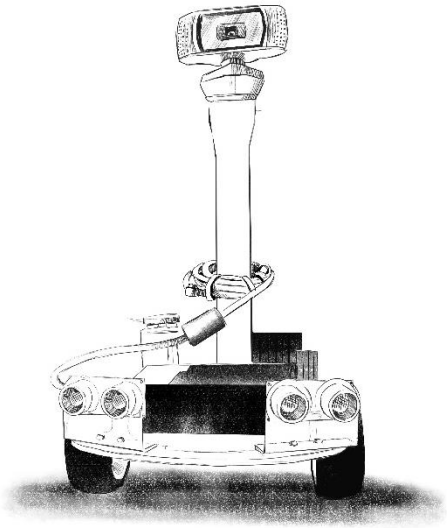
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrotécnica Energia e Automação



Navegação de um Robô Móvel por Processamento de Imagem

ALEXANDRE DUARTE GOULÃO
(Licenciado em Engenharia Eletrotécnica)

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

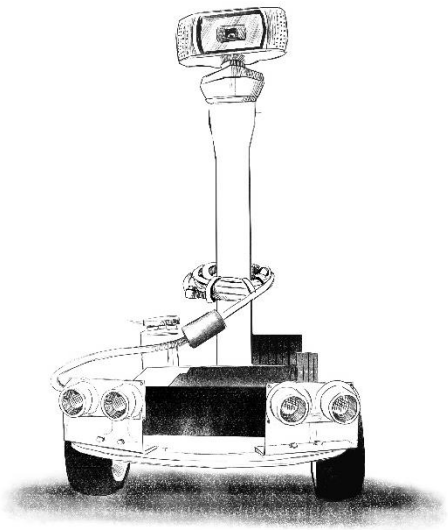
Orientadora:
Prof.ª Doutora Maria da Graça Vieira de Brito Almeida

Júri:
Presidente: Prof. Doutor Luís Manuel dos Santos Redondo
Vogais:
Grau e Nome

Setembro de 2022

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrotécnica Energia e Automação



Navegação de um Robô Móvel por Processamento de Imagem

ALEXANDRE DUARTE GOULÃO
(Licenciado em Engenharia Eletrotécnica)

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

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

Júri:
Presidente: Prof. Doutor Luís Manuel dos Santos Redondo
Vogais:
Grau e Nome

Setembro de 2022

Agradecimentos

O desenvolvimento desta dissertação só foi possível com o apoio e ajuda de um grupo de diversas pessoas que me acompanharam, não só neste projeto, mas também ao longo da minha jornada académica. Deste modo agradeço:

- À Professora Doutora Maria da Graça Almeida, pela orientação, disponibilidade, pela ajuda ao longo da elaboração do projeto e por todo o material disponibilizado para a construção do robô;
- Aos técnicos superiores, Nuno Santos e César Ferrolho, por todo o apoio no laboratório, quer na realização de testes, bem como na ajuda técnica de soldadura;
- À minha namorada, pela ajuda durante esta fase, pela presença, pelo carinho e também pelo apoio, não só durante esta dissertação, mas também ao longo do meu percurso enquanto estudante;
- Aos meus pais, pelo apoio e confiança investido em mim desde sempre e também pela ajuda durante o processo de construção do robô;
- Aos meus amigos, pela ajuda, pela companhia e por me terem acompanhado no meu percurso académico.

Resumo

A robótica móvel é um campo científico notável, que pretende automatizar a movimentação de um mecanismo com uma designada função, para que este tenha a habilidade de realizar as suas tarefas sem permanecer numa posição fixa. Para este efeito, utiliza uma variedade de recursos para identificar o seu ambiente durante o deslocamento. Um dos métodos mais significativos, para o auxílio do reconhecimento do espaço envolvente, é o processamento de imagem.

Esta dissertação visa o desenvolvimento de um robô móvel, composto por uma câmera, numa posição alta, com o intuito de realizar a aquisição da imagem, e sensores para evitar obstáculos. O utilizador interage com o robô móvel através do número de dedos da mão. Assim, o algoritmo de processamento de imagem desenvolvido permite controlar a movimentação do robô.

O robô móvel foi estruturado com uma perspetiva simples e de baixo custo, mas que também conseguisse desempenhar o seu papel eficientemente. Para realizar o controlo do mesmo é utilizado um *Raspberry Pi model 3B+*.

O foco principal deste projeto está presente no algoritmo de processamento de imagem, desenvolvido em *python*, com auxílio da biblioteca *OpenCV*. Empregando métodos de extração de características e de obtenção de contornos, foi possível identificar a mão do utilizador e o número de dedos apresentados. Este programa permite assim, controlar o robô móvel como um tipo de comando, em que consoante o número de dedos exibidos irá executar um diferente tipo de movimento.

Palavras-Chave

Robótica, Robô Móvel, Processamento de Imagem, *Raspberry Pi*, Deteção de Dedos.

Abstract

Mobile robotics is a remarkable scientific field, that aims to automate the movement of a mechanism with a designated function, so that it has the ability to perform its tasks without remaining in a fixed position. For this purpose, it utilizes a variety of features to identify its environment while moving. One of the most significant methods for assisting in the recognition of the surrounding space, is image processing.

This dissertation aims to develop a mobile robot, composed of a camera in an elevated position, in order to perform image acquisition, and sensors to avoid obstacles. The user interacts with the mobile robot through the number of fingers on the hand. Thus, the developed image processing algorithm allows control of the robot's movement.

The mobile robot was structured with a low-cost perspective and to be relatively simple, but also able to perform its role efficiently. A Raspberry Pi model 3B+ is used to control it.

The main focus of this project is present in the image processing algorithm, developed in python, with the aid of the OpenCV library. By employing methods of feature and contour extraction, it was possible to identify the user's hand and the number of fingers displayed. From this program, it is possible to control the mobile robot as a type of remote, where depending on the number of fingers displayed it will perform a different movement function.

Keywords

Robotics, Mobile Robot, Image Processing, Raspberry Pi, Finger Detection.

Lista de Abreviaturas, Acrónimos e Siglas

B&W – Preto e Branco (de *Black and White*)

CCD – *Charged-coupled Device*

CMOS – *Complementary Metal Oxide Semiconductor*

CMYB – Sistema de cores: Ciano, Magenta, Amarelo e Preto (de *Cyan*, *Magenta*, *Yellow and Black*)

DC – Corrente Direta (de *Direct Current*)

FPS – *Frames* por segundo

GPIO – *General Purpose Input*/*Output*

GPS – Sistema de Posicionamento Global (de *Global Positioning System*)

HSV – Sistema de cores: Matiz, Saturação e Brilho/Valor (de *Hue*, *Saturation*, *Value*)

IA – Inteligência Artificial

IR – Infravermelho (de *Infrared*)

JPL – *Jet Propulsion Laboratory*

LIDAR – *Light Detection and Ranging*

PWM – Modulação por Largura de Pulso (de *Pulse-Width Modulation*)

MOSFET – *Metal Oxide Semiconductor Field-effect Transistor*

RGB – Sistema de cores: Vermelho, Verde e Azul (de *Red*, *Green*, *Blue*)

SLAM – *Simultaneous Localization and Mapping*

USB – *Universal Serial Bus*

Índice Geral

Índice de Figuras

Índice de Tabelas

1. Introdução

Desde o início da humanidade, que o ser humano sonhou com aplicações ou ferramentas que conseguissem simplificar as suas tarefas de trabalho ou do quotidiano, com o objetivo de facilitar o processo e torná-lo mais simples, rápido, seguro e preciso.

Para este fim surge a robótica, que consiste num ramo científico e tecnológico, encarregue de automatizar vários processos e equipamentos, controlar e realizar ações e desenvolver ferramentas, tudo com o propósito de assistir, ajudar e até substituir os seres humanos.

1.1. Enquadramento e Motivação

A robótica consiste num domínio extremamente vasto, com a possibilidade para o desenvolvimento de um número ilimitado de aplicações nas diversas áreas da ciência. Apesar de muito já ter sido feito nesta área, continua a ser um mundo com um potencial crescente de aplicações, quer na indústria, como ao nível de assistência pessoal e ainda do ponto de vista lúdico.

