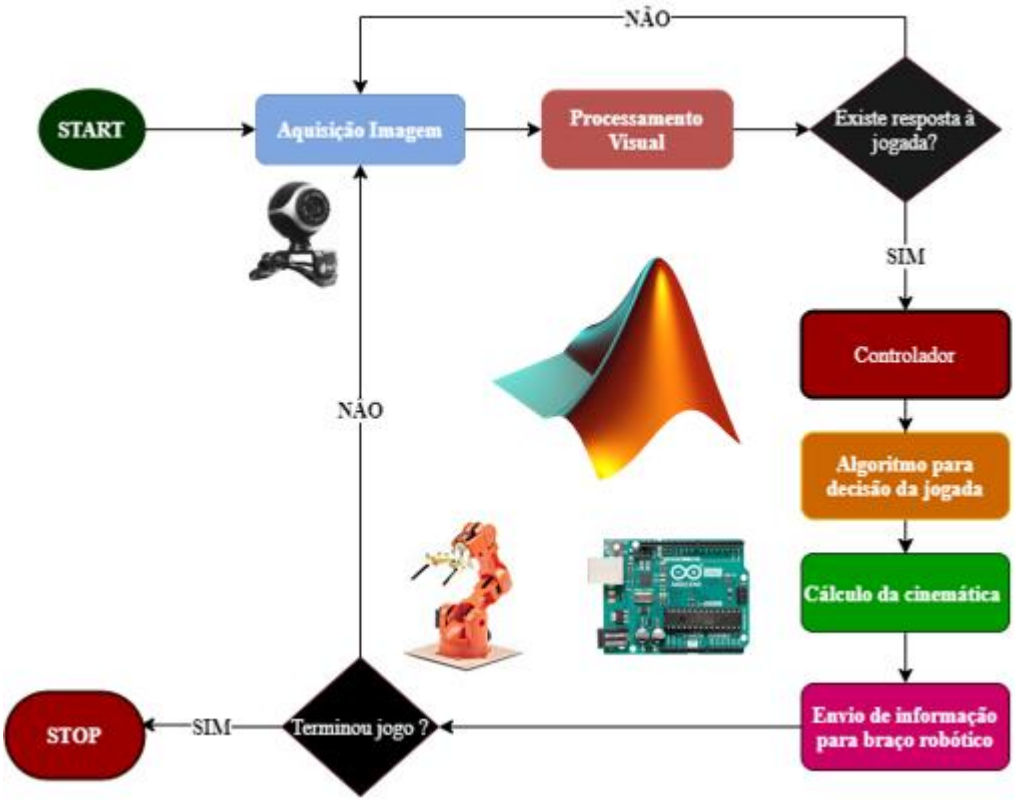


Figura 36 - Fluxograma do sistema a implementar no projeto

Para o controlo da solução pode-se representar o processo pelo fluxograma presente na Figura 36. Dando início ao processo irá ser adquirido o estado do tabuleiro e extraída a informação útil para o processo. Em seguida com base na informação proveniente irá ser gerida a decisão a executar pelo sistema de controlo e calculada a resposta. Após isto, com base na resposta calculada irão ser determinados os parâmetros das juntas de forma que o robô execute a jogada. Com os parâmetros calculados serão então enviados para o robô através do microcontrolador; o robô colocará assim a peça na casa correspondente à jogada determinada. Findo este processo será verificado o estado do tabuleiro e as condições de vitória para o jogo, caso não exista ainda um vencedor ou um empate, o jogo continuará até que se verifique uma destas condições.

3.2. Câmara

Para a aquisição de imagem foi utilizada uma câmara comum da marca *NGS* modelo *XPRESSCAM 300 (*Figura 37*)*, esta possui um sensor de captura com resolução de 8 Mpx. A ligação da mesma é feita por USB ao computador onde estará a ser executado o software de controlo.

Figura 37- Câmara usada para aquisição - NGS XPRESSCAM 300

3.3. Robô

O robô escolhido para o desenvolvimento do projeto foi o *Arduino Braccio*, os elementos decisivos para a sua escolha foram a disponibilidade, portabilidade e comprovada funcionalidade, sendo composto por peças leves e de baixo custo permite ainda uma fácil adaptação num ambiente de desenvolvimento.

Figura 38- Robô Arduino Braccio

Este possui uma composição bastante comum e utilizada em ambiente industrial. É composto por 6 servomotores, juntas de plástico e uma base de sustentação que permite, devido à sua marcação observar a rotação da base em graus relativamente à posição inicial.
Pode-se decompor o mesmo quanto à sua anatomia em:

1. **Base**, possibilita a rotação do mesmo segundo o eixo vertical,
2. **Ombro**, em conjunto com as juntas que sucedem os movimentos horizontais e verticais do robô,
3. **Antebraço**, partilha o eixo da junta anterior,
4. **Punho** – **Horizontal**, permite controlar o ângulo de ataque da garra,
5. **Punho – Rotativo**, permite rodar o punho,
6. **Garra**, responsável pela captura dos objetos.

É ainda composto por uma placa *Braccio Shield V1*, que permite o controlo dos servomotores através do microcontrolador utilizado, protegendo o mesmo e alimentado os servomotores de forma isolada. Um microcontrolador *Arduino Uno*, permite controlar as juntas do robô através do mesmo. Para este projeto irá ser usado um método de controlo híbrido onde o microcontrolador será responsável pela receção dos parâmetros das juntas e o controlador externo (computador com *Software* desenvolvido) irá realizar o controlo do processo. Para a comunicação é utilizada uma porta USB e para controlo das juntas do robô é utilizada uma mensagem enviada com a seguinte composição: *"Início, J1, J2, J3, J4, J5, J6"*, onde no campo início deverá estar um valor booleano que serve de controlo, se for verdadeiro permite a verificação dos valores das juntas, de J1 a J6 que após serem verificados se estiverem de acordo com os limites definidos são enviados para o robô. Na Figura 39 apresenta-se um excerto do programa utilizado no microcontrolador onde é possível verificar como são processadas as mensagens recebidas pelo mesmo.

```
PLC_CMD = Serial.readString();  // Recebe e armazena a mensagem na variável PLC_CMD
Serial.print("Mensagem recebida = "+PLC_CMD+"\n");
String CMD1= getValue(PLC_CMD,',',0);
start = CMD1.toInt();

String CMD2 = getValue(PLC_CMD,',',1);
M1 = CMD2.toInt();

String CMD3 = getValue(PLC_CMD,',',2);
M2 = CMD3.toInt();

String CMD4 = getValue(PLC_CMD,',',3);
M3 = CMD4.toInt();

String CMD5 = getValue(PLC_CMD,',',4);
M4 = CMD5.toInt();

String CMD6 = getValue(PLC_CMD,',',5);
M5 = CMD6.toInt();

String CMD7 = getValue(PLC_CMD,',',6);
M6 = CMD7.toInt();


  if ( M1>=0  && M1<=180 &&  M2>=0  && M2<=180 &&  M3>=0  && M3<=180 &&  M4>=0  && M4<=180 &&  M5>=0  && M5<=180 &&  M6>=10  && M6<=100)
  {
      Serial.println("Mensagem recebida corretamente :) \n");
         if(start==1){
            BraccioRobot.moveToPosition(pos.set(M1, M2, M3, M4, M5, M6),30);
            Serial.println("Posicionado\n");
            Serial.flush();
                    }
  }
```

Figura 39 – Programa para leitura da mensagem pelo microcontrolador

3.4. Sistema de visão

Para determinar o estado do tabuleiro de jogo será necessário adquirir uma imagem do mesmo, para este efeito será usado um sensor ótico.

No entanto para além disso é importante definir o que se vai detetar e qual o ambiente em que o elemento a detetar se insere. Assim sendo, são tarefas principais para o ambiente de visão:

1. Identificar área do tabuleiro de jogo,
2. Diferenciar as peças em jogo com base no seu tipo, "X ou O",
3. Verificar e validar com base no posicionamento as peças no tabuleiro.

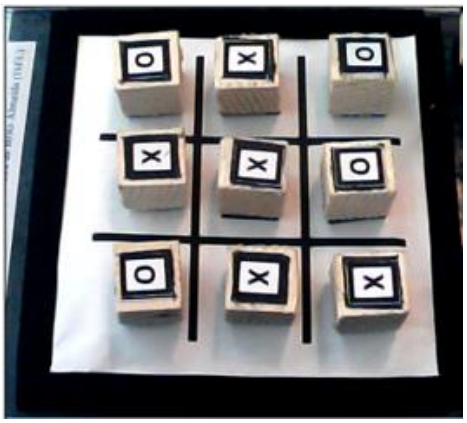


Figura 40 – Tabuleiro com peças colocadas – Imagem Câmara NGS

Existem ainda algumas dificuldades que podem surgir através da interação com agentes externos, sendo no processamento de imagem o principal fator a iluminação; para tentar minimizar os efeitos deste fator, como é possível observar na Figura 40, a área de jogo encontra-se demarcada com uma moldura preta, esta foi utilizada de forma a facilitar a sua identificação através do gradiente formado com o fundo branco do tabuleiro.

Após os testes feitos, verificou-se que a intensidade da luz incidente pode ser baixa, devido à facilidade que existe em detetar os contornos pelo contraste existente, no entanto, a reflexão da mesma pode ser prejudicial uma vez que pode incidir nos contornos da área de jogo e com base na sua dimensão pode invalidar a identificação da área de jogo, uma vez criar assim áreas de sombra. (Figura 41)

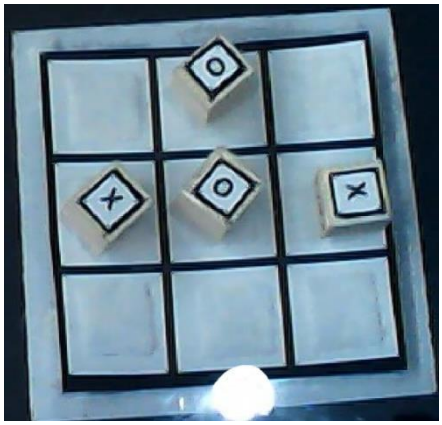


Figura 41 - Exemplo de incidência de luz excessiva

Para além do tabuleiro de jogo existe ainda a necessidade de identificação das peças, como é possível observar na Figura 40, a estratégia para identificar as mesmas passou pelo mesmo princípio descrito anteriormente, existindo um contraste facilmente identificável consegue-se isolar as peças presentes na imagem. Aqui, no entanto, o maior desafio passou pela diferenciação da marca presente nas peças, para isto o método passou por isolar as peças através de segmentação da imagem e em seguida verificar se é possível encontrar contornos que permitam a construção de um círculo na imagem resultante, em caso deste existir sabe-se que se trata de uma peça "*O*", caso contrário "*X*".

Desta forma, existindo um contraste que facilita a deteção dos contornos que serão utilizados para o processamento, os filtros utilizados para o processamento da imagem adquirida permitiram exponenciar e tomar partido desta característica.

Para isto foi criada uma função (*getBoard*, Figura 42) que permite obter o estado do tabuleiro e ainda com base numa matriz de casas preenchidas anteriormente retornar qual a nova jogada bem como validar a mesma com base na sua colocação, esta será descrita em seguida, sendo composta pelas operações necessárias para concluir as tarefas dos sistemas de visão.

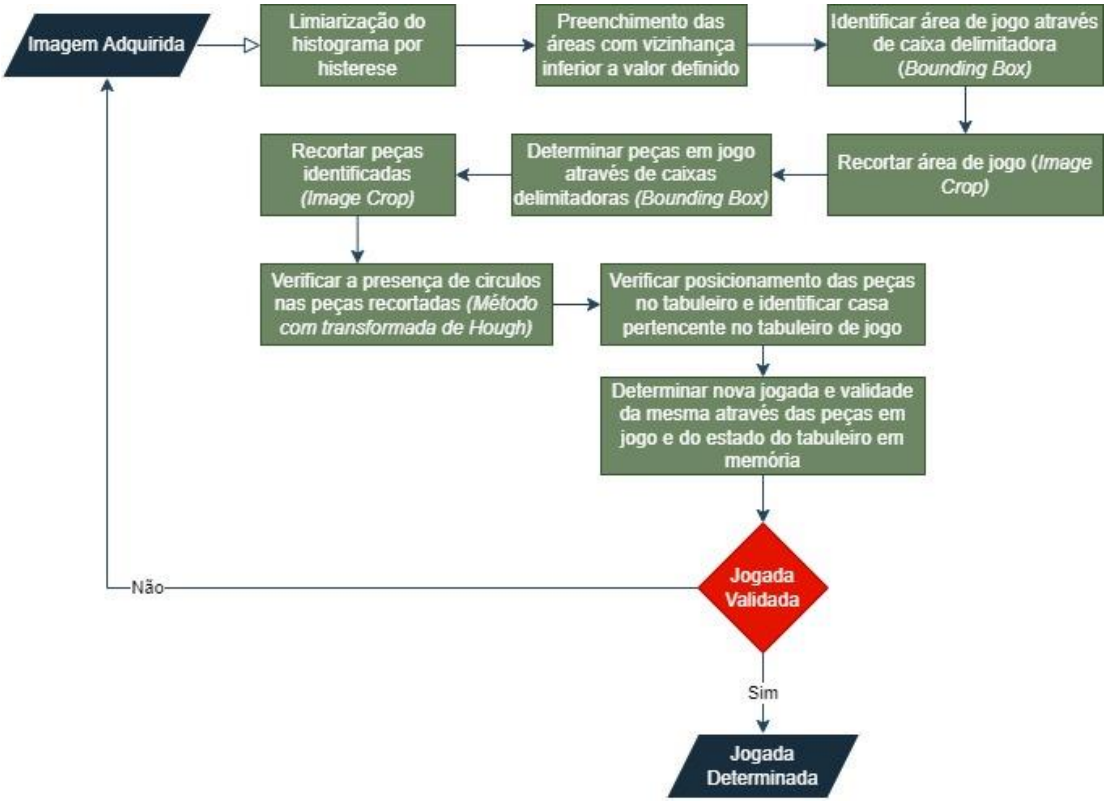


Figura 42 - Fluxograma da função getBoard

Identificar área do tabuleiro de jogo:

O processo de identificação da área de jogo começa com a aquisição do tabuleiro, Figura 43.

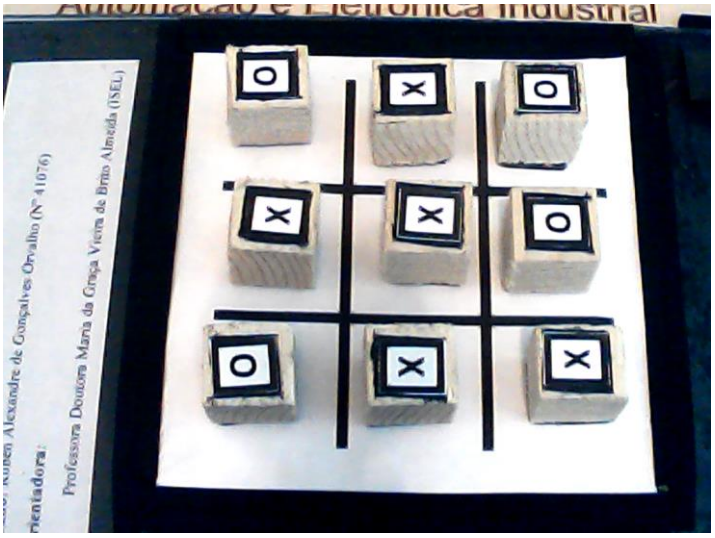


Figura 43 – Exemplo de imagem adquirida como entrada para sistema de visão

Como etapa inicial do processo de processamento é aplicada a técnica de limiarização do histograma por histerese, esta operação é realizada tomando partido do contraste que existe nos elementos de forma a remover possível ruído na imagem introduzido por fatores externos. Os valores de cor presentes na imagem são comparados conforme uma referência e se estiverem colocados abaixo do limite são substituídos por *0* caso contrário o valor será *1*, desta forma o produto desta operação é uma imagem a preto e branco onde são facilmente identificáveis os contornos presentes, Figura 44.