Este trabalho visa o desenvolvimento de um veículo semiautónomo, que consiga navegar por um ambiente desconhecido e evitar colisões com obstáculos. Este sistema terá o auxílio do processamento de imagem e de diversos sensores, com vista a tornar o veículo praticamente autónomo.

A robótica móvel é um dos tópicos pelo qual tenho um interesse especial, sendo que a minha motivação com esta dissertação é aprofundar o meu conhecimento sobre a área e também poder contribuir para o desenvolvimento futuro deste ramo, seja com uma perspetiva de trabalho ou simplesmente do quotidiano.

Irei também, com este projeto, aprofundar os meus conhecimentos de processamento de imagem, inclusive na aplicação e estudo de outros algoritmos, conciliando o *hardware* e respetiva programação do robô móvel.

1.2. Objetivos

Esta dissertação irá centrar-se, maioritariamente, nas temáticas da robótica móvel e do processamento de imagem, tendo por objetivo desenvolver um robô móvel, guiado por uma câmera assistida por visão computacional, utilizando também sensores para evitar colisões.

Para além da informação que irá receber dos respetivos sensores de proximidade, o robô vai conseguir seguir ordens fornecidas pelo utilizador através de um algoritmo de processamento de imagem. Isto é, o utilizador poderá comunicar um conjunto de ordens ao robô móvel, através do número de dedos que apresenta à câmera.

O controlo do robô móvel será efetuado por um controlador, o *Raspberry Pi*.

1.3. Metodologia de Trabalho

Este projeto foi desenvolvido por fases, sendo as tarefas mencionadas em baixo as mais indicativas das numerosas atividades necessárias para alcançar os objetivos especificados:

- Pesquisa bibliográfica;
- Estudo sobre o funcionamento dos robôs móveis diferenciais, com especial ênfase nos robôs móveis com 3 rodas;
- Estudo dos diversos controladores e escolha do controlador a utilizar;
- Escolha da plataforma de programação a utilizar;
- Aquisição dos diversos elementos de hardware, com vista à montagem do robô móvel;
- Análise do método a ser utilizado para dar ordens ao robô e que algoritmos específicos de processamento de imagem serão necessários implementar;
- Implementação do sistema desenvolvido no robô;
- Testes sobre o sistema implementado (*hardware* e *software*).

1.4. Estrutura

Este documento encontra-se dividido em seis capítulos.

No presente capítulo, **Capítulo 1 – Introdução**, para além desta secção, é efetuada uma introdução ao trabalho e apresentadas as motivações, objetivos e a metodologia de trabalho.

Capítulo 2 – Estado da Arte – Neste capítulo é apresentada a evolução e o estado da arte das áreas científicas envolvidas neste trabalho.

Capítulo 3 – Desenvolvimento de um Sistema Semiautónomo – São descritos os métodos propostos e os respetivos conceitos teóricos necessários ao desenvolvimento e implementação de um sistema de condução semiautónoma. É apresentado também a estrutura mecânica e a implementação do sistema de controlo proposto.

Capítulo 4 – Resultados Obtidos – Neste capítulo são apresentados e discutidos os resultados obtidos nos testes efetuados às várias etapas do algoritmo implementado e à movimentação do robô.

Capítulo 5 – Conclusões – Finalmente, neste capítulo, são apresentadas as conclusões sobre os resultados obtidos, é feito ainda um balanço das limitações da metodologia adotada e são avaliadas possíveis perspetivas de desenvolvimento futuro do trabalho realizado.

2. Estado da Arte

Este capítulo apresenta o estado da arte relativo às principais tecnologias presentes neste trabalho.

Primeiramente, é abordado o conteúdo em referência à robótica. É referido a sua origem e os tipos de robôs existentes, destacando com mais detalhe o robô móvel e as suas características.

De seguida, aborda-se o processamento de imagem e como é possível através da imagem obter informação do meio envolvente de um robô.

Por último, são destacadas as plataformas programáveis disponíveis, detalhando as mais significativas e exemplificando algumas que foram utilizadas em trabalhos anteriores, desenvolvidos no Departamento de Engenharia Eletrotécnica de Energia e Automação do Instituto Superior de Engenharia de Lisboa.

2.1. Robótica

Com o decorrer dos anos, o ser humano tem tido cada vez mais a necessidade de melhorar o desempenho e aumentar a produtividade a nível industrial. Com isto em mente, surgiu a automação e a robótica, que têm o objetivo de efetuar várias funções e aplicações sem interferência humana.

2.1.1. História da Robótica

O termo *robot*, teve origem da palavra checa *robota*, significando "trabalho". É definido, segundo o dicionário de *Webster*, como "uma máquina que se assemelha a uma criatura viva capaz de se mexer independentemente e de fazer ações complexas (como agarrar e mexer objetos)" [1]. Uma descrição mais precisa de robôs industriais vem da *Robot Institute of America*, que diz: "um robô é um manipulador reprogramável multifuncional projetado para mover materiais, ferramentas ou dispositivos especializados, através de movimentos variáveis programados para o desempenho de uma variedade de tarefas" [2]. Tendo em consideração estas duas definições, é assim necessário que o robô seja inteligente [3]. Com o intuito de dar inteligência ao robô, são utilizados algoritmos associados a sistemas de controlo e de deteção, que irão ser descritos posteriormente.

2.1.1.1. Ficção Científica

Um dos conceitos mais populares que definem os robôs, é que estes sejam semelhantes ao ser humano. Esta noção foi popularizada através da ficção científica, que teve um grande impacto no desenvolvimento da robótica. Neste tópico da ficção, um dos trabalhos mais universalmente reconhecidos é o romance *Frankenstein*, por *Mary Shelley* [4], que popularizou a ideia, diretamente traduzida para robôs, do monstro se virar contra o criador [5].

O famoso escritor *Isaac Assimov*, escreveu várias obras baseadas em robótica e é creditado como tendo inventado o termo "robótica". O conceito de robô que o escritor descreve, é o de uma máquina bem projetada e à prova de falhas, que cumpre as suas 3 leis da robótica [6]:

- **1ª Lei**: "Um robô não deve ferir um ser humano, ou através de inação permitir que um humano seja prejudicado";
- **2ª Lei**: "Um robô deve obedecer a ordens dadas por humanos, exceto quando esta entra em conflito com a primeira lei";
- **3ª Lei**: "Um robô deve proteger a sua própria existência, exceto se entrar em conflito com as duas primeiras leis".

Uma representação mais popular do conceito do robô está presente na trilogia *Star Wars*, na qual os robôs *R2D2* e *C3PO* são inteligentes, com a capacidade de se movimentarem e interagirem com os seus "mestres" humanos. Foi um desenvolvimento importante porque representam robôs como máquinas amigáveis e inofensivas [5].

2.1.1.2. Desenvolvimentos na Robótica

Entre o final da revolução industrial e o século XX, foram desenvolvidos uma série de dispositivos mecânicos no domínio da automação, que continham elementos de robótica e contribuíram substancialmente para o desenvolvimento deste domínio. Alguns destes elementos apresentam-se de seguida [5]:

- Nos meados do século XVIII, o inventor francês *Jacques de Vaucanson* construiu robôs mecânicos visando entreter o homem, conhecido como os músicos de tamanho humano (*human-sized musicians*);

- No início do século XIX, o mecânico suíço *Henri Maillardet* engenhou uma boneca mecânica (Figura 1). Com o auxílio de uma série de discos de latão, que tinham o objetivo de guiar o dispositivo num processo de escrita e desenho, esta boneca apresentava a capacidade de desenhar fotografias. A capacidade de memória permitia o armazenamento de até seis fotografias;

Figura 1: Autómato de *Maillardet – The Franklin Institute Science Museum*

- Em 1974, a *ASEA IRB* divulgou o robô manipulador, *IRb6*;
- Por fim, em 1975, uma das primeiras operações de montagens robóticas, surgiu do robô *Olivetti* "*Sigma*".

Durante o aperfeiçoamento da robótica, houve dois tipos de tecnologias que evoluíram para uso industrial. Estas tecnologias são: o braço robótico, também conhecido como manipulador industrial, e o carro móvel, designado de veículo guiado automatizado [7].

2.1.2. Robótica Industrial

Um braço robótico, ou manipulador industrial, é um dispositivo constituído por várias articulações rígidas conectadas em série por juntas. O mecanismo tem uma base de apoio, enquanto a outra extremidade está equipada com uma ferramenta que contém a capacidade de manipular objetos, executar tarefas de montagem ou outra qualquer utilidade dependente do objetivo a realizar [3].

As aplicações destes robôs consistem em operações de recolha e colocação de objetos, e o desempenho de delicadas tarefas de montagem [8]. O ponto focal da conceção dos manipuladores industriais, é que consigam executar uma tarefa repetida e eficientemente, dentro dum espaço de trabalho. Para serem multifuncionais, os manipuladores têm múltiplos graus de liberdade. Um exemplo é o braço *MOVEMASTER*, que como tem cinco juntas, tem 5 graus de liberdade (Figura 2).

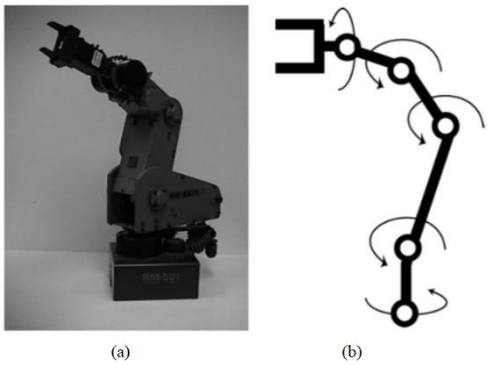


Figura 2: Um robô *MOVEMASTER*: (a) Braço robótico (b) Juntas associadas [7]

A área de trabalho de um braço robótico consiste numa zona de influência, em que este é capaz de funcionar corretamente, para que consiga aceder a qualquer ponto dentro do espaço designado. A ferramenta pode ser orientada segundo a configuração do objeto e a disposição do espaço, permitindo uma fácil e eficiente recolha. *Pitch*, *yaw* e *roll*[1] são os termos utilizados para descrever os 3 últimos movimentos associados à ferramenta. É usual dividir um robô em braço e punho, sendo que o conjunto do braço é o mecanismo de posicionamento de um robô, enquanto o subconjunto do punho é o mecanismo de orientação da ferramenta. Na Figura 3, estes conceitos são ilustrados pelo robô "*Cincinnati Milacron T*[3]" e o braço robótico "*PUMA 560 series*", o qual tem seis graus de liberdade.

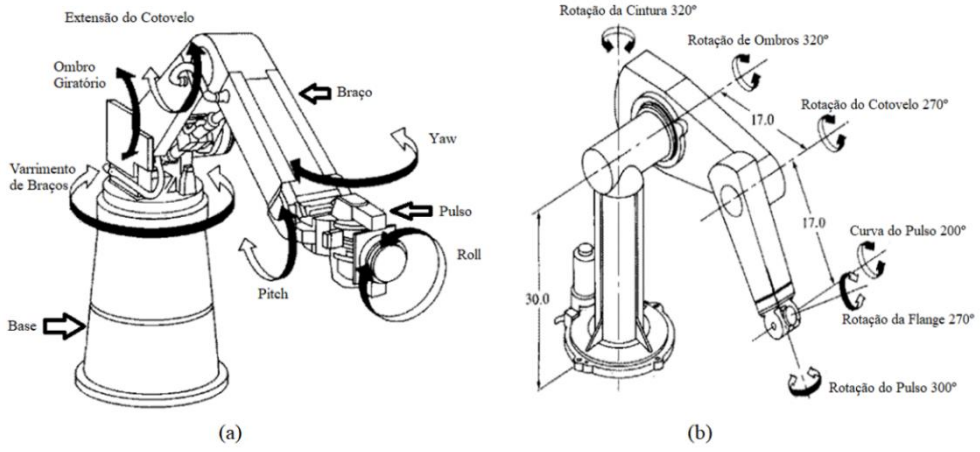


Figura 3: (a) *Cincinnati Milacron T3 robot arm*. (b) *PUMA 560 series robot arm* [3]

[1] Arfagem, guinada e volta

2.1.3. Robótica Móvel

A maioria dos animais, e todas as entidades cognitivas, requerem a capacidade de navegar com propósito e inteligência [9].

São muitos os exemplos de aplicações bem-sucedidas com os manipuladores industriais. Contudo, outros desafios surgiram que implicavam a mobilidade, tendo sido este um dos motivos para o início do desenvolvimento dos robôs móveis. Inicialmente, foi acrescentada a mobilidade de deslocar um manipulador sobre um carril, mas facilmente se percebeu que, utilizando robótica móvel, se poderia ir mais longe.

Esta área da robótica tem o objetivo de transformar um "computador" com rodas, num mecanismo inteligente e autónomo, com a capacidade de identificar propriedades e detetar padrões ou falhas. Tem também a finalidade de aprender com a experiência, ou seja, localizar-se no espaço de trabalho e construir mapas para conseguir navegar autonomamente [10]. Para alcançar tal feito, é necessária a aplicação simultânea de várias disciplinas, as quais serão mencionadas mais tarde neste documento.

O surgimento dos primeiros robôs autónomos, data o início dos anos 50, com *William Grey Walter*, que construiu alguns robôs móveis capazes de aprender tarefas (tais como o robô *Machina Speculatrix*), através do condicionamento instrumental, como evasão de obstáculos e fototaxia[2] (Figura 4) [11] [12].

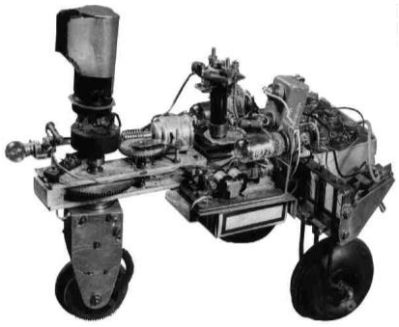


Figura 4: *Machina Speculatrix* (*Elsie*), por *William Grey Walter*, 1951

Em 1969, o cientista *Nils Nilsson* desenvolveu o robô móvel *Shakey*. Dispunha de um localizador de alcance visual, uma câmera e sensores táteis [10]. O robô conseguia completar tarefas que envolviam o planeamento, a procura de rotas, e a reorganização de

[2] "Tipo de movimento locomotor, que ocorre quando um organismo se move em direção ou para longe de um estímulo de luz." [74]

objetos simples, apesar de ter uma capacidade limitada de modelar e compreender o seu ambiente [13]. O *Shakey* é bastante importante no estudo da robótica móvel, porque para além de ter sido o primeiro a conseguir racionar sobre o ambiente à sua volta, também resultou numa série de avanços nas abordagens de inteligência artificial (IA) (Figura 5).

Figura 5: Robô *Shakey*, por *Nils Nilsson*, 1969

Na *JPL* (*Jet Propulsion Laboratory*), foi desenvolvido o *JPL Rover* nos anos 70, para exploração planetária. Utilizando uma câmera, sensores de distância a *laser* e sensores de movimento, o robô conseguia classificar o seu ambiente [10]. Este robô ainda hoje é a opção mais viável para exploração planetária à distância, um exemplo são os *Mars Rovers*, que foram utilizados em várias missões de exploração da superfície de Marte. Estes veículos são exemplificados na Figura 6.

(a) (b)

Figura 6: *Mars Rovers*: (a) *Sojourner* (1997); (b) *Perseverance* (2021)

Em 1989, foi revelado o primeiro robô com patas, *Genghis*, pelo roboticista *Rodney Brooks*, que consiste num robô com seis pernas, semelhante a um inseto (Figura 7). Este robô utiliza sensores e microprocessadores, em rede, para controlar os doze graus de liberdade nas suas seis pernas [14]. O objetivo do *Genghis* é aprender a andar para a frente. Quando ambos os sensores de movimento do robô entram em contacto com o solo,

é recebido um *feedback* negativo. Quando a roda móvel indicar que o robô está a avançar, é recebido um *feedback* positivo [15].