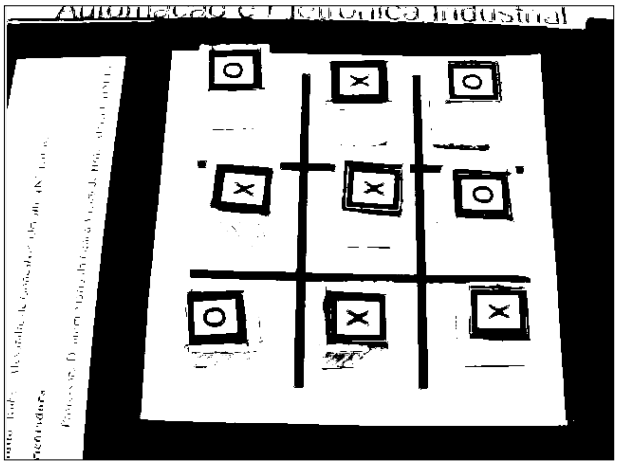


Figura 44 - Limiarização do histograma por limites

Usando como base a imagem resultante são preenchidas as áreas com vizinhança inferior a um valor definido, este encontra-se entre a dimensão do tabuleiro e de cada uma das casas de jogo, o objetivo é isolar a área do tabuleiro das restantes para identificar a mesma mais facilmente, Figura 45.

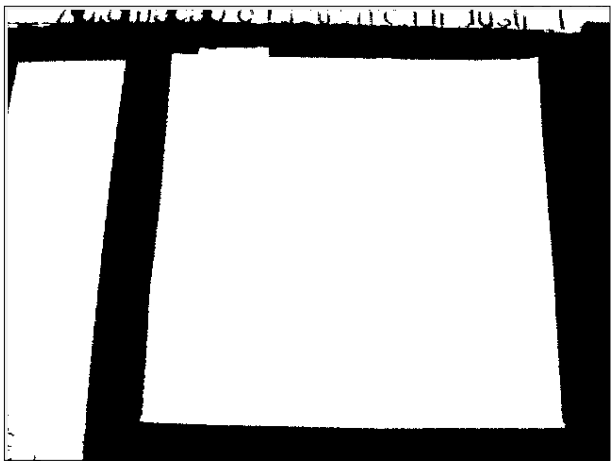


Figura 45 - Imagem com vizinhanças preenchidas

No seguimento é então utilizado um método bastante comum em sistemas de visão, o método de identificação de caixas delimitadoras ou *"Bounding Box"*, este utiliza os contornos identificados para colocar uma caixa delimitada por valores em coordenadas x e y que irá conter o objeto ou objetos contidos na imagem, Figura 46, com esta informação pode-se determinar pela área das regiões determinadas qual corresponde ao tabuleiro de jogo.

Figura 46- Caixa delimitadora aplicada em imagem com vizinhanças preenchidas

Após identificar a área de jogo é feito um recorte com base nos limites identificados anteriormente, resulta assim o tabuleiro sem qualquer elemento externo presente.

Identificar peças em jogo:

De forma semelhante, a técnica aplicada anteriormente é utilizada novamente para agora determinar a presença de peças no tabuleiro, aqui é aplicado um filtro com base na área das caixas delimitadoras encontradas e são utilizadas apenas as que se encontram dentro dos valores esperados para as peças. Ou seja, a área de uma peça é inferior à esperada para o tabuleiro conseguindo-se assim identificar e isolar cada peça, Figura 47.

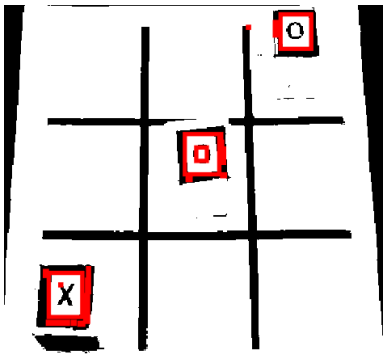


Figura 47- Tabuleiro com método Bounding Box a identificar peças em jogo

Com as peças isoladas é aplicada uma função para verificar a existência de um círculo na imagem, em caso afirmativo está-se perante uma peça "*O"* caso contrário será *"X"*. Esta utiliza a transformada de *Hough*, uma técnica que como explicado em 2.2.1, calcula a função de um conjunto acumulado de pontos previamente selecionados como contornos da imagem, e com esta informação permite determinar se se está perante uma linha ou um círculo, Figura 48.

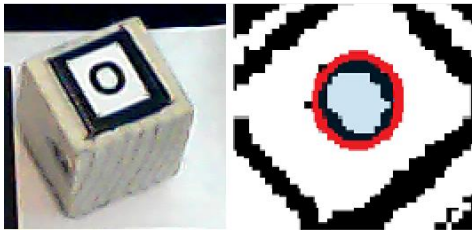
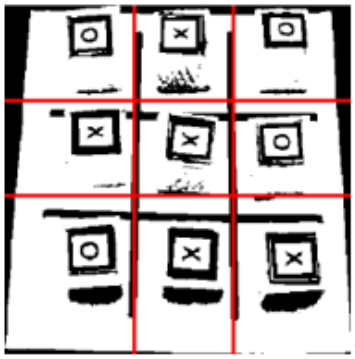


Figura 48 - Imagem de uma peça onde numa é detetado um círculo

Tendo as peças identificadas será apenas necessário verificar em que casa do tabuleiro foram colocadas. Para isto a imagem foi dividida pelo número de casas e as coordenadas das peças identificadas foram verificadas de acordo com esta divisão, Figura 49.



0 Posição 1	**1** Posição 4	**0** Posição 7
1 Posição 2	**1** Posição 5	**0** Posição 8
0 Posição 3	**1** Posição 6	**1** Posição 9

Figura 49- a) Tabuleiro com divisão de casas definida b) Matriz que guarda valor das posições com X

Compensação do posicionamento da câmara:

Devido ao posicionamento da câmara é necessário compensar o ângulo sob o qual está a ser adquirida a imagem com base na altura das peças relativamente ao tabuleiro. Considerando um ângulo de 30º para as posições mais afastadas da câmara e que as peças têm cerca de 20mm de altura tem-se:

$$20 * \tan(30) \approx 11.57\ mm$$

Uma vez que o tabuleiro tem 100mm de comprimento, sabe-se que tem que se projetar a divisão com cerca de 8% de tolerância (Figura 49 e Figura 50).

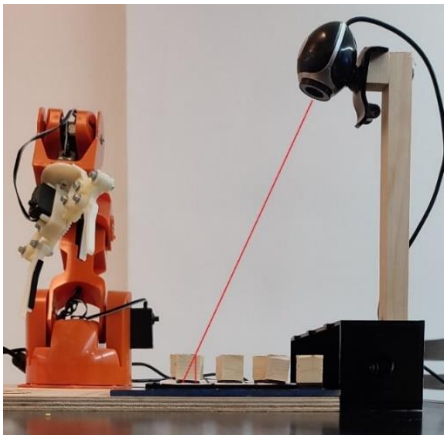


Figura 50 – Vista do ângulo de aquisição da câmara

Tomada de decisão:

Findo o processo, com base nas casas preenchidas anteriormente é verificada qual a casa que não se encontrava ainda em jogo e a validade da mesma, para validar a jogada será verificada se é uma casa que ainda não se encontrava preenchida e se a nova peça está identificada com a marca correta para a jogada a adquirir, *X* ou *O*. Caso a mesma seja validada esta é retornada bem como atualizadas as casas preenchidas, Figura 51. Nesta figura pode-se observar uma imagem com duas peças do mesmo utilizador em jogo, resultando assim numa jogada inválida. Verificar-se-á também uma jogada inválida se existirem várias peças identificadas como peças novas em jogo, Figura 52.

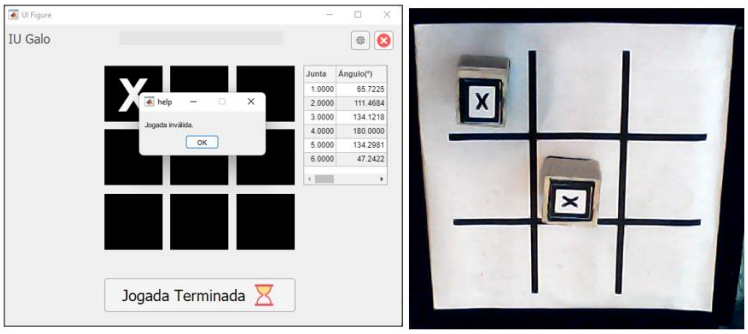


Figura 51 - Exemplo de jogada inválida com peça errada colocada

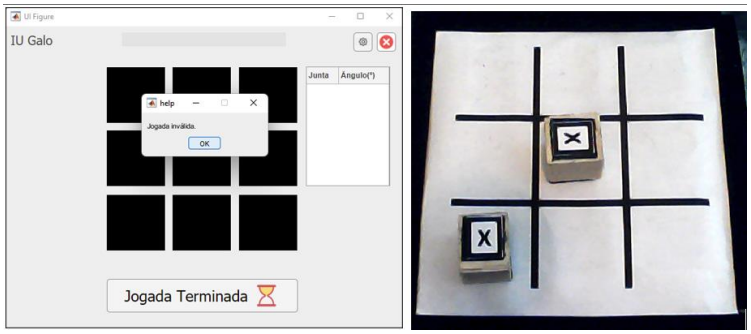


Figura 52 - Exemplo de jogada inválida com várias peças novas colocadas

3.5. Cinemática

A cinemática é essencial para o funcionamento eficiente do robô utilizado, esta permite calcular a posição e orientação do robô em relação ao seu sistema de coordenadas, o que é crucial para realizar as tarefas necessárias. Além disso, o cálculo da cinemática permite entender como a movimentação dos elementos do robô afeta a posição final do mesmo, o que para além de permitir prever o seu posicionamento é importante para evitar colisões e garantir a segurança do mesmo e do utilizador. Por fim, a cinemática será fundamental para garantir a precisão da posição do punho do robô, o que é essencial para a realização das tarefas designadas.

Para a realização deste trabalho foi ainda integrada uma técnica que permite descrever a estrutura de um robô de forma a integrar os seus parâmetros de juntas com o seu modelo 3D, designado por modelo *URDF* [21]. Este permite a integração dos mesmos em vários ambientes de simulação, possibilitando assim a análise dos movimentos do robô no âmbito da cinemática bem como da segurança prevendo colisões. Para aplicar este modelo será necessário inicialmente determinar a tabela de parâmetros *D-H*.

Assim, como notado anteriormente ir-se-á aplicar o método *D-H*, para isto inicialmente serão estabelecidos os referenciais das juntas do mesmo (Figura 53).

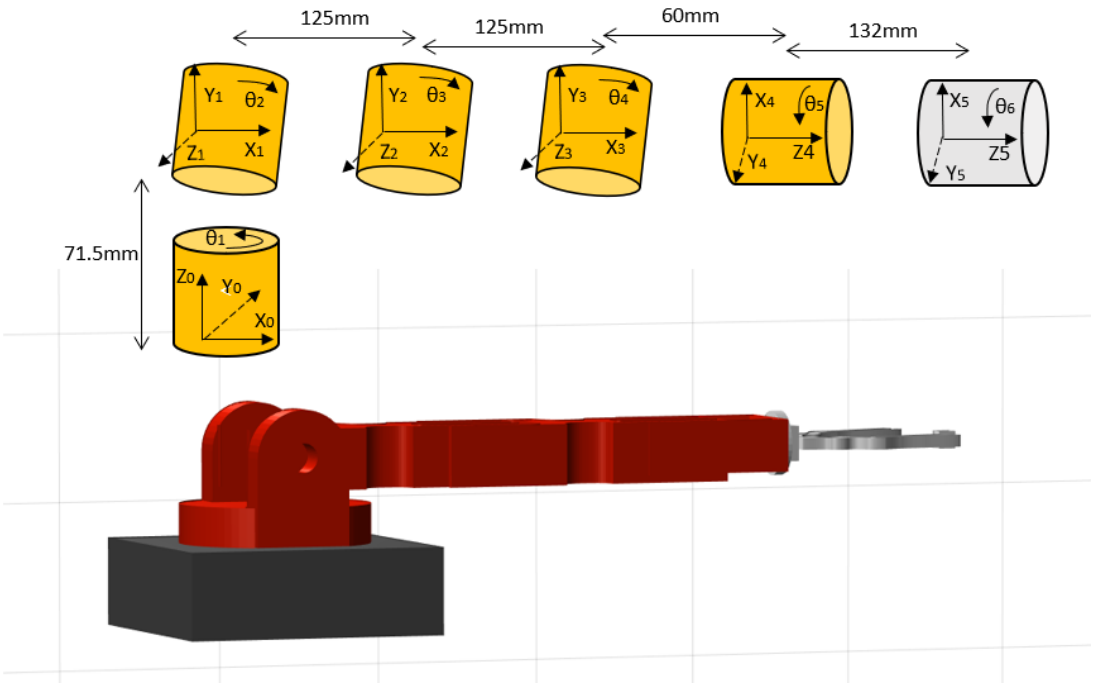


Figura 53- Referenciais aplicados no robô Arduino Braccio

Em seguida pode-se determinar a tabela de parâmetros *D-H*:

Tabela 2 - Tabela parâmetros D-H - Arduino Braccio

Junta i	$\boldsymbol{\theta_i}$ (º)	$\boldsymbol{\alpha_i}$ (º)	$\boldsymbol{a_i}$**(mm)**	$\boldsymbol{d_i}$**(mm)**
J_1	θ_1	$-\pi/2$	0	71.5
J_2	θ_2	0	125	0
J_3	θ_3	0	125	0
J_4	θ_4	$\pi/2$	60	0
J_5	θ_5	0	0	132

Após determinar a tabela *D-H* pode-se começar a construir o ficheiro *URDF*, para isto irão ser utilizados como parâmetros:

- *Robot name* – nome para o robô,
    - *Link name* – nome do elo,
        - *Visual* – estrutura que contém os elementos visuais da junta especificada,
            - *Geometry* – Ficheiro dos elementos geométricos,
                - *Mesh* – Ficheiro com o modelo 3D para a junta associada,
            - *Material name* – nome do material aplicado,
    - *Joint name* – nome da junta,
    - *Joint type* – tipo de junta,
        - *Axis* – eixo de rotação da junta,
        - *Limit* – especifica os limites mecânicos do robô, tais como, velocidade, esforço e rotação ou posição,
        - Origin rpy – vetor de rotação entre a junta i_1 a i_{n-1},
        - Origin xyz – vetor de translação entre a junta i_1 a i_{n-1},
        - Parent link – junta que antecede a atual,
        - Child link – junta que sucede a atual,

Para cada uma das juntas irão ser preenchidos os valores especificados anteriormente. Na Figura 54 pode-se ver um exemplo dos valores utilizados para a junta 1.

```
<?xml version="1.0"?>
<robot name="braccio">

  <link name="J1">
    <visual>
      <geometry>
        <mesh filename="ficheiro_3D_Base.stl"/>
      </geometry>
      <material name="orange"/>
    </visual>
  </link>

  <joint name="Base" type="revolute">
    <axis xyz="0 0 -1"/>
    <limit effort="1000.0" lower="0.0" upper="3.141592653589793" velocity="4.0"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <parent link="base_link"/>
    <child link="link1"/>
  </joint>

</robot>
```

Figura 54 - Exemplo de parametrização URDF para a primeira junta

Com a parametrização do modelo *URDF* concluída o mesmo poderá ser importado para um software de simulação apropriado e utilizado para prever e simular trajetórias bem como estudar a sua implementação no ambiente previsto.

Para concluir a recolha de elementos para o cálculo da cinemática é então necessário identificar a localização para colocação das peças, para isto foi utilizada a base do robô como origem do referencial e determinada a distribuição com base num tabuleiro com 9 casas, os restantes elementos são introduzidos como variáveis sendo estes: as dimensões segundo *X* e *Y* do tabuleiro (*tabX* e *tabY*), a altura do mesmo em relação à base do robô e a sua distância segundo *X* e *Y* relativamente à origem (*distRX* e *distRY*), Figura 55.

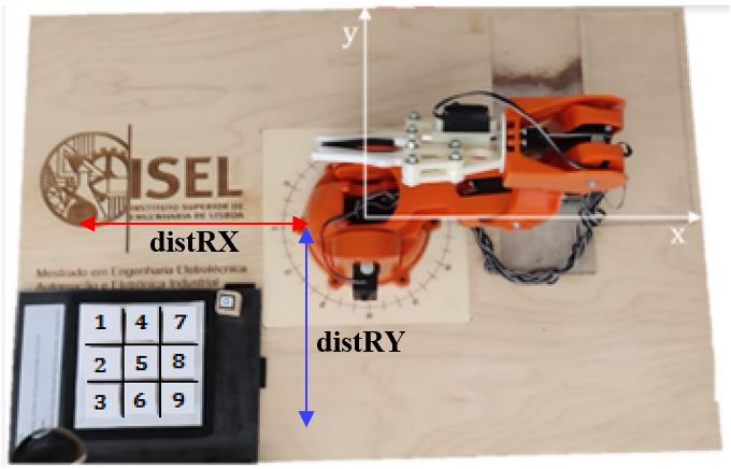


Figura 55 – Ambiente de trabalho com eixos cartesianos e numeração do tabuleiro

Na Figura 56, apresenta-se a função utilizada para cálculo da relação de distâncias entre células e das mesmas para o robô, os valores resultantes irão corresponder às coordenadas para colocação da peça segundo *X* e *Y*, onde *distRX* e *distRY* representam respetivamente as distâncias segundo *X* e *Y* dos limites do tabuleiro, e *tabX* e *tabY* correspondem ao tamanho do tabuleiro segundo *X* e *Y*, respetivamente.

```
for i=1:3
    for j=1:3
        cel=cel+1;

        tabCoord(cel,1)=distRX+(tabX/6)+(tabX/3*(i-1));
        tabCoord(cel,2)=distRY+(tabY/6)+(tabY/3*(3-j));
    end
end
```

Figura 56 - Função para cálculo das coordenadas em X e Y de colocação das peças

Para a simulação foi utilizado o software MATLAB, e criada uma função denominada *robotKinBraccio* (Figura 57), esta permite utilizar o modelo *URDF* construído anteriormente e com as coordenadas finais do punho determinar a cinemática inversa do robô.

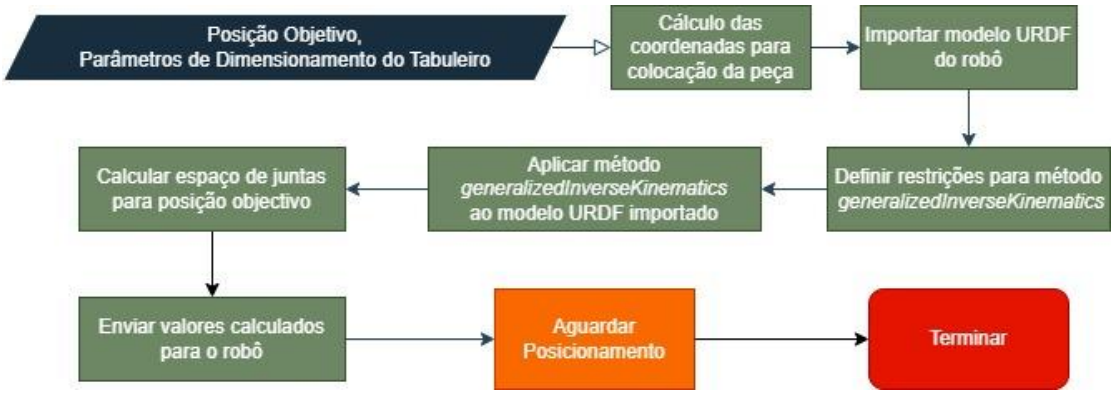


Figura 57- Fluxograma da função robotKinBraccio

O método aplicado, utiliza um objeto denominado por *generalizedInverseKinematics* [22]. Este utiliza um conjunto de funções para, de forma não linear, calcular com base nas restrições aplicadas a melhor solução para a trajetória desejada. Recebe como entradas: a descrição do robô, a posição e orientação desejadas no espaço de trabalho e um conjunto de restrições que podem ser aplicadas às posições das juntas do robô. Como restrições pode-se usar, o ângulo de ataque do punho, os limites cartesianos do ambiente de simulação, a distância ao objeto, a pose do punho relativamente ao objeto, entre outras. Para o cálculo dos parâmetros de juntas que permitirão o robô atingir a posição definida é usado por defeito o algoritmo *Broyden–Fletcher–Goldfarb–Shanno (BFGS)* este é um dos métodos de otimização mais

populares em problemas de otimização multivariável por ser eficiente e geralmente convergente. É um algoritmo de otimização utilizado para encontrar o mínimo de uma função de várias variáveis, este atualiza a matriz *Hessiana* aproximada a cada iteração usando técnicas de previsão e utiliza uma busca linear na direção determinada para encontrar a próxima solução [23].

Neste processo foram utilizadas as seguintes restrições:

- Limites cartesianos (*constraintCartesianBounds*), este tem como função garantir que o robô não possui nenhum elemento colocado para lá do limite inferior vertical do mesmo, prevenindo assim colisões com o tabuleiro,
- Distância ao objeto (*constraintPositionTarget*), utilizada para garantir que o punho se situa na posição final com uma tolerância mínima, permitindo assim uma maior repetibilidade no processo,

Assim, importando o modelo *URDF* e aplicado o método apresentado consegue-se determinar a posição das juntas para as posições desejadas do tabuleiro.

Utilizando como parâmetros para calcular as coordenadas de colocação de peça na posição 1 do tabuleiro (Figura 59), com:

- Distância em relação ao tabuleiro:
  - Segundo Y =-0.175m
  - Segundo X =-0.220m
- Tamanho do tabuleiro:
  - Segundo Y =-0.100m
  - Segundo X =0.100m
- Altura em relação à base do braço = 0.100m

(estes parâmetros podem ser determinados usando a orientação presente na Figura 55, como demonstrado na Figura 58)

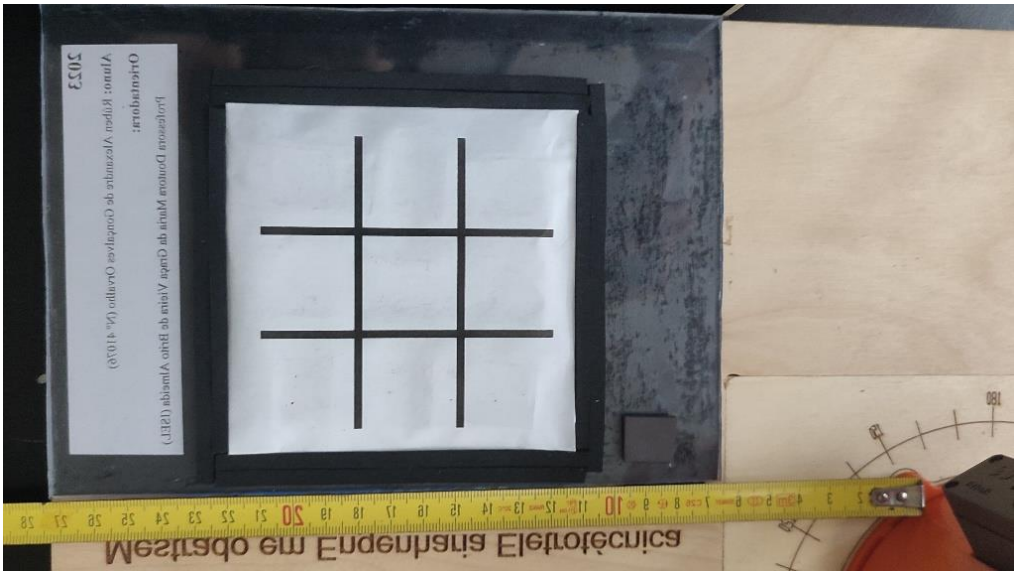


Figura 58 - Exemplo de medição da distância segundo X

Retiram-se os seguintes valores para as coordenadas da peça:

- Segundo X: -0.203m
- Segundo Y: -0.092m

Aplicando o método *generalizedInverseKinematics*, recebem-se como parâmetros para as juntas:

1. **Base**, 65.71º
2. **Ombro**, 111.48º
3. **Antebraço**, 134.09º
4. **Punho** – **Horizontal**, 180.00º
5. **Punho – Rotativo**, 134.51º
6. **Garra**, 16.40º

Como se pode verificar pela Figura 59 e pela Figura 60, consegue-se uma reprodução muito aproximada da posição real do braço após o envio dos parâmetros, mesmo tratando-se de um robô bastante limitado e com algumas fragilidades. Com a utilização do modelo URDF é possível simular as diversas trajetórias do robô numa aplicação real.

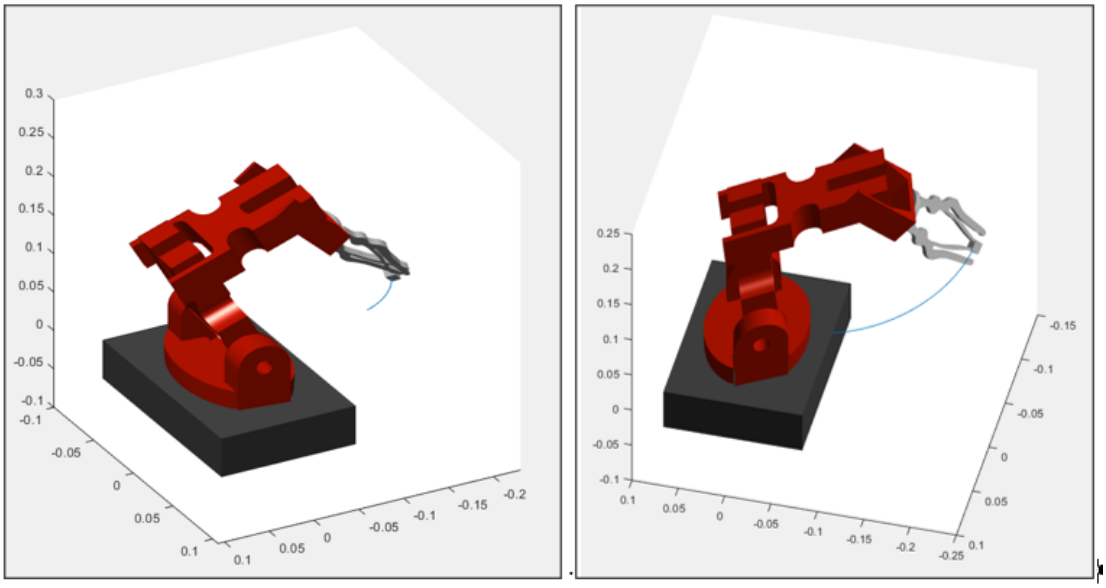


Figura 59 - Simulação com modelo URDF para posição 1 do tabuleiro

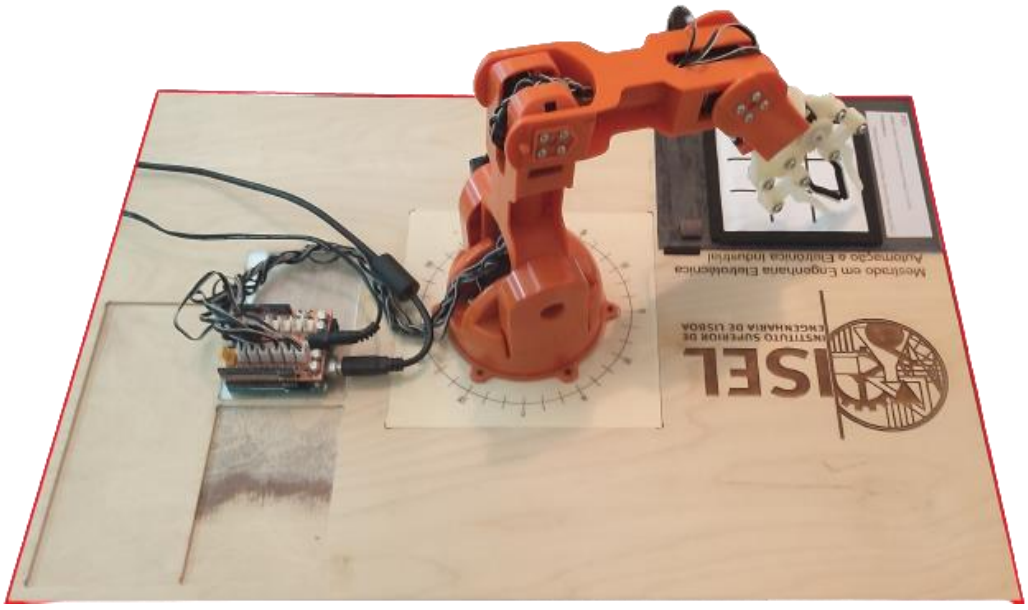


Figura 60 – Posicionamento do robô para parâmetros calculados em simulação para posição 1

3.6. Ambiente de interação

Como responsável pela interação com o utilizador e com os elementos do projeto foi criado um ambiente que irá fazer a gestão de todo o processo. Para isto foi utilizado o MATLAB, mantendo assim a plataforma utilizada anteriormente. Para além disto, será aqui feita a gestão das funções necessárias para o correto funcionamento de todo o processo.

Para iniciar o processo será necessário efetuar algumas configurações, as portas de comunicação do elemento de aquisição de imagem e do robô e os parâmetros de dimensionamento do tabuleiro. Assim, no arranque do processo será recebida uma mensagem indicando que o *hardware* e o tabuleiro deverão ser configurados (Figura 61).

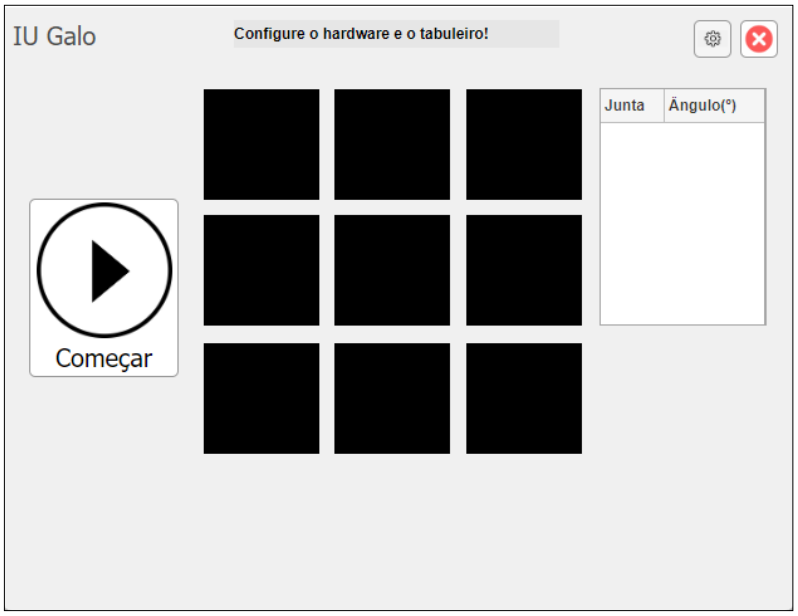


Figura 61 - Interface Utilizador - Menu principal - Início

Para este efeito será automaticamente aberto o menu de configuração (Figura 62). Os valores introduzidos neste menu serão utilizados pela função apresentada na Figura 56 e determinados com base nas indicações da Figura 55.

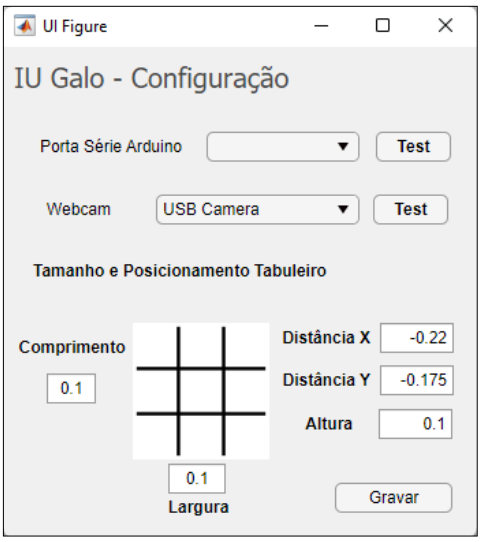


Figura 62 - Interface Utilizador - Menu Configuração

As configurações de tamanho e posicionamento apresentadas ao ser aberta a janela, derivam de um ficheiro previamente guardado em formato *csv*, este permite configurar o ambiente no ficheiro de forma que o utilizador não necessite de introduzir estes dados sempre que iniciar o *software*.

Após a configuração é possível iniciar o jogo premindo o botão "*Começar"*, o utilizador irá ser questionado sobre as peças que irá usar e se pretende começar a jogar em primeiro lugar (Figura 63).

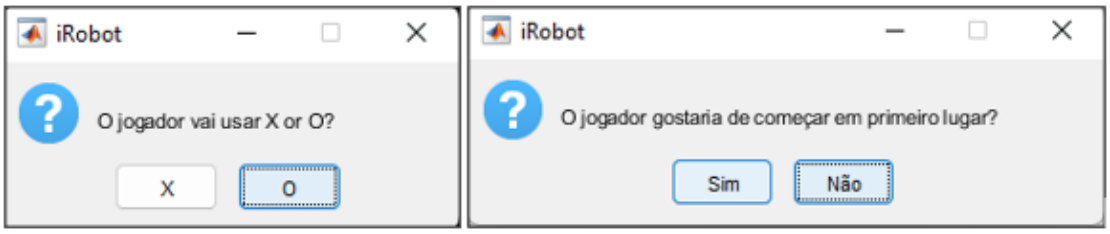


Figura 63 - Interface Utilizador - Escolhas Iniciais

Assim, será então iniciado o processo de jogo, caso o jogador decida começar em primeiro lugar terá de indicar quando terminou a sua jogada para ser feita a aquisição e atualização do estado do tabuleiro de jogo, caso contrário a primeira jogada será feita pelo robô.