Figura 7: *Genghis Robot*, *MIT*, 1989

2.1.3.1. Tipos de Rodas

Um dos mecanismos mais significativos para o movimento dos robôs são as rodas. Os robôs com rodas podem viajar a grandes velocidades, consumindo o mínimo de energia possível, e podem ser manipulados utilizando apenas alguns graus de liberdade. Para além disso, o controlo do robô com rodas é menos complexo e causa menos desgaste na superfície onde se move, em comparação com outras soluções [16] [17]. O número e a posição das rodas podem ser utilizados para classificar este tipo de robôs.

Os tipos de rodas mais utilizadas são divididas em várias categorias, sendo estas, **rodas motoras**, **orientáveis** (**centradas**, **não centradas** e **esféricas**), **omnidirecionais** e ***mecanum*** [18] [19] [20] [21]:

- **Rodas motoras (convencionais)**: São empregues devido à sua simplicidade, considerando que têm apenas dois graus de liberdade (avançar ou recuar), e porque estão disponíveis em vários tamanhos e formas. Encontram-se numa grande variedade de configurações, onde são normalmente utilizadas duas destas rodas com o objetivo de conduzir o robô. Um exemplo pode ser visualizado na Figura 8. Uma desvantagem deste género de rodas consiste na dificuldade da calibração de múltiplas destas em simultâneo.

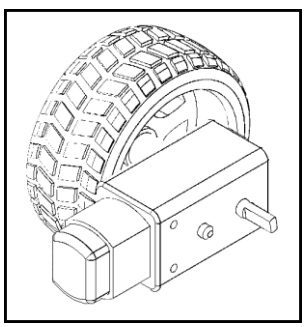


Figura 8: Exemplo de uma roda motora

- **Rodas Orientáveis ou Livres**: Têm uma estrutura mecânica diferente das rodas anteriormente mencionadas, que contém um mecanismo de direção visando permitir uma roda fixa rodar em torno do seu eixo vertical. Ao contrário das motoras, estas têm apenas o objetivo de estabilizar o robô. Existem três tipos distintos de rodas orientáveis, sendo estas:
    - **Rodas Orientáveis Centradas**: Admitem o movimento da roda em torno de um eixo vertical que atravessa o seu centro. As duas grandes vantagens destas rodas são a acessibilidade da integração em qualquer projeto e a estabilidade que proporcionam ao robô (Figura 9a);
    - **Rodas Orientáveis não Centradas (castor)**: São ligeiramente diferentes das centradas, considerando que o eixo vertical está ligeiramente descentrado e não passa pelo centro da roda. As rodas castor conseguem rodar 360 graus livremente em torno deste eixo. Também proporcionam estabilidade ao robô (Figura 9b);
    - **Rodas Esféricas**: São feitas de uma esfera metálica montada num suporte. Devido à utilização da esfera a fazer a função de uma roda, permite uma mobilidade sem restrições em todas as direções. Tal como as rodas anteriores, são frequentemente utilizadas em conjunto com outras rodas motoras (Figura 9c).

Figura 9: Rodas Orientáveis: (a) Centrada; (b) Não Centrada; (c) Esférica

- **Rodas Omnidirecionais (sueca)**: São essencialmente uma mistura de uma roda motora com rolos giratórios passivos, em que cada um tem o seu próprio eixo de rotação. Quando a roda gira no sentido horário ou anti-horário, possibilita a locomoção do robô em qualquer direção. As grandes desvantagens destas rodas é que não são muito eficientes em superfícies com irregularidades e são relativamente complexas. (Figura 10a).

- **Rodas *Mecanum***: São um tipo de roda omnidirecional com roletes montados num ângulo de 45 graus em relação ao raio[3] central. Tal como as anteriores, estas rodas permitem movimentar o robô em qualquer direção a um ritmo mais acelerado e podem ser utilizadas tanto para condução, como para direção. As desvantagens destas rodas são idênticas às omnidirecionais (Figura 10b).

Figura 10: Rodas Omnidirecionais: (a) Sueca; (b) *Mecanum*

2.1.3.2. Configurações de Rodas

Na robótica móvel não existe um sistema de disposição de rodas que consiga, em simultâneo, obter um sistema de maneabilidade, controlabilidade e estabilidade perfeitos, considerando que cada configuração de rodas tem as suas vantagens e desvantagens [22]. Tendo isto em conta, é escolhida a disposição de rodas mais prática e que melhor se enquadra na aplicação que o utilizador tem em mente.

A estabilidade estática é alcançada utilizando uma configuração com três rodas, como é verificado na configuração da Figura 11 (a projeção vertical do centro de gravidade do robô sobre o solo está dentro do polígono criado pelos pontos de contacto entre as rodas), mas a estabilidade irá ser superior aumentando o número de rodas [17].

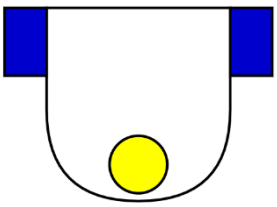


Figura 11: Configuração com 2 rodas fixas (azul) e 1 roda castor passiva (amarelo)

[3] "Cada uma das barras que unem rigidamente a zona central (ou cubo) à perimetral (aro). O centro conecta com um eixo, que pode servir para transmitir à roda uma tração motriz." [75]

Robôs Omnidirecionais:

Estes tipos de robôs podem ser projetados utilizando rodas esféricas, castores, suecas ou *mecanum* e são capazes de se movimentar em todas as direções [22]. São utilizadas três ou quatro rodas para criar uma configuração desejada. Utilizando uma plataforma triangular ou hexagonal, é possível instalar três rodas, com os seus eixos num ângulo de 120 graus, um em relação ao outro (Figura 12a). Quando são utilizadas quatro rodas, as duas configurações mais comuns são [18] [23]:

1) As rodas estão dispostas simetricamente nas laterais duma plataforma móvel quadrada; (Figura 12b);
2) As rodas são posicionadas simetricamente nos cantos de uma plataforma móvel quadrada, e os seus eixos estão inclinados em 90 graus uns em relação aos outros. (Figura 12c).

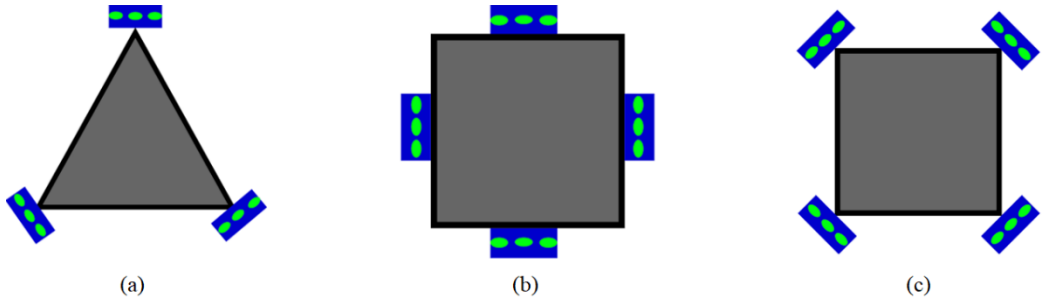


Figura 12: Rodas omnidirecionais: (a) Triangular; (b) Quadradas nas laterais; (c) Quadradas nos cantos

As desvantagens destes tipos de configurações, é que têm uma elevada sensibilidade a superfícies irregulares e uma calibração relativamente complexa.

Robôs com 3 rodas ou "Triciclo":

Como referido anteriormente, esta configuração é a mais utilizada na robótica móvel, tendo em conta que é pouco complexa, bastante eficiente e o custo é relativamente baixo, além de manter a estabilidade estática [18]. O centro de gravidade deste tipo de robô está colocado dentro do triângulo formado pelas rodas.

Este tipo de robô é composto por duas rodas motoras e uma roda livre orientável centrada, castor ou esférica, para criar estabilidade e manter o balanço do veículo (Figura 13). Se ambas as rodas motoras tiverem a mesma velocidade e a direção for para a frente, o robô irá deslocar-se nesse sentido. No mesmo caso, mas com direção para trás, o robô irá seguir o sentido indicado. Para os casos em que a direção é igual, mas as velocidades são

ligeiramente diferentes, é possível que o veículo faça curvas ligeiras. Quando a velocidade de uma é relativamente maior que a outra, o robô é capaz de fazer curvas mais apertadas.