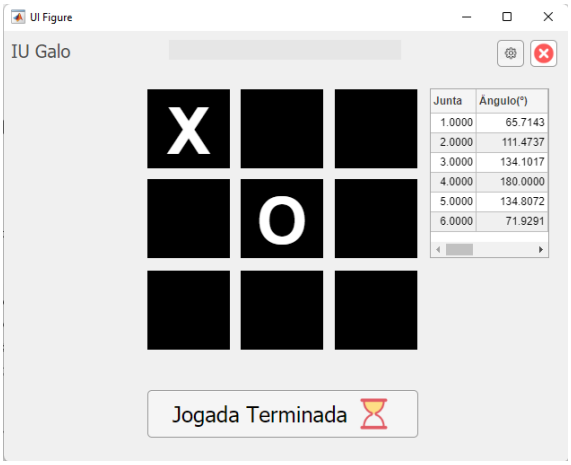


Figura 64 – Interface Utilizador – Jogo a decorrer

Para a decisão da jogada a ser protagonizada pelo robô foi utilizado um algoritmo presente na biblioteca colaborativa online "*Rosetta Code*" [24] sendo este adaptado ao ambiente criado na interface apresentada, Figura 65; este verifica após cada jogada algumas condições para determinar a resposta.

Como evidenciado no fluxograma apresentado este começa por verificar quais as casas vazias e se existem condições para fazer com que a próxima jogada seja vencedora, esta verificação é feita para ambos os jogadores e caso alguma possibilidade de vitória seja verificada a jogada é feita nessa posição, dando origem a uma vitória ou um bloqueio caso se verifique que o adversário tem condições para a vitória. Após isto são verificadas algumas estratégias de jogo comuns como: a possibilidade de colocar duas peças em linha de forma a preparar a vitória na próxima jogada, a possibilidade de colocar uma peça na casa central, num dos cantos ou numa das laterais, caso nenhuma se verifique é feita uma jogada aleatória. Ao terminar o jogo, será indicado qual o vencedor ou em caso de empate que ninguém ganhou.

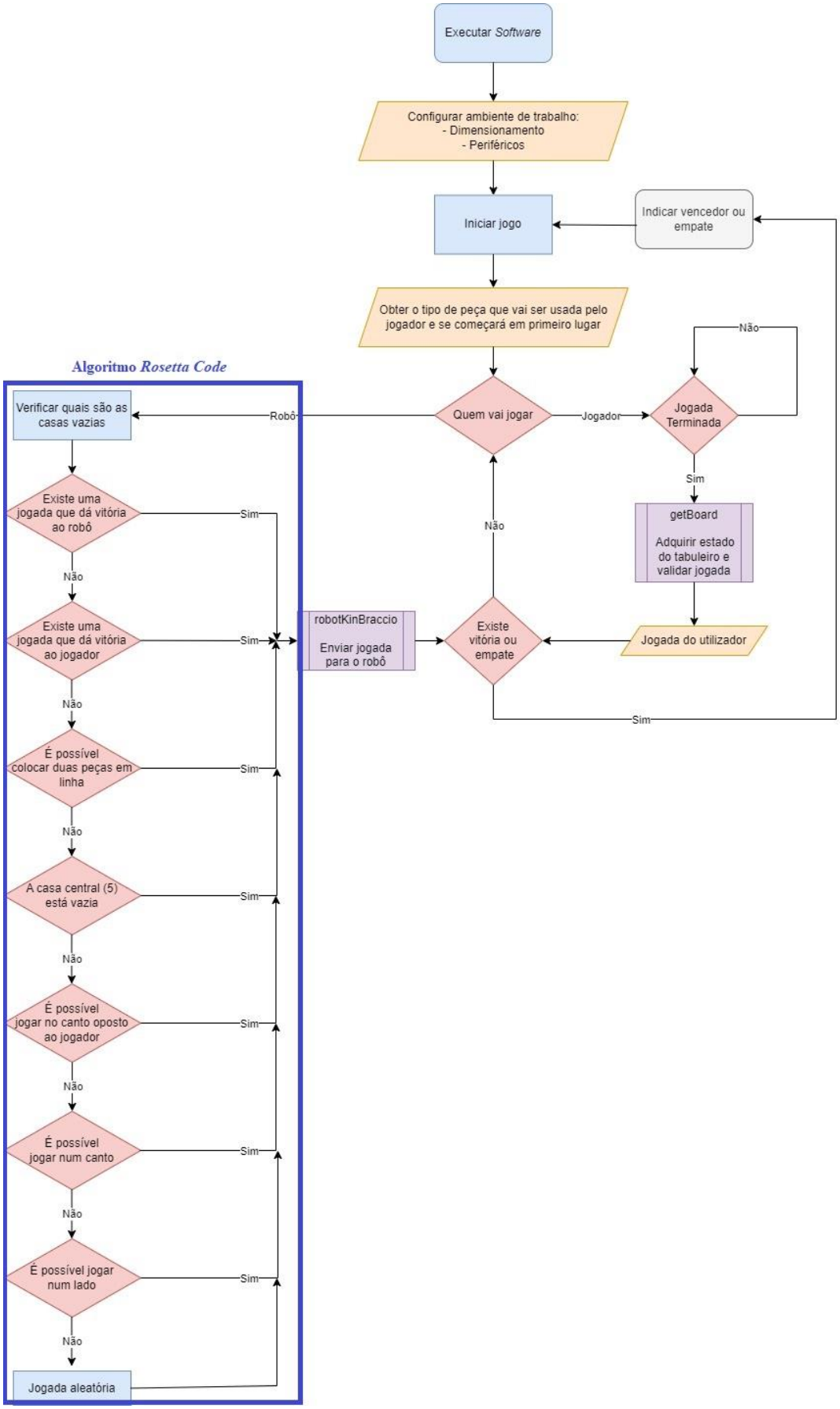


Figura 65 - Fluxograma do ambiente de interação com algoritmo Rosetta Code

4. RESULTADOS

Ao parametrizar corretamente a função para aquisição de imagem foi possível verificar o potencial da aplicação e foi possível obter resultados bastante positivos, apesar de as condições de iluminação tornarem ainda assim o sistema em certas situações bastante vulnerável. Tentando contornar estas vulnerabilidades foram utilizados elementos com elevado contraste, essencialmente preto e branco, desta forma mesmo com fraca luminosidade conseguem-se facilmente identificar as áreas de interesse, Figura 66.

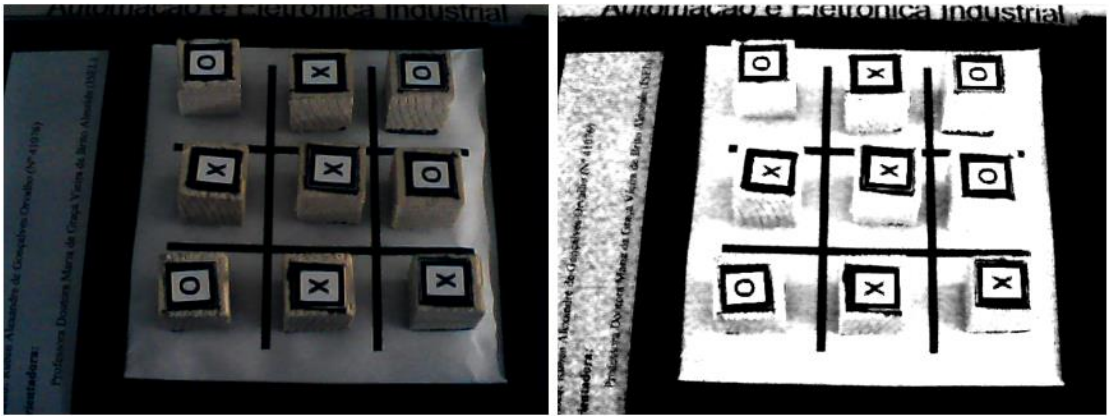


Figura 66 - Exemplo de vantagem de contraste em iluminação insuficiente

Foi também interessante verificar a flexibilidade que se obtém ao integrar neste tipo de soluções um microcontrolador, este permite fazer a ponte entre o *software* e o *hardware* de uma forma bastante eficaz e fiável, tornou possível enviar os valores para as juntas correspondentes às posições calculadas pela cinemática e verificar a sua correspondência na movimentação do robô.

Igualmente foi bastante proveitosa a aplicação do modelo *URDF* de forma a obter um resultado em simulação mais aproximado do modelo real, Figura 67. Pode-se assim simular os movimentos do robô antes de ir para o modelo físico, evitando deste modo possíveis danos.

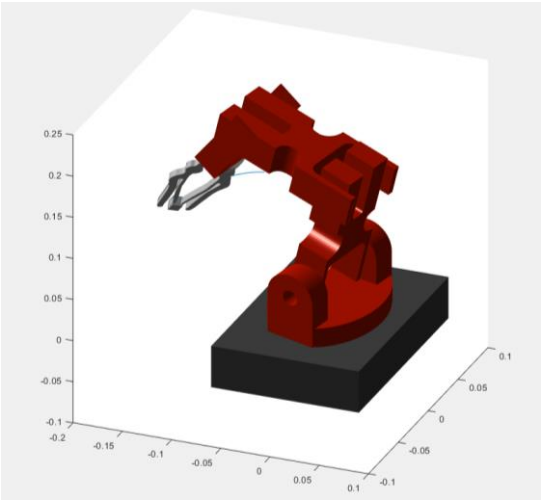


Figura 67 - Modelo URDF em simulação

Relativamente à cinemática, após importar o modelo do robô foi possível verificar os seus movimentos com maior detalhe, no entanto existem algumas limitações no seu movimento, as peças tiveram que ser bastante leves e o tabuleiro reduzido e colocado numa posição próxima do mesmo. Isto deve-se ao facto de a área de trabalho do manipulador ser reduzida.

Relativamente ao desempenho do manipulador, devido ao seu tamanho e fragilidades a posição inicial para colocação do tabuleiro tornou-se problemática para algumas posições. Uma vez não ser possível manter o ângulo de ataque do punho o mais vertical possível, sendo o ideal 90º, em algumas colocações de peças verificaram-se colisões com peças colocadas em casas adjacentes, Figura 68. A vantagem de usar um ângulo de ataque próximo dos 90º é que este evita a colisão entre o punho e as peças ao fazer a abertura para largar a peça.

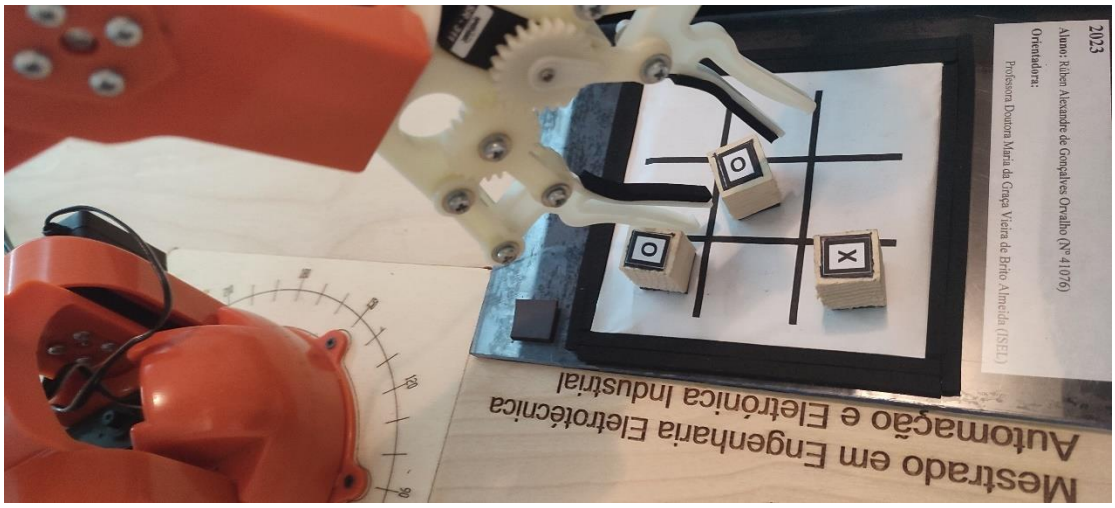


Figura 68 - Colisão ao colocar peça

Para contornar este problema e tomando partido da flexibilidade de configuração da posição do tabuleiro, Figura 69, foi testada uma posição de colocação do tabuleiro intermédia, mais próxima do manipulador e numa posição mais central, Figura 70.

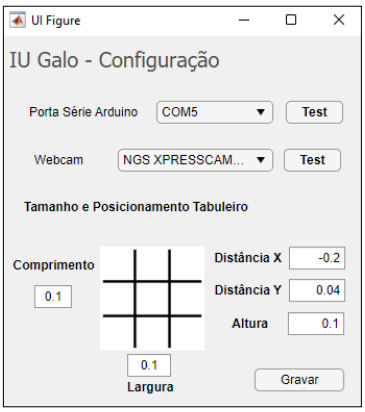


Figura 69 - Configuração para posição central do tabuleiro

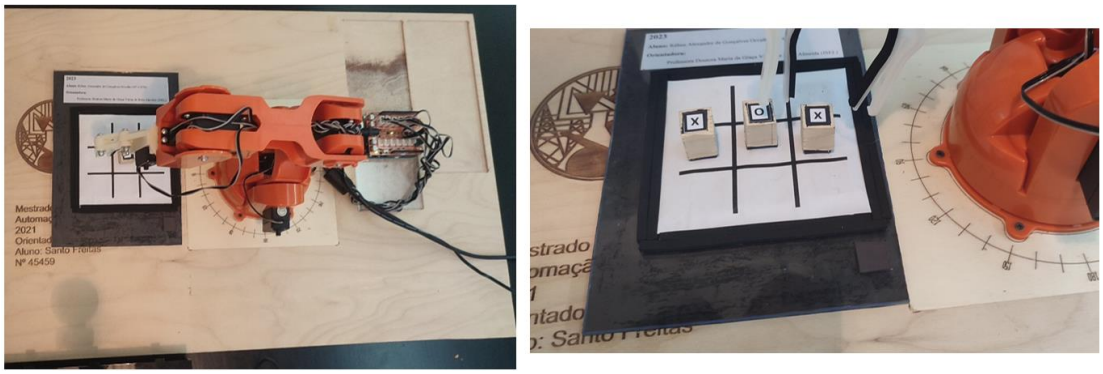


Figura 70 - Nova posição para colocação do tabuleiro

Esta posição previne as situações de colisão presenciadas anteriormente uma vez permitir manter o ângulo de ataque do punho perto dos 90º, possibilita ainda fazer a abertura da garra e colocação da peça numa posição com altura superior à da peça. Pode-se assim delimitar as áreas de trabalho mais adequadas para o manipulador com base na sua eficiência, Figura 71, estando marcada a vermelho a área menos aconselhada para a colocação do tabuleiro, a amarelo a área desaconselhada, mas que permite a operação com algumas limitações e a verde a área mais vantajosa para o robô.

Figura 71 – Visão geral do ambiente de trabalho

Por fim conseguiu-se integrar o sistema de decisão e de interação com o utilizador com bastante sucesso e verificou-se que todos os sistemas interagem de forma bastante eficaz, Figura 72 e Figura 73.

Figura 72 - Fim de jogo com vitória

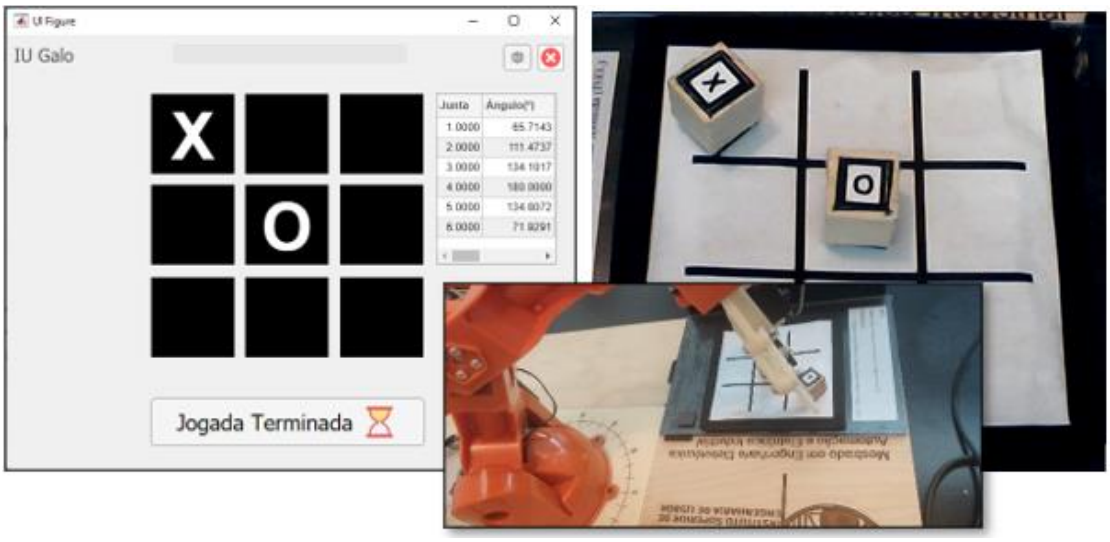


Figura 73 – Resposta do robô após jogada inicial por parte do utilizador

5. CONCLUSÕES

5.1. Conclusões gerais

Após dimensionar e integrar as diferentes vertentes no projeto proposto foi bastante interessante verificar o potencial do relacionamento entre o processamento de imagem e a robótica.

Ao analisar o processamento de imagem verificou-se que apesar de bastante flexível e com um potencial já comprovado em diversas aplicações, é uma tecnologia bastante sensível às condições luminosas do ambiente em que se encontra inserida, sendo que esta condição terá sido a mais impactante no processo. Ainda assim, a câmara escolhida excedeu as expectativas, em parte devido à capacidade introduzida pelo processamento da aquisição e pelo facto de se ter escolhido uma solução com elementos a preto e branco maioritariamente, o que pelo seu elevado contraste facilita a aplicações dos métodos escolhidos.

Relativamente ao robô, apesar do ambiente de simulação ser bastante útil para determinar a sua posição e verificar sua a reprodução, o modelo físico disponível tem algumas limitações a nível de construção, sendo bastante frágil e com pouca capacidade de carga, estas não impactaram de forma decisiva o projeto realizado, mas limitam bastante a sua aplicabilidade.

Abordando o microcontrolador utilizado, este teve um desempenho irrepreensível durante o processo, permitindo o controlo das juntas do robô.

O *software* utilizado, MATLAB, tornou-se bastante capaz e dotado de imensas potencialidades, permitiu integrar vertentes de tecnologia diferentes no mesmo ambiente bem como construir uma interface para interação.

Quanto à interação das tecnologias com o utilizador foi interessante verificar como pela interação entre diversas tecnologias se consegue responder de forma autónoma a uma ação.

Em suma, pode-se considerar que a solução proposta foi desempenhada com sucesso, o que comprova a metodologia adotada.

5.2. Trabalho futuro

Após os resultados obtidos neste trabalho considero que este seja uma demonstração que se pode utilizar equipamentos de baixo custo e globalmente acessíveis para desenvolver soluções interessantes e com expressão na demonstração da interação entre tecnologias bastante presentes na indústria e sociedade atual.

Todavia a utilização de um robô com maior capacidade de carga permitiria a introdução em tarefas mais exigentes e onde se possa tirar melhor partido do sistema de aquisição e processamento de imagem. Entrariam em jogo, no entanto outras preocupações como a segurança do utilizador onde a previsão de trajetórias e estratégias de controlo teriam que ser implementadas.

Como trabalho futuro poder-se-ia utilizar um robô com melhor desempenho e construção mais robusta de forma a integrar o mesmo em ambientes mais exigentes e tirar partido de outras funcionalidades das tecnologias utilizadas.

Por fim, como melhoramento do processo poderiam ser melhorados os tempos de resposta do manipulador, integrando o seu controlo também na plataforma MATLAB. Em termos de processamento de imagem, poderia ainda ser implementada uma rotina para verificação automática da posição do tabuleiro bem como uma estratégia para facilitar a identificação das peças.

BIBLIOGRAFIA

[1] H. M. d. Costa, "Ensaios Mecânicos," 2019 September 2019. [Online]. Available: https://www.researchgate.net/figure/Figura-15-A-principio-e-de-forma-bem-simplificada-o-olho-humano-pode-ser-considerado_fig2_335639923.

[2] B. Jähne, Digital Image Processing 5th revised and extended edition, 2005.

[3] D. A. Forsyth and J. Ponce, Computer Vision: A Modern Approach, 2011.

[4] Mathworks, "Image Referencing Toolbox," March 2023. [Online]. Available: https://www.mathworks.com/help/pdf_doc/images/images_ref.pdf.

[5] J. Canny, A Computational Approach to Edge Detection, 1986.

[6] C. Harris and M. Stephens, A Combined Corner and Edge Detector, 1988.

[7] INFAIMON, "Controlo de qualidade em peças especulares no setor automóve," INFAIMON, 2022. [Online]. Available: https://infaimon.com/pt-pt/automotivo/controlo-qualidade-automovel/.

[8] M. Ceccarelli, Fundamentals of Mechanics of Robotic Manipulation, 2004.

[9] Ege Üniversitesi, "Robot Teknolojisi," 2015.

[10] B. Siciliano, L. Sciavicco, L. Villani and G. Oriolo, Robotics, Modelling, Planning and Control, 2009.

[11] F. O. Nunes and A. G., Sistemas Robóticos, ISEL, 2008.

[12] FANUC, "Robôs Delta," [Online]. Available: https://www.fanuc.eu/pt/pt/rob%c3%b4s/p%c3%a1gina-filtro-rob%c3%b4s/delta-robots/dr-3ib-series/dr3ib-8l.

[13] Meccanismo Complesso, "3D Rotations and Euler Angles," [Online]. Available: https://www.meccanismocomplesso.org/en/3d-rotations-and-euler-angles-in-python.

[14] R. Noyce and M. Hoff, A History of Microprocessor Development at Intel, 1981.

[15] J. G. Casas, Elogio de la pereza, la ciencia de la computación en una perspectiva histórica, 2003.

[16] T. Wilmshurst, Designing Embedded Systems with PIC Microncontrollers, 2010.

[17] Arduino, "Arduino UNO," Arduino, [Online]. Available: https://store.arduino.cc/products/arduino-uno-rev3.

[18] Khan Academy, "Computer and the Internet: Computer Components," [Online]. Available: https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:computers#xcae6f4a7ff015e7d:computer-components.

[19] D. Russel, The Principles of Computer Network, 1989.

[20] C. Moler, The Origins of Matlab, 2004.

[21] L. Joseph, Mastering ROS for Robotics Programming, 2015.

[22] Matlab, "generalizedInverseKinematics," [Online]. Available: https://www.mathworks.com/help/solutions/robotics.

[23] C. T. Kelley, Iterative Methods for Optimization, 1999.

[24] Rosetta Code, "Tic Tac Toe Code," [Online]. Available: https://rosettacode.org/wiki/Tic-tac-toe.

[25] R. Szabo and R.-S. Ricman, "Building a Tic-tac-toe Playing Robotic Arm," in *2022 30th Telecommunications Forum (TELFOR)*, 2022.

[26] K. Radlak and F. Marcin, "Integration of robotic arm manipulator with computer vision in a project-based learning environment," in *IEEE Frontiers in Education Conference, FIE*, Texas, 2015.

[27] S. A. Nugroho, A. S. Prihatmanto and A. S. Rohman, "Design and implementation of kinematics model and trajectory planning for NAO humanoid robot in a tic-tac-toe board game," in *IEEE 4th International Conference on System Engineering and Technology (ICSET)*, Bandung, 2014.

ANEXO A – Programa Arduino

```
#include <BraccioRobot.h>
#include <Servo.h>
#include <String.h>

String PLC_CMD; //Variável para receber a mensagem
int   start=0,M1=90,M2=45,M3=180,M4=180,M5=90,M6=90;     //Variáveis inteiras para auxiliar no controlo do sistema
String M1r="",M2r="",M3r="",M4r="",M5r="",M6r="";
Position myInitialPosition(90, 45, 180, 180, 90,  80),pos,currpos; //Definir posição inicial
String getValue(String data, char separator, int index); // Protótipo da função que efetua split da mensagem


void setup() {

    BraccioRobot.init(myInitialPosition); // Função que manda o robô para uma posição de setup inicial -> Obrigatório para habilitar os 6 eixos do robô

    //init() vai inicializar o robô e enviar para a seguinte pos:
    //Base     (M1): 90 graus
    //Shoulder (M2): 90 graus
    //Elbow    (M3): 90 graus
    //Wrist    (M4): 90 graus
    //Wrist rot(M5): 90 graus
    //gripper  (M6): 72 graus

    pos.set(M1, M2, M3, M4, M5, M6);
    Serial.begin(9600); // Inicializa a comunicação série
}

void loop() {


    while(!Serial){
        Serial.print("Braccio ON: à espera de comunicação!");
```

```
            BraccioRobot.moveToPosition(pos, 10);  }

      while (Serial.available())
      {
            BraccioRobot.moveToPosition(pos, 10);

            M1r=String(BraccioRobot.currentPosition.getBase());
            M2r=String(BraccioRobot.currentPosition.getShoulder());
            M3r=String(BraccioRobot.currentPosition.getElbow());

M4r=String(BraccioRobot.currentPosition.getWristRotation());
            M5r=String(BraccioRobot.currentPosition.getWrist());
            M6r=String(BraccioRobot.currentPosition.getGripper());


            Serial.print(M1r+","+M2r+","+M3r+","+M4r+","+M5r+","+M6r+"
            \n");
           // Aguarda que seja recebido algum caracter via porta série
           //Formato que se está à espera de receber ://

           /*      start,M1,M2,M3,M4,M5,M6
            *      start = 0 : Indica parar o sistema
            *      start = 1 : Arrancar o sistema
            *      M1 a M6 = Valores de coordenada do robô
            *      M1: base, 0 a 180 graus
            *      M2: shoulder, 15 a 165 graus
            *      M3: elbow, , 0 a 180 graus
            *      M4: wrist, , 0 a 180 graus
            *      M5: wrist rotation, , 0 a 180 graus
            *     M6: gripper, 10 a 73 graus. 73 graus = gripper fechada
            */
           PLC_CMD  =  Serial.readString();    // Recebe  e  armazena  a
mensagem
           Serial.print("Mensagem recebida = "+PLC_CMD+"\n");
           String CMD1= getValue(PLC_CMD,',',0);
           start = CMD1.toInt();
           String CMD2 = getValue(PLC_CMD,',',1);
           M1 = CMD2.toInt();

           String CMD3 = getValue(PLC_CMD,',',2);
```

```
          M2 = CMD3.toInt();

          String CMD4 = getValue(PLC_CMD,',',3);
          M3 = CMD4.toInt();

          String CMD5 = getValue(PLC_CMD,',',4);
          M4 = CMD5.toInt();

          String CMD6 = getValue(PLC_CMD,',',5);
          M5 = CMD6.toInt();

          String CMD7 = getValue(PLC_CMD,',',6);
          M6 = CMD7.toInt();

            if ( M1>=0  && M1<=180 &&  M2>=0  && M2<=180 &&  M3>=0  &&
            M3<=180 &&  M4>=0  && M4<=180 &&  M5>=0  && M5<=180 &&
            M6>=10  && M6<=100)
            {
                Serial.println("Mensagem  recebida  corretamente  :)
\n");
                   if(start==1){
                      BraccioRobot.moveToPosition(pos.set(M1, M2, M3,
                     M4, M5, M6),60);
                      Serial.println("Posicionado\n");
                      Serial.flush();
                               }
            }
      }
 }

 String getValue(String data, char separator, int index)
 {
   int found = 0;
   int strIndex[] = {0, -1};
   int maxIndex = data.length()-1;

   for(int i=0; i<=maxIndex && found<=index; i++){
     if(data.charAt(i)==separator || i==maxIndex){
          found++;
          strIndex[0] = strIndex[1]+1;
```

```
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }

    return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

ANEXO B – Programa desenvolvido em MATLAB

Interface Utilizador – Janela Principal

```
classdef UI < matlab.apps.AppBase

    % Propriedades dos componentes da app
    properties (Access = public)
        UIFigure             matlab.ui.Figure
        JointTable           matlab.ui.control.Table
        CloseButton          matlab.ui.control.Button
        Field_1              matlab.ui.control.Label
        Field_4              matlab.ui.control.Label
        Field_2              matlab.ui.control.Label
        Field_5              matlab.ui.control.Label
        Field_7              matlab.ui.control.Label
        Field_8              matlab.ui.control.Label
        Field_3              matlab.ui.control.Label
        Field_6              matlab.ui.control.Label
        Field_9              matlab.ui.control.Label
        IUGaloLabel          matlab.ui.control.Label
        MoveFinishedButton   matlab.ui.control.Button
        STARTButton          matlab.ui.control.Button
        ConfigButton         matlab.ui.control.Button
        InfoBox              matlab.ui.control.Label
    end


    properties (Access = private)
        designTracker = ('         '); % Array com tracking do estado das casas
para app
        resp = "Não"; % Controlo da pChoice do utilizador
        turn = 1; % Controlo de turnos de jogo
        gameOver = 0; % Terminar jogo
        choose = 1; % Iniciar escolhas de prioridade e peças
        boards = false(3, 3, 2);     % Peças em jogo
        rep = ['   1 | 4 | 7' ; '   2 | 5 | 8' ; '   3 | 6 | 9']; % Representação
na cmd line
        sFilled = false(1,9); % Guarda casas preenchidas
        ic=0; % Variável para usar imagens de teste de forma incremental
        pHuman = [false ; true]; % Array que guarda prioridade de turno
        humanFirst = "Sim" % Guarda prioridade de jogo
        marks = 'OX'; % String que guarda peças escolhidas
        pChoice = 0 ; % Peça escolhida pelo jogador
        startup = 0; % Jogo a começar = 1
    end

    properties (Access = public)
        move = 0 % Última pChoice registada
        ard  % Arduino usado
        webcam % Webcam usada
        distRX % Distância da base do robô para o tabuleiro em X
        distRY % Distância da base do robô para o tabuleiro em Y
        tabX % Comprimento em X do tabuleiro
        tabY % Comprimento em Y do tabuleiro
        tabAlt % Altura do tabuleiro em relação à base ro robô
        configDone=false % Configurações definidas
```

```
        joints % Valores das juntas em graus para visualização
    end

    methods (Access = public)

        function Galo_PrincP(app)

            % Condições iniciais
            if (app.choose==1)
                clc
                app.choose=0;
                UpdBoard(app);
                app.boards = false(3, 3, 2);    % Peças em jogo
                  % Representação na cmd line
                app.rep = ['   1 | 4 | 7' ; '   2 | 5 | 8' ; '   3 | 6 | 9'];
                app.sFilled=false(1,9);
                app.ic=0;
                app.resp="Não";
                app.gameOver=false;

                % Escolhas iniciais

                app.pChoice=questdlg("O    jogador    vai    usar    X    or
O?","iRobot","X","O","O");

                app.humanFirst = questdlg("O jogador gostaria de começar em
primeiro lugar?","iRobot","Sim","Não","Não");

                    if app.humanFirst == "Não"
                app.pHuman = [false ; true];
                else
                app.pHuman = [true ; false];
                app.MoveFinishedButton.Enable = 1;
                app.MoveFinishedButton.Visible = 1;
                end


                if app.pChoice=="O" && app.humanFirst=="Sim"
                    app.marks = 'OX';
                elseif app.pChoice=="X" && app.humanFirst=="Sim"
                    app.marks = 'XO';
                elseif app.pChoice=="O" && app.humanFirst=="Não"
                    app.marks = 'XO';
                elseif app.pChoice=="X" && app.humanFirst=="Não"
                    app.marks = 'OX';
                end

                app.STARTButton.Enable = 0;
                app.STARTButton.Visible = 0;
                return
            end

            % Start game

            if (~app.gameOver)

                if app.pHuman(app.turn)
                    app.resp=0;
```

```
                    % Variável usada para incrementar número da imagem no path
em testes com imagens pre' recolhidas

                    %app.ic=app.ic+1;