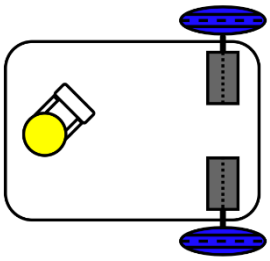


Figura 13: Configuração de um robô com 3 rodas

Robôs com 4 rodas ou "Quadriciclo":

Para este tipo de configuração existem duas soluções possíveis, 2 rodas motoras com 2 livres, ou 2 pares de rodas motoras [22]:

- **2 rodas motoras e 2 rodas livres:** Este caso é idêntico ao anterior, exceto que tem duas rodas livres para criar um maior equilíbrio. Desta vez, o centro de gravidade permanece dentro de um retângulo produzido pelas quatro rodas, criando uma alternativa mais estável do que a variante com apenas uma roda livre (Figura 14a);
- **2 pares de rodas motoras:** Esta configuração evita a utilização de rodas livres, contendo assim 4 rodas motoras, em que cada par de rodas está ligado por uma linha e giram na mesma direção. A complexidade deste sistema provém da calibração das velocidades das quatro rodas em simultâneo, considerando que se um dos pares tiver uma velocidade ligeiramente inferior ao outro, o robô não irá conseguir andar em linha reta (Figura 14b).

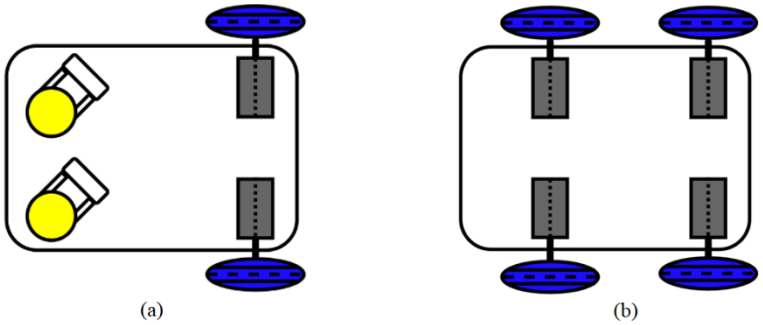


Figura 14: Robô com 4 rodas: (a) 2 motoras com 2 livres; (b) 2 pares de motoras

Robôs que dispõem de cinco ou mais rodas motoras depararam-se com o problema de calibração da configuração anterior, considerando ainda que quantas mais rodas forem

adicionadas, mais complexo será o sistema. Deste modo, este tipo de robô é utilizado apenas para projetos de maior escala ou que requerem uma estrutura maior.

2.1.3.3. Codificadores

Os codificadores, ou *encoders*, são utilizados de modo a obter a localização relativa de um robô através da sua velocidade e direção, tendo em conta a sua função de monitorizar a velocidade angular de um eixo rotativo (por exemplo, dum motor). Existem dois tipos de codificadores resultantes do tipo de sinal que o dispositivo gera, sendo estes **absolutos** ou **incrementais** [24] [25] [26] [27]:

- **Codificadores absolutos** – Fornecem uma medição direta da localização do robô e estão divididos em **óticos** e **magnéticos**. Os **óticos** são constituídos por um disco binário com n segmentos e 2^n setores, sendo que cada um destes últimos representa uma posição absoluta. Os **magnéticos** funcionam a partir da utilização de uma variedade de sensores, com o intuito de detetar o campo magnético do disco rotativo, em relação ao eixo;
- **Codificadores incrementais** – São dispositivos eletromecânicos que enviam impulsos, sempre que a localização de um eixo, de um motor rotativo, sofre alterações.

2.1.4. Sensores e Deteção

Para um robô saber onde está, como chega ao local, ou ser capaz de planear a rota, é necessário a utilização dos sensores e algoritmos de deteção.

Para a robótica móvel destacam-se dois grupos distintos de sensores, sendo estes, **visuais** ou **não visuais** [9]:

- **Sensores visuais** – Manipulam a luz refletida, proveniente de objetos no ambiente, com o intuito de conhecer o espaço e os objetos envolventes;
- **Sensores não visuais** – Utilizam áudio e outras propriedades.

Os diferentes tipos de sensores podem ser classificados de **internos** e **externos** [9]:

- **Sensores internos** – Monitorizam informação sobre as propriedades internas de um sistema robótico. Essencialmente, num exemplo de um braço robótico, controlam as juntas do robô;

- **Sensores externos** – Monitorizam características de observação do ambiente, tais como humidade, cor do objeto, entre outras. Pode-se ainda dividir os sensores externos em: **sensores de contacto**, que detetam o objeto a partir do toque, e **sensores sem contacto**, como é o caso dos sistemas baseados em câmeras.

Um sensor pode ser **ativo** ou **passivo**, dependendo da gestão da energia utilizada no processo de deteção [9] [24]:

- **Sensores ativos** – Fazem observações ao alterar o ambiente circundante ou gerando energia. Alguns exemplos são sensores *laser*, sensores ultrassónicos ou sensores de contacto;
- **Sensores passivos** – Recolhem energia de forma passiva, por exemplo uma câmera em que cada imagem obtida é composta por uma matriz que representa a intensidade luminosa.

2.1.4.1. Sensores ativos de distância ou proximidade

Estes sensores têm o objetivo de conseguir detetar um objeto, sem a necessidade de contacto direto. São utilizados para detetar a presença de obstáculos próximos do robô móvel e evitar assim uma colisão. Um sensor de medição de distância, utiliza um de dois métodos de medição, **tempo de voo** ou triangulação, no entanto, nesta dissertação irei apenas detalhar o primeiro.

Tempo de voo (ou ***Time of Flight***) – Consiste no princípio em que é emitido um sinal pelo sensor, que irá ser refletido por um objeto e de seguida retornado ao sensor. A partir do intervalo de tempo entre a emissão e o retorno do sinal, é possível calcular a distância entre o sensor e o objeto, tendo também em conta a velocidade de propagação. Esta distância é obtida pela equação 1, em que v consiste na velocidade de propagação e Δt o tempo de retorno:

$$d = \frac{v \times \Delta t}{2} \quad (1)$$

Os sensores de distância mais comuns que utilizam o método de tempo de voo são os sensores **ultrassónicos** e os sensores a ***laser***:

- **Sensores ultrassónicos** – Estes sensores funcionam a partir do método do tempo de voo para determinar a distância, sendo que o sinal enviado consiste num pulso

ultrassónico, com uma velocidade de 34300 cm/s (velocidade do som), que é refletido num objeto e retornado à origem (Figura 15a);

- **Sensores a *laser*** – Estes sensores também operam com base no mesmo método, no entanto, em vez de ser enviado um sinal ultrassónico, é utilizado um feixe de *laser* para determinar a distância entre o emissor e o objeto (Figura 15b).

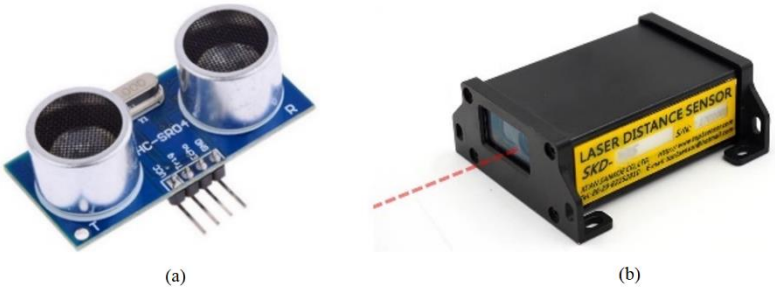


Figura 15: Sensores de distância de tempo de voo: (a) Ultrassónico; (b) *Laser*

2.1.4.2. Sensores seguidores de linha *IR* (*infrared*)

Estes sensores são importantes para aplicações em robôs com a finalidade de percorrer trajetos determinados por linhas pretas e/ou brancas. O robô consegue determinar a presença destas linhas utilizando sensores de linha infravermelhos, compostos por dois díodos, um que emite um feixe (transmissor infravermelho) e o outro que o recebe (recetor infravermelho) (Figura 16) [28].