                    [app.move,isValid,      isQuit,      app.sFilled]      =
GetMoveFromPlayer(app,app.webcam, app.sFilled, app.pChoice);
                    app.gameOver = isQuit;

                else
                    app.move = GetMoveFromComputer(app, app.turn, app.boards);
                    app.joints=robotKinBraccio(app.ard,app.move,0,
app.distRX, app.distRY, app.tabX, app.tabY, app.tabAlt);
                    FillJoints(app,app.joints);
                    isValid = true;
                    isQuit = false;
                end
                if isValid && ~isQuit
                    app.designTracker(app.move)= app.marks(app.turn);
% Atualizar jogada na variável de controlo para o ecrã
                    app.sFilled(1,app.move)=true;
% Atualizar posição jogada na variável de controlo
                    UpdBoard(app);
% Atualizar jogadas no ecrã de acordo com variável de controlo
                    [r, c] = ind2sub([3 3], app.move);
% Converter jogada em índices matriciais correspondentes
                    app.boards(r, c, app.turn) = true;
% Atualizar tabuleiro de acordo com indices matriciais da jogada
                    app.rep(r, 4*c) = app.marks(app.turn);
% Escrever jogada no array de controlo para cmd line
                    if CheckWin(app, app.boards(:, :, app.turn))
                        app.gameOver = true;
                        fprintf('\n')
                        if app.pHuman(app.turn)
                        msgbox('O jogador ganhou!','Game Over','help');
                        else
                        msgbox('O robot ganhou!','Game Over','help');
                        end
                    elseif CheckDraw(app,app.boards)
                        app.gameOver = true;
                        msgbox('Ninguém ganhou!','help');
                    elseif (app.ic>=5)
                        app.gameOver = true;
                        msgbox('Ninguém ganhou!','help');
                    end
                    app.turn = ~(app.turn-1)+1;

                    if (~app.pHuman(app.turn))
                        app.MoveFinishedButton.Enable = 0;
                        app.MoveFinishedButton.Visible = 0;
                        Galo_PrincP(app)
                    else
                        app.MoveFinishedButton.Enable = 1;
                        app.MoveFinishedButton.Visible = 1;
                    end
                elseif ~isValid
                app.MoveFinishedButton.Enable = 1;
                app.MoveFinishedButton.Visible = 1;
                end
            else
            end
        end
```

```
        function [move,isValid, isQuit, sFilled] = GetMoveFromPlayer(~,webcam,
sFilled, pChoice)
        % pChoices: 1-9, 0 é uma pChoice inválida
        % isValid - indica se a pChoice é válida
        % isQuit - indica se o jogador quer sair do jogo
        try
        [move, sFilled] = getBoard(webcam, pChoice, sFilled);
        catch
            msgbox('Não foi possível determinar a jogada, tente
novamente!','help');
            move=0;
        end
        % de 1v1
                if isempty(move)
                    msgbox('Jogaremos novamente em breve!','help');
                    move = 0;
                    isValid = true;
                    isQuit = true;
                else
                    isQuit = false;
                    if isnan(move) || move < 1 || move > 9
                        msgbox('Jogada inválida.','help');
                        isQuit = false;
                        isValid = false;
                    else
                        isValid = true;
                    end
                end
            end

        function move = GetMoveFromComputer(app,pNum, boards)

            if ~any(boards(:))      % Primeira pChoice no canto
                move = 1;
            else                     % Regras de "Newell & Simon's"
                pMe = boards(:, :, pNum);
                pThem = boards(:, :, ~(pNum-1)+1);
                possMoves = find(~(pMe | pThem)).';

                % Procurar pChoice vencedora
                move = FindWin(app, pMe, possMoves);
                if move
                    return
                end

                % Procurar bloquear o jogador
                move = FindWin(app, pThem, possMoves);
                if move
                    return
                end

                % Criar dupla (V)
                for m = possMoves
                    newPMe = pMe;
                    newPMe(m) = true;
                    if CheckDouble(app, newPMe, pThem)
                        move = m;
                        return
                    end
                end
```

```
                % Procurar colocar 2 em linha, sem deixar o outro fazer
                % dupla (V)
                notGoodMoves = false(size(possMoves));
                for m = possMoves
                    newPMe = pMe;
                    newPMe(m) = true;
                    if CheckPair(app,newPMe, pThem)
                        nextPossMoves = possMoves;
                        nextPossMoves(nextPossMoves == m) = [];
                        theirMove = FindWin(app, newPMe, nextPossMoves);
                        newPThem = pThem;
                        newPThem(theirMove) = true;
                        if ~app.CheckDouble(newPThem, newPMe)
                            move = m;
                            return
                        else
                            notGoodMoves(possMoves == m) = true;
                        end
                    end
                end
                possMoves(notGoodMoves) = [];

                % Jogar no centro se possível
                if any(possMoves == 5)
                    move = 5;
                    return
                end

                % Jogar no canto oposto do outro jogador
                corners = [1 3 7 9];
                move    =    intersect(possMoves,    corners(~(pMe(corners)    |
pThem(corners)) & pThem(fliplr(corners))));
                if ~isempty(move)
                    move = move(1);
                    return
                end

                % Jogar num canto vazio
                move = intersect(possMoves, corners);
                if move
                    move = move(1);
                    return
                end

                % Jogar num lado vazio
                sides = [2 4 6 8];
                move = intersect(possMoves, sides);
                if move
                    move = move(1);
                    return
                end

                % pChoice aleatória, nenhuma opção determinada anteriormente
                possMoves = find(~(pMe | pThem));
                move = possMoves(randi(length(possMoves)));
            end
        end

        function move = FindWin(app,board, possMoves)
        % board - Tabuleiro (Matriz 3x3)
```

```
            % move - pChoice
                for m = possMoves
                    newPMe = board;
                    newPMe(m) = true;
                    if CheckWin(app,newPMe)
                        move = m;
                        return
                    end
                end
                move = 0;
            end

            function win = CheckWin(~,board)
            % win - Verificar se existe uma condição de vitória - Colunas,
            % Linhas e Diagonais
            win = any(all(board,1)) || any(all(board, 2)) || all(diag(board)) ||
all(diag(fliplr(board)));
            end

            function fork = CheckDouble(~,p1, p2)
            % fork - Verificar se existe dupla (V)
                fork = sum([sum(p1)-sum(p2) (sum(p1, 2)-sum(p2, 2)).' ...
                    sum(diag(p1))-sum(diag(p2)) ...
                    sum(diag(fliplr(p1)))-sum(diag(fliplr(p2)))] == 2) > 1;
            end

            %%

            function pair = CheckPair(~,p1, p2)
            % pair - Verificar se existe um par (2 peças em linha desbloqueadas)
                pair = any([sum(p1)-sum(p2) (sum(p1, 2)-sum(p2, 2)).' ...
                    sum(diag(p1))-sum(diag(p2)) ...
                    sum(diag(fliplr(p1)))-sum(diag(fliplr(p2)))] == 2);
            end

            function draw = CheckDraw(~,boards)
            % boards - Ambos os tabuleiros (Matriz 3x3x2)
                draw = all(all(boards(:, :, 1) | boards(:, :, 2)));
            end


            function
updateCfg(app,ardInput,webcamInput,info,distRX,distRY,tabX,tabY,tabAlt)
                app.ard=ardInput;
                app.webcam=webcamInput;
                app.InfoBox.Text=info;
                app.distRX=distRX;
                app.distRY=distRY;
                app.tabX=tabX;
                app.tabY=tabY;
                app.tabAlt=tabAlt;
                app.configDone=true;
            end
        end

        methods (Access = private)

            function UpdBoard(app)
                for i=1:1:9
                    switch i
                        case 1
```

```
                        app.Field_1.Text = app.designTracker(1);
                    case 2
                        app.Field_2.Text = app.designTracker(2);
                    case 3
                        app.Field_3.Text = app.designTracker(3);
                    case 4
                        app.Field_4.Text = app.designTracker(4);
                    case 5
                        app.Field_5.Text = app.designTracker(5);
                    case 6
                        app.Field_6.Text = app.designTracker(6);
                    case 7
                        app.Field_7.Text = app.designTracker(7);
                    case 8
                        app.Field_8.Text = app.designTracker(8);
                    case 9
                        app.Field_9.Text = app.designTracker(9);
                end
            end
        end

        function CleanUp(app) %Função que limpa as varáveis de jogo

        app.designTracker = ('         '); % Array com tracking do estado das
casas para app
        app.turn=1; % Controlo de turnos de jogo
        app.gameOver = 0; % Terminar jogo
        app.choose = 1; % Iniciar escolhas de prioridade e peças
        app.boards = false(3, 3, 2);    % Peças em jogo
        app.rep = ['   1 | 4 | 7' ; '   2 | 5 | 8' ; '   3 | 6 | 9']; %
Representação na cmd line
        app.sFilled=false(1,9); % Guarda casas preenchidas
        app.ic=0; % Variável para usar imagens de teste de forma incremental
        app.pHuman = [false ; true]; % Array que guarda prioridade de turno
        app.pChoice = 0 ; % Peça escolhida pelo jogador
        app.startup = 0; % Variável de controlo para dar início ao jogo

        %% Reset aos botões %%
        app.MoveFinishedButton.Enable = 0;
        app.MoveFinishedButton.Visible = 0;

        app.STARTButton.Enable = 1;
        app.STARTButton.Visible = 1;
        end


        function FillJoints(app,jointsValues)

            app.JointTable.Data=jointsValues;

        end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
    % Se não estiverem as configs escolhidas app não pode permitir start
```

```
            if ~app.startup
                app.MoveFinishedButton.Enable = 0;
                app.MoveFinishedButton.Visible = 0;
                app.InfoBox.Text="Configure o hardware e o tabuleiro!";
                configWin(app);
            end
        % Jogo a decorrer
            if (~app.gameOver)&&(app.startup)&&(app.configDone)
                if (~app.pHuman(app.turn))
                    app.MoveFinishedButton.Enable = 0;
                    app.MoveFinishedButton.Visible = 0;
                end
                Galo_PrincP(app)
            else
               CleanUp(app);
            end

        % Jogo terminou
            if app.gameOver
                CleanUp(app);
            end

          end

          % Button pushed function: CloseButton
          function CloseBtn(app, event)
              try
                  delete(configWin(app));
              catch
              end
              delete(app);
          end

          % Botão : MoveFinishedButton
          function MoveFinishedButtonPushed(app, event)
              % Limpar botão depois de carregar

              app.MoveFinishedButton.Enable = 0;
              app.MoveFinishedButton.Visible = 0;
              startupFcn(app);
          end

          % Botão : STARTButton
          function STARTButtonPushed(app, event)
             if(app.configDone)
              % Começar jogo
              app.gameOver = 0 ; % Dar início ao jogo
              app.startup = 1 ; % Dar início ao jogo

              startupFcn(app);

              % Se a primeira jogada for do robô, dar ínicio à jogada
              if (~app.pHuman(app.turn))
                 Galo_PrincP(app)
              end
             else
              msgbox('É necessário gravar as configurações antes de
iniciar!','help');
             end
          end
```

```
        % Botão : ConfigButton
        function ConfigButtonPushed(app, event)
            configWin(app);
        end
    end

    % Inicialização de componentes
    methods (Access = private)

        % Criar UIFigure e componentes
        function createComponents(app)

            % Criar UIFigure e seconder até estar tudo criado
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 630 480];
            app.UIFigure.Name = 'UI Figure';

            % Criar InfoBox
            app.InfoBox = uilabel(app.UIFigure);
            app.InfoBox.BackgroundColor = [0.902 0.902 0.902];
            app.InfoBox.FontWeight = 'bold';
            app.InfoBox.Position = [185 449 259 22];
            app.InfoBox.Text = '';

            % Criar ConfigButton
            app.ConfigButton = uibutton(app.UIFigure, 'push');
            app.ConfigButton.ButtonPushedFcn     =     createCallbackFcn(app,
@ConfigButtonPushed, true);
            app.ConfigButton.Icon = 'config.png';
            app.ConfigButton.IconAlignment = 'center';
            app.ConfigButton.Position = [551 441 30 30];
            app.ConfigButton.Text = '';

            % Criar STARTButton
            app.STARTButton = uibutton(app.UIFigure, 'push');
            app.STARTButton.ButtonPushedFcn     =     createCallbackFcn(app,
@STARTButtonPushed, true);
            app.STARTButton.Icon = 'play.png';
            app.STARTButton.IconAlignment = 'top';
            app.STARTButton.BackgroundColor = [1 1 1];
            app.STARTButton.FontName = 'Tahoma';
            app.STARTButton.FontSize = 20;
            app.STARTButton.Position = [22 187 119 142];
            app.STARTButton.Text = 'Começar';

            % Criar MoveFinishedButton
            app.MoveFinishedButton = uibutton(app.UIFigure, 'push');
            app.MoveFinishedButton.ButtonPushedFcn  =  createCallbackFcn(app,
@MoveFinishedButtonPushed, true);
            app.MoveFinishedButton.Icon = 'waiting-icon-gif.gif';
            app.MoveFinishedButton.IconAlignment = 'right';
            app.MoveFinishedButton.FontName = 'Tahoma';
            app.MoveFinishedButton.FontSize = 24;
            app.MoveFinishedButton.Position = [161 28 302 53];
            app.MoveFinishedButton.Text = 'Jogada Terminada';

            % Criar IUGaloLabel
            app.IUGaloLabel = uilabel(app.UIFigure);
            app.IUGaloLabel.FontName = 'Tahoma';
            app.IUGaloLabel.FontSize = 20;
            app.IUGaloLabel.FontColor = [0.302 0.302 0.302];
```

```
app.IUGaloLabel.Position = [9 447 191 27];
app.IUGaloLabel.Text = 'IU Galo';

% Criar Field_9
app.Field_9 = uilabel(app.UIFigure);
app.Field_9.BackgroundColor = [0 0 0];
app.Field_9.HorizontalAlignment = 'center';
app.Field_9.FontSize = 72;
app.Field_9.FontWeight = 'bold';
app.Field_9.FontColor = [1 1 1];
app.Field_9.Position = [370 126 92 88];
app.Field_9.Text = '';

% Criar Field_6
app.Field_6 = uilabel(app.UIFigure);
app.Field_6.BackgroundColor = [0 0 0];
app.Field_6.HorizontalAlignment = 'center';
app.Field_6.FontSize = 72;
app.Field_6.FontWeight = 'bold';
app.Field_6.FontColor = [1 1 1];
app.Field_6.Position = [265 126 92 88];
app.Field_6.Text = '';

% Criar Field_3
app.Field_3 = uilabel(app.UIFigure);
app.Field_3.BackgroundColor = [0 0 0];
app.Field_3.HorizontalAlignment = 'center';
app.Field_3.FontSize = 72;
app.Field_3.FontWeight = 'bold';
app.Field_3.FontColor = [1 1 1];
app.Field_3.Position = [161 126 92 88];
app.Field_3.Text = '';

% Criar Field_8
app.Field_8 = uilabel(app.UIFigure);
app.Field_8.BackgroundColor = [0 0 0];
app.Field_8.HorizontalAlignment = 'center';
app.Field_8.FontSize = 72;
app.Field_8.FontWeight = 'bold';
app.Field_8.FontColor = [1 1 1];
app.Field_8.Position = [370 228 92 88];
app.Field_8.Text = '';

% Criar Field_7
app.Field_7 = uilabel(app.UIFigure);
app.Field_7.BackgroundColor = [0 0 0];
app.Field_7.HorizontalAlignment = 'center';
app.Field_7.FontSize = 72;
app.Field_7.FontWeight = 'bold';
app.Field_7.FontColor = [1 1 1];
app.Field_7.Position = [370 328 92 88];
app.Field_7.Text = '';

% Criar Field_5
app.Field_5 = uilabel(app.UIFigure);
app.Field_5.BackgroundColor = [0 0 0];
app.Field_5.HorizontalAlignment = 'center';
app.Field_5.FontSize = 72;
app.Field_5.FontWeight = 'bold';
app.Field_5.FontColor = [1 1 1];
app.Field_5.Position = [265 228 92 88];
```

```
            app.Field_5.Text = '';

            % Criar Field_2
            app.Field_2 = uilabel(app.UIFigure);
            app.Field_2.BackgroundColor = [0 0 0];
            app.Field_2.HorizontalAlignment = 'center';
            app.Field_2.FontSize = 72;
            app.Field_2.FontWeight = 'bold';
            app.Field_2.FontColor = [1 1 1];
            app.Field_2.Position = [161 228 92 88];
            app.Field_2.Text = '';

            % Criar Field_4
            app.Field_4 = uilabel(app.UIFigure);
            app.Field_4.BackgroundColor = [0 0 0];
            app.Field_4.HorizontalAlignment = 'center';
            app.Field_4.FontSize = 72;
            app.Field_4.FontWeight = 'bold';
            app.Field_4.FontColor = [1 1 1];
            app.Field_4.Position = [265 328 92 88];
            app.Field_4.Text = '';

            % Criar Field_1
            app.Field_1 = uilabel(app.UIFigure);
            app.Field_1.BackgroundColor = [0 0 0];
            app.Field_1.HorizontalAlignment = 'center';
            app.Field_1.FontSize = 72;
            app.Field_1.FontWeight = 'bold';
            app.Field_1.FontColor = [1 1 1];
            app.Field_1.Position = [161 328 92 88];
            app.Field_1.Text = '';

            % Criar CloseButton
            app.CloseButton = uibutton(app.UIFigure, 'push');
            app.CloseButton.ButtonPushedFcn     =     createCallbackFcn(app,
@CloseBtn, true);
            app.CloseButton.Icon = 'close icon.png';
            app.CloseButton.IconAlignment = 'center';
            app.CloseButton.BackgroundColor = [1 1 1];
            app.CloseButton.FontWeight = 'bold';
            app.CloseButton.Position = [588 441 30 30];
            app.CloseButton.Text = '';

            % Criar JointTable
            app.JointTable = uitable(app.UIFigure);
            app.JointTable.ColumnName = {'Junta'; 'Ângulo(°)'};
            app.JointTable.ColumnWidth = {50, 80};
            app.JointTable.RowName = {};
            app.JointTable.ColumnSortable = [false false];
            app.JointTable.ColumnEditable = [false false];
            app.JointTable.Position = [476 228 133 189];

            % Mostrar UIFigure agora que está tudo criado
            app.UIFigure.Visible = 'on';
        end
    end

    % Inciar app e encerrar
    methods (Access = public)

        % Construir app
```

```
        function app = UI_exported

            % Criar UIFigure e componentes
            createComponents(app)

            % Registar app
            registerApp(app, app.UIFigure)

            % Executar startupFcn
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Código que é executado quando se fecha a app
        function delete(app)

            % Apagar UIFigure
            delete(app.UIFigure)
        end
    end
end
```

Interface Utilizador – Janela de Configuração

```
classdef configWin < matlab.apps.AppBase

    % Propriedades dos componentes da app
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        DistanceYEditField          matlab.ui.control.NumericEditField
        DistnciaYEditFieldLabel     matlab.ui.control.Label
        XEditField                  matlab.ui.control.NumericEditField
        LarguraEditFieldLabel       matlab.ui.control.Label
        YEditField                  matlab.ui.control.NumericEditField
        ComprimentoLabel            matlab.ui.control.Label
        HeightEditField             matlab.ui.control.NumericEditField
        AlturaEditFieldLabel        matlab.ui.control.Label
        DistanceXEditField          matlab.ui.control.NumericEditField
        DistnciaXEditFieldLabel     matlab.ui.control.Label
        Image                       matlab.ui.control.Image
        TamanhoePosicionamentoTabuleiroLabel  matlab.ui.control.Label
        ardTestButton               matlab.ui.control.Button
        SerialPortListDropDown      matlab.ui.control.DropDown
        PortaSrieArduinoLabel       matlab.ui.control.Label
        SaveButton                  matlab.ui.control.Button
        webcamTestButton            matlab.ui.control.Button
        WebcamDropDown              matlab.ui.control.DropDown
        WebcamDropDownLabel         matlab.ui.control.Label
        IUGaloConfigLabel           matlab.ui.control.Label
    end


    properties (Access = private)
        CallingApp % Handle para UI de chamada
        info % Texto com estado da operação
    end

    properties (Access = public)
        ard  % Arduino usado
        webcam % Webcam usada
        distRX % Distância da base do robô para o tabuleiro em X
        distRY % Distância da base do robô para o tabuleiro em Y
        tabX % Comprimento em X do tabuleiro
        tabY % Comprimento em Y do tabuleiro
        tabAlt % Altura do tabuleiro em relação à base ro robô
        varNames % Nomes das variáveis a ler do ficheiro config
        varReturn % Variável que recebe os valores lidos do ficheiro config
    end

    methods (Access = private)

        function updateArdPort(app,ardPort)
           app.ard=serialport(ardPort,9600);
        end
    end


    % Callbacks
    methods (Access = private)
```

```
function startupFcn(app, appPrincipal)
app.CallingApp=appPrincipal;

%Inicializar variáveis para enviar para o programa principal
app.varNames={'distRX','distRY','tabX','tabY','tabAlt'};
app.varReturn=array2table(zeros(0,size(app.varNames,2)));

%Ler o ficheiro com config e substituir os valores nas respectivas
%variáveis
app.varReturn=readCfgfile(app.varNames);

for i=1:size(app.varReturn,2)
    if (app.varReturn.Properties.VariableNames(i))=="distRX"
        app.distRX=app.varReturn{1,i};
        app.DistanceXEditField.Value=app.distRX;
    elseif (app.varReturn.Properties.VariableNames(i))=="distRY"
        app.distRY=app.varReturn{1,i};
        app.DistanceYEditField.Value=app.distRY;
    elseif (app.varReturn.Properties.VariableNames(i))=="tabX"
        app.tabX=app.varReturn{1,i};
        app.XEditField.Value=app.tabX;
    elseif (app.varReturn.Properties.VariableNames(i))=="tabY"
        app.tabY=app.varReturn{1,i};
        app.YEditField.Value=app.tabY;
    elseif (app.varReturn.Properties.VariableNames(i))=="tabAlt"
        app.tabAlt=app.varReturn{1,i};
        app.HeightEditField.Value=app.tabAlt;
    end
end
% Configurar arduino,

camList=webcamlist;
app.WebcamDropDown.Items=camList;
if ~(app.webcam)
app.webcam=webcam(camList(1).Value);
end



end

% Alterar valor: WebcamDropDown
function WebcamDropDownValueChanged(app, event)
    app.webcam = webcam(app.WebcamDropDown.Value);
end

% Botão: webcamTestButton
function webcamTestButtonPushed(app, event)

     % Verificar se existe alguma webcam selecionada
        try
        preview(app.webcam)
        pause(10);
        closePreview(app.webcam)
        catch
             msgbox("Camera was not found",'Error','error')
        end

end
```

```
        % Botão: SaveButton
        function SaveButtonPushed(app, event)

            try app.ard=serialport(app.SerialPortListDropDown.Value,9600);
                if (app.ard.Port ~= app.SerialPortListDropDown.Value)
                  % Atualizar porta do arduino

                  app.ard=serialport(app.SerialPortListDropDown.Value,9600);
                end
                set(app.ard, 'timeout',.5);
                app.webcam = webcam(app.WebcamDropDown.Value);
                  % Atualizar webcam
                app.info=('');
            catch
                app.info=('Arduino não está configurado');
            end

            % Atualizar a app principal com os valores


updateCfg(app.CallingApp,app.ard,app.webcam,app.info,app.distRX,app.distRY,ap
p.tabX,app.tabY,app.tabAlt);

            delete(app);
        end

        % Abrir Drop down: SerialPortListDropDown
        function SerialPortListDropDownOpening(app, event)
                app.SerialPortListDropDown.Items = serialportlist("all");
        end

        % Botão: ardTestButton
        function ardTestButtonPushed(app, event)
             % Verificar se arduino corresponde
              Test_BraccioToPos(app);
        end

        % Alterar valor: SerialPortListDropDown
        function SerialPortListDropDownValueChanged(app, event)
            app.ard=serialport(app.SerialPortListDropDown.Value,9600);
        end
    end

    % Inicializar componentes
    methods (Access = private)

        % Criar janela e componentes
        function createComponents(app)

            % Criar janela e esconder até estar tudo criado
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 347 359];
            app.UIFigure.Name = 'UI Figure';
            app.UIFigure.WindowStyle = 'modal';

            % Criar IUGaloConfigLabel
            app.IUGaloConfigLabel = uilabel(app.UIFigure);
            app.IUGaloConfigLabel.FontName = 'Tahoma';
            app.IUGaloConfigLabel.FontSize = 20;
            app.IUGaloConfigLabel.FontColor = [0.302 0.302 0.302];
```

```
            app.IUGaloConfigLabel.Position = [9 326 208 27];
            app.IUGaloConfigLabel.Text = 'IU Galo - Configuração';

            % Criar WebcamDropDownLabel
            app.WebcamDropDownLabel = uilabel(app.UIFigure);
            app.WebcamDropDownLabel.HorizontalAlignment = 'right';
            app.WebcamDropDownLabel.Position = [27 230 53 22];
            app.WebcamDropDownLabel.Text = 'Webcam';

            % Criar WebcamDropDown
            app.WebcamDropDown = uidropdown(app.UIFigure);
            app.WebcamDropDown.Items = {};
            app.WebcamDropDown.ValueChangedFcn    =    createCallbackFcn(app,
@WebcamDropDownValueChanged, true);
            app.WebcamDropDown.Position = [113 230 149 22];
            app.WebcamDropDown.Value = {};

            % Criar webcamTestButton
            app.webcamTestButton = uibutton(app.UIFigure, 'push');
            app.webcamTestButton.ButtonPushedFcn    =    createCallbackFcn(app,
@webcamTestButtonPushed, true);
            app.webcamTestButton.FontWeight = 'bold';
            app.webcamTestButton.Position = [272 230 55 22];
            app.webcamTestButton.Text = 'Test';

            % Criar SaveButton
            app.SaveButton = uibutton(app.UIFigure, 'push');
            app.SaveButton.ButtonPushedFcn      =      createCallbackFcn(app,
@SaveButtonPushed, true);
            app.SaveButton.Position = [244 19 86 22];
            app.SaveButton.Text = 'Gravar';

            % Criar PortaSrieArduinoLabel
            app.PortaSrieArduinoLabel = uilabel(app.UIFigure);
            app.PortaSrieArduinoLabel.HorizontalAlignment = 'right';
            app.PortaSrieArduinoLabel.Position = [22 276 110 22];
            app.PortaSrieArduinoLabel.Text = 'Porta Série Arduino';

            % Criar SerialPortListDropDown
            app.SerialPortListDropDown = uidropdown(app.UIFigure);
            app.SerialPortListDropDown.Items = {};
            app.SerialPortListDropDown.DropDownOpeningFcn                    =
createCallbackFcn(app, @SerialPortListDropDownOpening, true);
            app.SerialPortListDropDown.ValueChangedFcn                       =
createCallbackFcn(app, @SerialPortListDropDownValueChanged, true);
            app.SerialPortListDropDown.Position = [150 276 112 22];
            app.SerialPortListDropDown.Value = {};

            % Criar ardTestButton
            app.ardTestButton = uibutton(app.UIFigure, 'push');
            app.ardTestButton.ButtonPushedFcn     =     createCallbackFcn(app,
@ardTestButtonPushed, true);
            app.ardTestButton.FontWeight = 'bold';
            app.ardTestButton.Position = [274 276 55 22];
            app.ardTestButton.Text = 'Test';

            % Criar TamanhoePosicionamentoTabuleiroLabel
            app.TamanhoePosicionamentoTabuleiroLabel = uilabel(app.UIFigure);
            app.TamanhoePosicionamentoTabuleiroLabel.FontWeight = 'bold';
            app.TamanhoePosicionamentoTabuleiroLabel.Position  =  [23  183  221
27];
```

```
app.TamanhoePosicionamentoTabuleiroLabel.Text    =    'Tamanho    e
Posicionamento Tabuleiro';

% Criar Imagem do tabuleiro
app.Image = uiimage(app.UIFigure);
app.Image.Position = [96 58 100 100];
app.Image.ImageSource = 'board.jpg';

% Criar DistnciaXEditFieldLabel
app.DistnciaXEditFieldLabel = uilabel(app.UIFigure);
app.DistnciaXEditFieldLabel.HorizontalAlignment = 'right';
app.DistnciaXEditFieldLabel.FontWeight = 'bold';
app.DistnciaXEditFieldLabel.Position = [200 136 70 22];
app.DistnciaXEditFieldLabel.Text = 'Distância X';

% Criar DistanceXEditField
app.DistanceXEditField = uieditfield(app.UIFigure, 'numeric');
app.DistanceXEditField.Position = [277 136 54 22];

% Criar AlturaEditFieldLabel
app.AlturaEditFieldLabel = uilabel(app.UIFigure);
app.AlturaEditFieldLabel.HorizontalAlignment = 'right';
app.AlturaEditFieldLabel.FontWeight = 'bold';
app.AlturaEditFieldLabel.Position = [217 73 40 22];
app.AlturaEditFieldLabel.Text = 'Altura';

% Criar HeightEditField
app.HeightEditField = uieditfield(app.UIFigure, 'numeric');
app.HeightEditField.Position = [276 73 54 22];

% Criar ComprimentoLabel
app.ComprimentoLabel = uilabel(app.UIFigure);
app.ComprimentoLabel.HorizontalAlignment = 'center';
app.ComprimentoLabel.FontWeight = 'bold';
app.ComprimentoLabel.Position = [10 129 83 22];
app.ComprimentoLabel.Text = 'Comprimento';

% Criar YEditField
app.YEditField = uieditfield(app.UIFigure, 'numeric');
app.YEditField.HorizontalAlignment = 'center';
app.YEditField.Position = [33 99 36 22];

% Criar LarguraEditFieldLabel
app.LarguraEditFieldLabel = uilabel(app.UIFigure);
app.LarguraEditFieldLabel.HorizontalAlignment = 'center';
app.LarguraEditFieldLabel.FontWeight = 'bold';
app.LarguraEditFieldLabel.Position = [119 12 50 22];
app.LarguraEditFieldLabel.Text = 'Largura';

% Criar XEditField
app.XEditField = uieditfield(app.UIFigure, 'numeric');
app.XEditField.HorizontalAlignment = 'center';
app.XEditField.Position = [122 33 42 22];

% Criar DistnciaYEditFieldLabel
app.DistnciaYEditFieldLabel = uilabel(app.UIFigure);
app.DistnciaYEditFieldLabel.HorizontalAlignment = 'right';
app.DistnciaYEditFieldLabel.FontWeight = 'bold';
app.DistnciaYEditFieldLabel.Position = [200 105 70 22];
app.DistnciaYEditFieldLabel.Text = 'Distância Y';
```

```
            % Criar DistanceYEditField
            app.DistanceYEditField = uieditfield(app.UIFigure, 'numeric');
            app.DistanceYEditField.Position = [277 105 54 22];

            % Mostrar janela agora com tudo criado
            app.UIFigure.Visible = 'on';
        end
    end

    % Iniciar e encerrar app
    methods (Access = public)

        % Construir app
        function app = configWin_exported(varargin)

            runningApp = getRunningApp(app);

            % Verificar se app está a correr
            if isempty(runningApp)

                % Criar componentes
                createComponents(app)

                % Registar app
                registerApp(app, app.UIFigure)

                % Excutar função startupFcn
                runStartupFcn(app, @(app)startupFcn(app, varargin{:}))
            else

                % Alterar foco para app atual
                figure(runningApp.UIFigure)

                app = runningApp;
            end

            if nargout == 0
                clear app
            end
        end

        % Código que é executado quando se fecha a app
        function delete(app)

            % Fechar app
            delete(app.UIFigure)
        end
    end
end
```

Funções – robotKinBraccio

```
function [joints] = robotKinBraccio(ard,robotMove,testMode, distRX, distRY,
tabX, tabY, tabAlt)

% Verificar se a jogada está dentro dos limites

if  (1 <= robotMove)&&(robotMove <= 9)
    %% Cálculos cinemática robô

    % Se algum dos valores de parametrização for 0 inicializar com valores
    % por defeito


    if testMode
        % Distância em relação ao tabuleiro
        distRY=-0.175;
        distRX=-0.22;

        % Tamanho tabuleiro em m
        tabY=0.1;
        tabX=0.1;

        % Altura em relação à base do braço
        tabAlt=0.1;
    end


    %% Inicializar tabuleiro e importar modelo do manipulador

    cel=0;
    tabCoord=zeros(9,2); % Dimensoes Hor x Ver

      %             9 | 6 | 3
      %             8 | 5 | 2
      % X           7 | 4 | 1
      %<----------X Robô % Posição do braço
      %           |
      %           | Y
      %           \/
    % Determinar coordenadas do centro das células de acordo com dimensão do
    % tabuleiro


    for i=1:3
        for j=1:3
            cel=cel+1;

            tabCoord(cel,1)=distRX+(tabX/6)+(tabX/3*(i-1));
            tabCoord(cel,2)=distRY+(tabY/6)+(tabY/3*(3-j));
        end
    end

    name = strcat('Braccio jogada : ',int2str(robotMove),...
        ' X: ',num2str(tabCoord(robotMove,1)),...
        ' Y: ',num2str(tabCoord(robotMove,2)));
```

```
    % Importar o modelo do manipulador
    lbr                                                                    =
importrobot('C:\Users\ruben\Dropbox\mest\braccio\urdf\braccio.urdf');
    lbr.DataFormat = 'row';