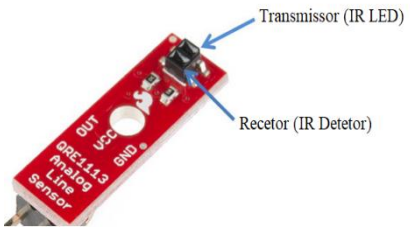


Figura 16: Sensor de linha IR típico da *Sparkfun Eletronics* (editada)

2.1.4.3. Sensores baseados em imagem

Utilizando um sensor baseado em imagem, é possível converter uma imagem num sinal eletrónico. Quando uma câmera capta uma imagem, a luz atravessa a lente e sensibiliza o sensor. O dispositivo consiste numa matriz de condensadores, que medem a quantidade de luz sobre elas [29]. O processo para a conversão num sinal eletrónico, está exemplificado no diagrama da Figura 17.

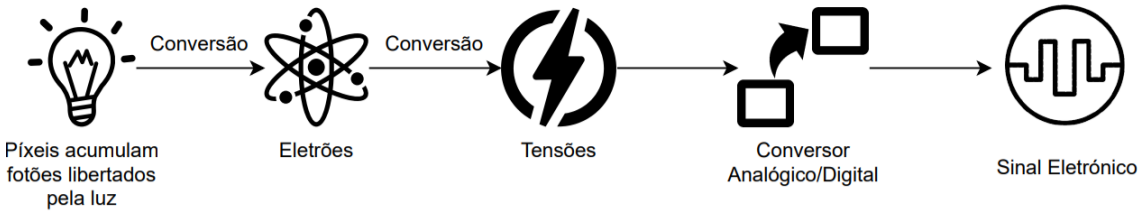


Figura 17: Diagrama do processo de conversão de luz num sinal eletrónico

De seguida, os circuitos eletrónicos presentes dentro da câmera, processam o sinal eletrónico que pode ser utilizado como informação. Presentemente, existem 2 tecnologias que podem ser utilizadas por sensores de imagem numa câmera, sendo estas **CCD (*Charge-coupled Device*)** ou **CMOS (*Complementary Metal-Oxide Semiconductor*)**. Estes sensores seguem ambos o mesmo conceito anteriormente descrito, no entanto, durante o processo de leitura as tecnologias divergem [29] [30]:

- **CCD** – O princípio de funcionamento de um sensor CCD, começa durante a exposição à luz, onde são libertos fotões, sendo estes acumulados por cada um dos píxeis[4] do sensor. Estes fotões libertam eletrões que geram cargas, as quais após a exposição são recolhidas de cada píxel. Estas cargas irão ser amplificadas e transformadas em tensão;
- **CMOS** – Cada píxel de um sensor CMOS tem MOSFETs (*metal–oxide–semiconductor field-effect transistor*) que são utilizados para amplificar e mover a carga. Como cada píxel pode ser lido independentemente, estes dispositivos oferecem a vantagem de permitir que o sensor seja mais versátil para vários tipos de aplicações.

[4] Os píxeis funcionam como condensadores.

2.1.4.4. Sensores de Orientação

Na robótica móvel, houve sempre dificuldade em identificar com precisão, a localização ou orientação do robô, sem a ajuda de dispositivos externos. Para solucionar este problema, há uma série de abordagens que podem ser utilizadas como sistemas de posição de robôs [31]. Os exemplos mais comuns destes sensores são os **giroscópios** (**mecânicos** e **óticos**) e as **bússolas**. Uma configuração de um robô com um destes sensores é exemplificada na Figura 18.

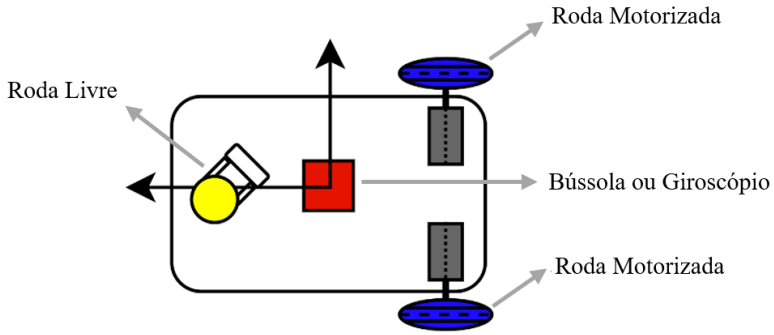


Figura 18: Configuração de um robô com 3 rodas com uma bússola ou giroscópio

Giroscópios: São dispositivos que podem ser instalados na estrutura do robô e possuem a capacidade de detetar a velocidade angular, enquanto que a estrutura gira. Como referido anteriormente, existem dois tipos de giroscópios, os **mecânicos** e os **óticos** [32]:

- **Giroscópio mecânico:** Consiste numa massa rotativa que gira em torno do seu eixo. Quando este fenómeno acontece, a massa tende a manter-se paralela a si mesma e resiste a quaisquer tentativas de modificar a sua orientação;
- **Giroscópio ótico:** Este tipo de giroscópio utiliza um *laser* para criar dois feixes de luz, que fluem em direções opostas. Os dois feixes colidem com sensores, sinalizando o momento exato em que cada um chegou ao objetivo. No instante em que um dos feixes entra em contacto com o sensor mais cedo do que o outro, significa que o objeto em questão mudou de direção.

Bússolas: Presentemente, as bússolas digitais são muito comuns na robótica móvel ou em veículos do quotidiano. São dispositivos que utilizam o campo magnético da Terra para determinar a orientação absoluta de um objeto. Quando devidamente integradas em qualquer veículo, é possível que indiquem a sua direção (Figura 19) [33].

Figura 19: Bússola magnética *CMPS03* da *Robot Eletronics*

2.1.4.5. Sistema de Posicionamento Global

O GPS (*Global Positioning System*) é o sensor de posicionamento mais amplamente utilizado. Atualmente, muitos automóveis estão equipados com sistemas GPS que lhes permitem navegar até ao seu destino desejado. Com a utilização do tempo e outros dados, o utilizador deste dispositivo pode determinar o percurso que deve seguir [34].

Esta tecnologia traduz-se diretamente para a robótica móvel, sendo possível utilizar os satélites que circundam a Terra, captando os seus sinais emitidos com o robô e processando os mesmos. Esta informação processada pode ser utilizada a fim de estimar a posição e a respetiva velocidade do robô. No entanto, este meio só é benéfico no espaço exterior, sendo que no interior não será muito útil.

2.1.5. Controlo e Navegação

Quando o ambiente onde um robô móvel circula não está estruturado, as dificuldades com a navegação autónoma tornam-se extremamente desafiantes [35]. Localização, mapeamento, uma combinação destes dois últimos designada SLAM (*Simultaneous Localization and Mapping*), planeamento de trajetórias e prevenção de colisões são algumas das questões que surgem durante a navegação num ambiente não estruturado.

Como foi anteriormente referido, a robótica móvel tem um papel importante no campo da robótica industrial, considerando as inúmeras tarefas que requerem mobilidade. No entanto, praticamente todas essas aplicações, necessitam do conhecimento prévio da sua posição atual. Assim, é necessário efetuar um cálculo da localização para estimar a posição de um robô em relação ao mapa de um ambiente, informação esta que poderá ser utilizada para decidir como o robô deve agir no seu próximo movimento [36] [37].

É possível mapear a navegação ao utilizar-se duas fontes de dados independentes, sendo estas **propriocetivas**[5] ou **exterocetivas**[6]. Os sensores destas fontes são utilizados para calcular a orientação e inclinação do robô. As **propriocetivas** utilizam giroscópios para examinar o mundo e recolher informação para o robô. As **exterocetivas** utilizam leituras de bússolas, codificadores ou outros sensores, com o objetivo de seguir os movimentos do robô no espaço ambiente [38].

Para além da informação da localização atrás descrita, poderá estar igualmente a decorrer a técnica da estimação de posição através da sua posição inicial, curso e velocidade. Esta estratégia é chamada de **navegação estimada** (*Dead reckoning*) e utiliza sensores propriocetivos para atualizar a posição. O movimento do robô é detetado através de codificadores e/ou sensores de direção que são depois combinados para calcular a sua posição [39]. Isto é, há toda uma combinação de dados e fatores que são depois tomados em consideração para o movimento de um robô.

2.1.5.1. Mapeamento

Para a situação em que um robô não tem um mapa do seu ambiente e tem de o construir à medida que está em andamento, existem dois tipos de mapeamento possíveis, um onde o robô conhece a localização do seu alvo e outro em que é desconhecido.

Conhecendo a localização do seu alvo, o robô pode navegar em direção ao seu objetivo, identificando, em simultâneo, os pontos de referência importantes no ambiente. Este reconhecimento permite ao robô desenvolver uma representação do espaço à medida que viaja até ao seu destino. Quando a localização é desconhecida, o robô deve movimentar-se através do espaço envolvente em busca da localização do alvo desejado, enquanto, simultaneamente, deteta e evita obstáculos [40].

Os sensores utilizados são determinados pelo tipo de mapeamento que é necessário realizar: ultrassónicos, câmeras digitais, e sensores *lasers* (exemplo dos "LIDAR – *light detection and ranging*") são os mais prevalentes.

Com a inovação do mapeamento na robótica móvel, surge também alguns problemas: **ruído de medição**, **alta dimensionalidade do ambiente** (quanto maior o espaço, mais complicada a criação do mapa), **problema de correspondência** (medições dos sensores

[5] Onde é que o robô está localizado no espaço?
[6] O que está a acontecer à volta do robô?

podem não corresponder à mesma entidade física no mundo real), **ambientes que mudam ao longo do tempo** e **robôs terem de escolher o próprio caminho durante o mapeamento** (a exploração para a criação do mapa pode conter surpresas indesejadas) [41].

O **planeamento de trajetória** tem a tarefa de encontrar um caminho contínuo que ligue um sistema desde um objetivo inicial a um final. Este planeamento de percurso de um robô, numa área confinada, pode ser planeado de várias maneiras. Navegação baseada em pontos de referência, planeamento reativo, e outros tipos diferentes de algoritmos de planeamento de trajetória, são alguns dos métodos disponíveis [42].

O planeamento torna-se mais complicado à medida que os graus de liberdade do sistema aumentam, ou seja, quanto maior a distância e o número de obstáculos, maior será o tempo até ser encontrada uma solução. O caminho ideal a seguir, será determinado por um conjunto de restrições e critérios, tais como o caminho mais curto entre dois locais finais ou o tempo mais curto para viajar sem colidir [43].

Para o efeito de tentar resolver o problema do planeamento surgem alguns *search algorithms*. Exemplos de alguns destes são o algoritmo ***Dijkstra*** e o ***A**** [44]:

- **Algoritmo *Dijkstra*** – Funciona a partir do cálculo do caminho mais curto desde a fonte até aos vértices, entre aqueles que estão mais próximos da origem. Encontra sempre o vértice que se encontra mais próximo, mantendo, simultaneamente, os novos numa fila de prioridade, para que apenas um caminho mais curto possa ser encontrado. Este algoritmo preocupa-se, exclusivamente, em encontrar a solução do caminho mais curto, sem atenção aos restantes fatores;
- **Algoritmo A*** – Opera semelhantemente ao algoritmo *Dijkstra*, na medida em que funciona com base no caminho mais curto, desde o ponto inicial até ao ponto final, mas orienta a sua pesquisa para os vértices mais significantes, por forma a economizar o máximo de tempo possível.

Durante a movimentação do robô, devem ser evitadas colisões entre o robô móvel e os respetivos obstáculos na sua trajetória. Para esta operação, é necessário um mapa do ambiente, uma localização desejada e a posição atual do robô, utilizando sensores ou qualquer outro sistema de localização. As colisões podem ser evitadas com a utilização

de algoritmos para evitar obstáculos. Estes algoritmos podem ser **com** ou **sem mapeamento** [22]:

- **Algoritmos com mapeamento:** Estes algoritmos baseiam-se em mapas topológicos ou representações geométricas do espaço de trabalho. Conseguem detetar colisões, realizando o cálculo das distâncias com base na sua posição atual em qualquer momento;
- **Algoritmos sem mapeamento:** Estes algoritmos não utilizam nenhuma representação do espaço de trabalho, e como tal, são utilizados sistemas de sensores para monitorizar o ambiente.

As distâncias podem ser obtidas diretamente dos sensores, permitindo que o planeador de trajetórias as contabilize enquanto deteta obstruções e evita colisões. A deteção de obstáculos é mais eficiente quando são utilizados sensores baseados em distância, como visto anteriormente [22].

2.1.5.2. Localização e Mapeamento em Simultâneo (SLAM)

Localização e mapeamento em simultâneo (SLAM) é o processo de construção de um modelo do ambiente (o mapa) e a estimativa do estado de um robô, que se movimenta, em simultâneo, no seu interior. Este estado é tipicamente caracterizado pela sua posição e orientação enquanto outras variáveis, tais como velocidade, informação dos sensores, e parâmetros de calibração podem ser de igual modo adicionadas [45]. Um dos principais problemas com o SLAM, é que as medições realizadas pelos sensores conterão sempre ruído, e a mobilidade do robô também causará imprecisões na sua posição. A utilização de uma técnica probabilística é uma forma de eliminar estas incertezas [46].

2.2. Processamento de Imagem

Com a introdução de cada vez mais tecnologias no quotidiano do ser humano, este ficou cada vez mais dependente delas e das facilidades que elas fornecem. Uma tecnologia importante que surgiu é o processamento de imagem, que visa adquirir um nível de reconhecimento semelhante ou melhorado da visão humana [47].

O processamento de imagem representa uma área intrigante da ciência cognitiva e engenharia informática. As aplicações destas áreas cresceram, significativamente, ao longo do tempo, sendo que alguns dos campos mais rapidamente emergentes são [48]:

- Deteção remota;
- Diagnóstico técnico;
- Imagem médica e respetivo diagnóstico automático ou assistido;
- **Navegação autónoma de veículos (visão robótica)**.

As técnicas de processamento de sinais são organizadas em torno de operações matemáticas (matrizes, derivadas, entre outros). A resposta da unidade de processamento pode ser uma figura ou um conjunto de parâmetros associados à imagem de entrada [49].

Na Figura 20, é demonstrado um diagrama de funcionamento do processamento de imagem, desde a aquisição, tratamento e armazenamento.

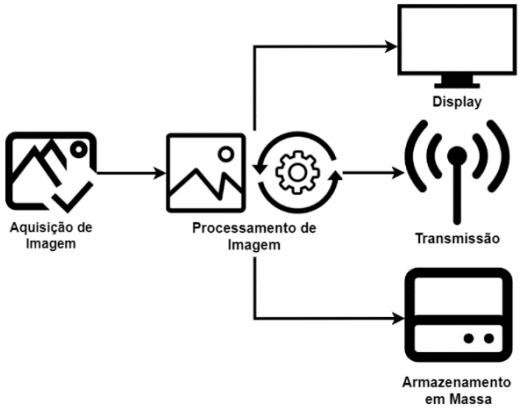


Figura 20: Diagrama de funcionamento do processamento de imagem

2.2.1. Imagem

Nesta secção são apresentados os conceitos básicos associados ao processamento de uma imagem, desde a aquisição, aos elementos fundamentais para a sua manipulação com algoritmos.

Como visualizado no diagrama da Figura 20, o primeiro passo no processamento de imagem é a aquisição. O objetivo desta aquisição é converter uma imagem ótica num conjunto de dados numéricos que podem ser, posteriormente, processados num computador [50]. Este processo depende inteiramente do sistema de *hardware* e do seu sensor (CCD ou CMOS), sendo este que irá converter a luz captada num sinal elétrico [51].

Matematicamente, a imagem é representada por uma matriz (*M* x *N*), em que cada píxel ocupa uma posição da mesma. A imagem é definida como uma função 2D, *f* (*x*, *y*), onde o *x* e o *y* representam as coordenadas e *f* consiste no valor da intensidade nesse ponto, correspondente a uma cor [52]. Cada par de coordenadas (*x*, *y*) corresponde a um ponto na imagem designado de píxel (elemento da imagem).