%% Definir limites das juntas (usado apenas em desenvolvimento)

% Limitar rotação da gripper de forma a não obstruir a abertura
  lbr.Bodies{1, 5}.Joint.PositionLimits(1,:)=[deg2rad(134),deg2rad(136)];

%% Define end-effector body name

    gripper = 'gripper_pickup';

    % Definir dimensões das peças
    scale=1;
    pieceHeight = tabAlt + 0.02 * scale; %
    pieceP_X= tabCoord(robotMove,1); %
    pieceP_Y= tabCoord(robotMove,2); %
    pieceP_Z= pieceHeight/2 ; % altura do tabuleiro + 10mm, peça tem ~20mm
    piecePosition = [pieceP_X, pieceP_Y, pieceP_Z];

    % O método implementado usa objectos como elos de junção para a
    % configuração do robô.

    body = rigidBody('pieceFrame');
    setFixedTransform(body.Joint, trvec2tform(piecePosition))
    addBody(lbr, body, lbr.BaseName);

    % A trajetória vai usar 5 pontos chave sendo o primeiro a configuração
    % definida pelo ficheiro modelo (posição home)

    numWaypoints = 5;
    q0 = homeConfiguration(lbr);
    qWaypoints = repmat(q0, numWaypoints, 1);

    %% Criar a função generalizedInverseKinematics, inputs:
    % Limites cartesianos - limita a posição da gripper, 'cartesian'
    % Alvo de posição - Define a distância entre a gripper e o objecto,
'position'
    % Restrição de alinhamento - Alinha o eixo da gripper com o objecto,
    % 'aiming' (este não está a ser utilizado pois não permitia que o
    % manipulador atingisse todas as posições pretendidas)
    % Alvo de orientação - Mantém a orientação da gripper ao aproximar-se
    % do objecto, 'orientation'
    % Limite de junta - Limita a variação das juntas entre cada ponto, 'joint'

    gik      =      generalizedInverseKinematics('RigidBodyTree',      lbr,
'ConstraintInputs', {'cartesian','position','orientation','joint'});

    %% Criar objectos que vão ser usados como restrições para a função

    % Limites cartesianos para a gripper
    % (5mm acima do tabuleiro apenas, x e y não se está a limitar)

    heightAboveTable = constraintCartesianBounds(gripper);
    heightAboveTable.Bounds = [-inf, inf;
                               -inf, inf;
                               0.05, inf];
```

```
    % Tolerância para distância relativa entre a gripper e o objecto

    distanceFromPiece = constraintPositionTarget('pieceFrame');
    distanceFromPiece.ReferenceBody = gripper;
    distanceFromPiece.PositionTolerance = 0.0001;

    % Restrição de alinhamento (não está a ser usada na função)
    % Os testes foram feitos com o elo 4 (paralelo e coincidente com o eixo
    % da gripper) no entanto apesar de colocar a gripper na posição ideal
    % para colocação esta não permite atingir todas as posições)

    alignWithPiece = constraintAiming('link4');
    alignWithPiece.TargetPoint = [0, 0, -50];

    % Introduzir os limites das juntas do modelo na função

    limitJointChange = constraintJointBounds(lbr);

    % Criar uma tolerância limite de 1 grau para a orientação da gripper de
    % forma a que a mesma corresponda ao valor especificado pela
    % propriedade TargetOrientation. Usado na aproximação ao ojbecto.

    fixOrientation = constraintOrientationTarget(gripper);
    fixOrientation.OrientationTolerance = deg2rad(1);

    % Os objectos de junta da função usam uma propriedade para
    % resolver conflitos nos valores de orientação e posição das juntas nas
configurações.
    % Neste caso esta propriedade foi dexada a 0 de forma a desabilitar a
    % mesma.

    limitJointChange.Weights = zeros(size(limitJointChange.Weights));
    fixOrientation.Weights = 0;

    % Definir a distância entre a gripper e o objecto em cada ponto.

    intermediateDistance = 0.001;
    distanceFromPiece.TargetPosition = [0,0,intermediateDistance];

    % Determinar pontos pertencentes à trajetória

      [qWaypoints(2,:),~] = gik(q0, heightAboveTable, ...
                    distanceFromPiece, fixOrientation, ...
                    limitJointChange);

    % Habilitar os limites de orientação das juntas para determinar a posição
final

    limitJointChange.Weights = ones(size(limitJointChange.Weights));
    fixOrientation.Weights = 1;

    % Desabilitar a função de alinhamento com a peça visto que a função de
    % limite de orientação e alinhamento das juntas tem o mesmo propósito.

    alignWithPiece.Weights = 0;

    % Definir a orientação baseada na configuração anterior (qWaypoints(2,:))
    % Resolver a transformada da gripper para a base do modelo e converter
    % num vetor de posição.
```

```
    fixOrientation.TargetOrientation = ...
        tform2quat(getTransform(lbr,qWaypoints(2,:),gripper));

    % Definir a distância entre a gripper e o objecto

    finalDistanceFromPiece = 0.001;
    distanceFromPieceValues          =          linspace(intermediateDistance,
finalDistanceFromPiece, numWaypoints-1);

    % Definir o valor máximo de alteração nos valores das juntas entre cada
    % ponto

    maxJointChange = deg2rad(10);

    % Chamar a função para calcular os restantes pontos

    for k = 3:numWaypoints
        % Update the target position.
        distanceFromPiece.TargetPosition(3) = distanceFromPieceValues(k-1);
        % Usar restrições para os valores das juntas de forma a que
        % permaneçam perto dos valores anteriores
        limitJointChange.Bounds = [qWaypoints(k-1,:)' - maxJointChange, ...
                                   qWaypoints(k-1,:)' + maxJointChange];
        % Resolver configuração e adicionar ao array de pontos
        [qWaypoints(k,:),~] = gik(qWaypoints(k-1,:), ...
                                              heightAboveTable, ...
                                              distanceFromPiece, ...
                                              fixOrientation,
limitJointChange);
    end

    % Determinar a configuração entre pontos para gerar uma trajetória

    framerate = 15;
    r = rateControl(framerate);
    tFinal = 10;
    tWaypoints = [0,linspace(tFinal/2,tFinal,size(qWaypoints,1)-1)];
    numFrames = tFinal*framerate;
    qInterp = pchip(tWaypoints,qWaypoints',linspace(0,tFinal,numFrames))';

    % Calcular a posição da gripper para todas as configurações
    % determinadas

    gripperPosition = zeros(numFrames,3);

    for k = 1:numFrames
        gripperPosition(k,:) = tform2trvec(getTransform(lbr,qInterp(k,:), ...
                                                        gripper));
    end


    qi=rad2deg(qInterp(size(qInterp,1),:));


%% Enviar para manipulador os valores calculados

joints=zeros(6,6); % Valores das juntas para enviar para a aplicação
```

```
if ~isempty(qi) % Send to position
    for i=1:size(qi,2)
        if qi(1,i) < 0
           qi(1,i)=0;
        elseif qi(1,i) > 180
              qi(1,i)=180;
        end
        joints(i,1)=i;
        joints(:,2)=qi(1,:);
        switch i
            case 1
                M1=qi(1,i);
            case 2
                M2=qi(1,i);
            case 3
                M3=qi(1,i);
            case 4
                M4=qi(1,i);
            case 5
                M5=qi(1,i);
            case 6
                if qi(1,i)>73
                    qi(1,i)=73;
                end
                %M6=qi(1,i);
        end
    end
else
    % M6 trata-se da abertura e fecho da gripper, não  é importante o
    % resultado da cinemática, o controlo será feito como necessário

    M1=0; M2=M1; M3=M2; M4=M3; M5=M4; %M6=M5;
end

     %% Ir buscar peça
     % elseif robotMove==0
             for i=1:4
                    moveDone="";
                    switch i
                        case 1   % Posição de pickup inicial já com base na
pos, abrir garra

moveDone=comBraccio(ard,1,55,30,180,180,0,10);
                        case 2  % Posição de pickup em baixo

moveDone=comBraccio(ard,1,55,72,180,180,0,10);
                        case 3  % Posição de pickup em baixo, fechar garra

moveDone=comBraccio(ard,1,55,72,180,180,0,100);
                        case 4  % Posição de pickup inicial em cima com garra
fechada
                                % (espera por movimento)

moveDone=comBraccio(ard,1,55,30,180,180,0,100);
                    end
                      while moveDone==""

                      end
             end
    %  end
        %% Colocar peça
             for i=1:5
```

```
                    moveDone="";
                    switch i
                        case 1  % Posição de espera com peça

moveDone=comBraccio(ard,1,M1,30,180,180,0,100);
                        case 2  %Enviar robot para a posição

moveDone=comBraccio(ard,1,M1,M2,M3,M4,M5,100);
                        case 3  %Abrir garra
                                moveDone=comBraccio(ard,1,M1,M2,M3,M4,M5,80);
                        case 4  %Voltar para posição de espera
                                pause(5);

moveDone=comBraccio(ard,1,M1,45,180,180,0,80);
                        case 5  %Voltar para posição de espera

moveDone=comBraccio(ard,1,120,45,180,180,0,100);
                    end
                    % Mensagem diagnóstico para indicar que estamos à espera
do posicionamento
                      while moveDone==""
                        printf("Waiting for positioning")
                      end
            end

end

%%
end
```

Funções – getBoard

```
function [move, casasPreenchidas] = getBoard(webcam, jogada, casasPreenchidas)
%% Inicialização

move=0;
bbrectSize=0;
newPartCounter=0;


%%           -------------------
%           Ler e tratar imagem
% --------------------------------------

%          -------------------
% --------------------------------------

% Iniciar webcam e tirar snapshot

img_g = snapshot(webcam);

% Se a webcam não tiver iniciado correctamente tirar outro snapshot

if(~any(img_g))
    fprintf('Imagem inválida!')
    preview(webcam)
    pause(1);
    img_g = snapshot(webcam);
    closePreview(webcam)
end


%% Encontrar o campo de jogo

masked= mat2gray(img_g,[100 105]); % Transformar em grayscale
masked2D=masked(:, :, 1); % Extrair 1 das 3 layers da imagem

Ifill = imfill(masked2D,'holes'); % Preencher o interior dos objectos
encontrados

Iarea = bwareaopen(Ifill,100); % Remove objectos com vizinhanca inferior
à definida
Ifinal = logical(Iarea); % Define valores por pixel com base na sua vizinhanca
quad = regionprops(Ifinal,'boundingbox'); % Determinar rect. na imagem com as
coordenadas e comprimentos por eixo

for cnt = 1 : numel(quad) % Determinar a area dos  rect. encontrados
    bb = quad(cnt).BoundingBox;
    bbrectSize(cnt)= bb(3)*bb(4);
    if (bbrectSize(cnt)<210000 && bbrectSize(cnt)>150000)
       aJogo=quad(cnt).BoundingBox;
    end
end


%% Recortar e redimensionar área de jogo
```

```
boardCrop=imcrop(masked2D,aJogo);
board=imresize(boardCrop,[405,405]); %Definir valores fixos para dimensao da
imagem do tabuleiro;
board=imbinarize(board,0.30);
quadInside = regionprops(board,'BoundingBox'); % Determinar rect. na imagem
com as coordenadas e comprimentos por eixo

%% Determinar posição e identificar peças
bbrectSizeInside=0; i=1; partsInBoard=zeros(1,4);

%Verificar a posição das peças e retirar coordenadas e dimensões

for cnt = 1 : numel(quadInside)
   bbInside = quadInside(cnt).BoundingBox;
   bbrectSizeInside(cnt)= bbInside(3)*bbInside(4);
   if (bbrectSizeInside(cnt)<5000)&(bbrectSizeInside(cnt)>1000)
       partsInBoard(i,:)=quadInside(cnt).BoundingBox;
       i=i+1;
   end
end

%Iniciar variáveis para guardar peças encontradas
saveO=zeros(size(partsInBoard)); saveX=zeros(size(partsInBoard));


% Verificar peças encontradas e identificar O e X - Recortar a imagem e
% remover moldura

partBorder=3;

for i=1:size(partsInBoard,1)
    partCrop=imcrop(board,partsInBoard(i,:));
    rpartCrop=centerCropWindow2d(size(partCrop),[size(partCrop,1)-
partBorder,size(partCrop,2)-partBorder]);
    ypartCrop=imcrop(partCrop,rpartCrop);
    if imfindcircles(partCrop,[3 8],'ObjectPolarity','bright')
        saveO(i,:)=partsInBoard(i,:);
    else
        saveX(i,:)=partsInBoard(i,:);
    end
end


%% Valores das linhas e colunas definidos com base no resize aplicado à imagem
para 405x405
heightComp=15; % compensação para deslocamento por altura da peça devido ao
ângulo de aquisição
lin1=125-heightComp; lin2=235-heightComp;
col1=150; col2=265;

%% Verificar colocação da peça e identificar casas

for cnt=1 : size(partsInBoard,1)
    if saveO(cnt,3)~=0
        if (saveO(cnt,1) < col1)
            saveO(cnt,3)=1;
        elseif (saveO(cnt,1) < col2)
            saveO(cnt,3)=2;
        elseif (saveO(cnt,1) > col2)
            saveO(cnt,3)=3;
```

```
        end
    end
    if saveO(cnt,4)~=0
        if (saveO(cnt,2) < lin1)
            saveO(cnt,4)=1;
        elseif (saveO(cnt,2) < lin2)
            saveO(cnt,4)=2;
        elseif (saveO(cnt,2) > lin2)
            saveO(cnt,4)=3;
        end
    end

    if saveX(cnt,3)~=0
        if (saveX(cnt,1) < col1)
            saveX(cnt,3)=1;
        elseif (saveX(cnt,1) < col2)
            saveX(cnt,3)=2;
        elseif (saveX(cnt,1) > col2)
            saveX(cnt,3)=3;
        end
    end
    if saveX(cnt,4)~=0
        if (saveX(cnt,2) < lin1)
            saveX(cnt,4)=1;
        elseif (saveX(cnt,2) < lin2)
            saveX(cnt,4)=2;
        elseif (saveX(cnt,2) > lin2)
            saveX(cnt,4)=3;
        end
    end
end

for cnt=1 : size(partsInBoard,1)
    if saveO(cnt,3)~=0
        saveO(cnt,3) = saveO(cnt,4) + (saveO(cnt,3)-1)*3;
    end
    if saveX(cnt,3)~=0
        saveX(cnt,3) = saveX(cnt,4) + (saveX(cnt,3)-1)*3;
    end
end
%% Guardar valores já preenchidos no tabuleiro

    for cnt=1 : size(partsInBoard,1)
            if (saveO(cnt,3) ~=0) && (casasPreenchidas(saveO(cnt,3))== false)
&& (jogada=="O")
                casasPreenchidas(saveO(cnt,3))=true;
                newPartCounter=newPartCounter+1;
                move=saveO(cnt,3);
            elseif (saveX(cnt,3)) ~=0 && (casasPreenchidas(saveX(cnt,3))) ==
false && (jogada=="X")
                newPartCounter=newPartCounter+1;
                casasPreenchidas(saveX(cnt,3))=true;
                move=saveX(cnt,3);
            end
    end
end
```

Funções – comBraccio

```
function [done] = comBraccio(ard,start,M1,M2,M3,M4,M5,M6)

warning('off')

comStr=strcat(num2str(start),",",num2str(M1),",",...
              num2str(M2),",",num2str(M3),",",...
              num2str(M4),",",num2str(M5),",",num2str(M6));

exit="Não";
aux=0;
done="";

    while exit=="Não"

            readData=readline(ard);

            if aux==2
                write(ard,comStr,'string');
                aux=0;
            end

            if isempty(readData)
                readData="Não";
            end

        if readData=="Posicionado"
            exit='Sim';
        else
            exit='Não';

        end

        aux=aux+1;
        pause(1)

    end

    done="Completed";
end
```

Funções – readCfgFile

```
function [varReturn] = readCfgfile(varNames)

% Definir tabela de variáveis a preencher
if isempty(varNames)
varNames={'distRX','distRY', 'tabX','tabY','tabAlt'};
end
varReturn=array2table(zeros(0,size(varNames,2)));
varReturn.Properties.VariableNames= varNames;

% Formatação do ficheiro csv
fieldFormat = '%C%f';
% Ler ficheiro csv
fid = readtable('config.csv','Format',fieldFormat);

% Preencher valores na tabela de variáveis
for i=1:size(fid,1)
field=fid{i,1};
    for j=1:size(varReturn,2)
        if  field==varReturn.Properties.VariableNames(j)
            varReturn{1,j}=fid{i,2};
        end

    end
end

end
```