A gama contínua da função é dividida em **intervalos *K***, através da quantificação da imagem, que atribui um valor inteiro a cada amostra contínua. Quanto melhor for a aproximação da função da imagem obtida, melhor será a amostragem (ou seja, maior *M* e *N*) e a quantificação (maior *K*) [48]. Resumidamente, para criar uma imagem digital é necessário transformar o conteúdo contínuo, o que é realizado através da **amostragem** e **quantificação**.

- **Amostragem** – Consiste na digitalização do valor das coordenadas, ou seja, determina a resolução espacial de uma imagem digitalizada. Na sua essência, esta imagem irá ser transformada numa matriz;
- **Quantificação** – É o processo de conversão de dados da função, de uma imagem contínua para a sua contraparte digital, ou seja, é a digitalização dos valores da amplitude. Determina os níveis de tons de cor de uma imagem, sendo que estes níveis necessitam de ser suficientes para permitir a perceção humana e a identificação correta do ambiente [48]. O número de níveis de quantificação é determinado pelo número de *bits* no conversor analógico-digital, ou seja:
  - 1 *bit* – 2^1 – 2 níveis diferentes (imagem a preto e branco);
  - 4 *bits* – 2^4 – 16 níveis diferentes;
  - 8 *bits* – 2^8 – 256 níveis diferentes.

Na Figura 21, é possível observar o efeito da redução do nível de quantificação de uma imagem em tons de cinza (para melhor perceção). Começando por diminuir para 64 níveis (Figura 21a), ainda não é visível nenhuma alteração significativa. Logo após, é reduzida para 16 níveis (Figura 21b), onde já é possível visualizar degradação da informação. De seguida, é aplicada uma diminuição para 4 (Figura 21c) e, posteriormente, para 2 níveis (Figura 21d), onde a degradação já é bastante evidente. A redução do número de níveis de cor, implica uma redução do número de *bits* necessários para armazenar a imagem. A degradação mencionada, refere-se à perceção da imagem pelo ser humano. No entanto, pode ser vantajosa dependente da aplicação computacional desejada.

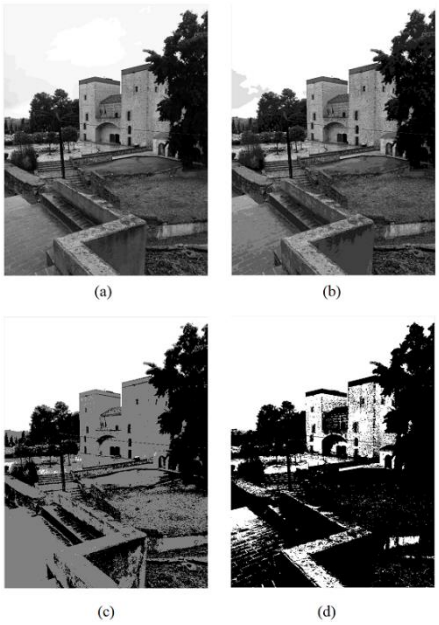


Figura 21: Níveis de quantificação de uma imagem: (a) 64; (b) 16; (c) 4; (d) 2

2.2.1.1. Mapa de Cores

A representação de uma cor está diretamente associada ao mapa de cores de uma imagem. De uma forma genérica, considera-se que existem dois tipos de imagens, a **preto e branco** e **a cores** [53]:

- **Imagens a preto e branco** – Existem apenas dois valores, o "0" para representar o preto e o "1" para representar o branco;
- **Imagens a cores** – São compostas por píxeis com várias intensidades (tonalidades). Por exemplo, numa escala de cinzentos, podem variar entre 0 e 255, sendo que o valor "0" representa o preto e o valor "255" o branco (isto para uma quantificação de 256 níveis).

Uma imagem é composta por uma gama de cores fundamentais e o seu respetivo espaço de cor, sendo este último também designado de mapa de cor. Por vezes, este mapa de cores utiliza apenas uma matriz (escala de cinzentos) e noutras situações, utiliza três matrizes (por exemplo, RGB - *Red*, *Green* e *Blue*). O principal objetivo do espaço de cor no processamento de imagem é facilitar a especificação de cores consistentemente, sem que ocorra perda de informação. Existem 3 mapas de cor mais comuns, sendo estes **RGB**,

CMYB (*Cyan*-C, *Magenta*-M, *Yellow*-Y e *Black*-B) e **HSV** (H-*Hue*[7], S-*Saturation* and V-*Value*) [48]:

- **RGB** – Este modelo corresponde a um dos métodos mais amplamente utilizados para a representação de cores na computação gráfica. Este espaço de cor representa as três cores primárias. Misturando-as em diferentes níveis, é produzida uma variedade de cores diferentes (os três canais a "0" representam a cor preta e os três canais a "255" representam a cor branca);
- **CMYB** – Este modelo, é uma técnica de mistura de cores subtrativa. Especifica as tintas que devem ser utilizadas para gerar uma tonalidade específica quando a luz é refletida de um substrato branco ("0" representa a cor primária e "1" a cor mais clara);
- **HSV** – Este esquema de cores evita o uso de cores básicas. A tonalidade tem o objetivo de especificar o tipo de cor, a saturação afeta os níveis de tons de cor da imagem e, finalmente, o valor irá definir o nível do brilho.

2.2.1.2. Histograma

O histograma de uma imagem representa a frequência relativa da ocorrência dos vários tons de cor (intensidade dos píxeis). É utilizado com a finalidade de encontrar as condições ideais de iluminação, para captar uma imagem e realizar a segmentação e compressão desta. Pode ser utilizado para melhorar a qualidade de uma imagem [48] [54]. Na Figura 22, é possível visualizar uma imagem com o respetivo histograma, indicando a frequência de cada um dos tons de cinza.

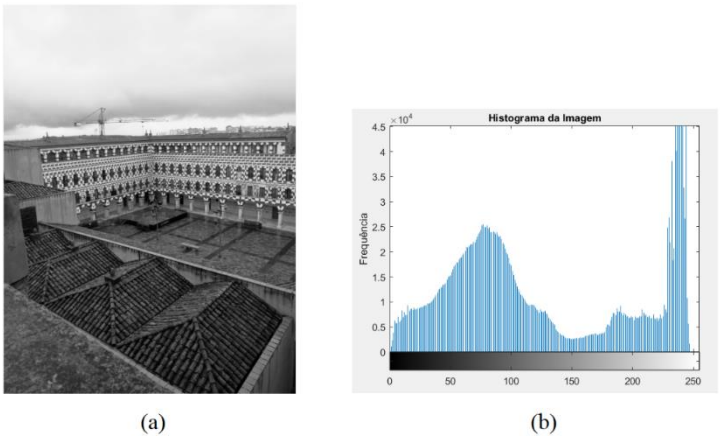


Figura 22: Histograma de uma imagem: (a) Original; (b) Histograma

[7] Tonalidade ou matiz.

2.2.2. Distâncias e Vizinhanças

Reconhecendo os conceitos de imagem e de píxel, é também importante calcular a distância entre dois píxeis, tendo em conta que esta resulta numa solução para obter o espaçamento entre as coordenadas.

Para realizar este cálculo, existe o método da **distância Euclideana** (método da linha reta). Em termos computacionais é usual usar-se simplificações, como o método ***City-Block distance*** (D_{CB}) e o método ***Chessboard distance*** (D_C) [48]:

- O método da distância Euclideana, D_E, consiste no distanciamento entre dois píxeis, utilizando a norma Euclideana, como é explicito na equação 3:

$$D_E\big((i,j),(h,k)\big) = \sqrt{(i-h)^2 + (j-k)^2} \tag{3}$$

- O método da distância D_{CB}, consiste na medição da trajetória entre os píxeis com base no módulo das coordenadas (equação 4):

$$D_{CB}\big((i,j),(h,k)\big) = |\,i-j\,| + |\,j-k\,| \tag{4}$$

- O método da distância D_C, segue o mesmo princípio que a anterior, no entanto, é utilizado o maior valor da distância das coordenadas (equação 5):

$$D_C\big((i,j),(h,k)\big) = \max\{\,|\,i-j\,| + |\,j-k\,|\,\} \tag{5}$$

Na maior parte dos casos em processamento de imagem, utiliza-se as simplificações do método da distância Euclideana, uma vez que os cálculos associados são mais rápidos.

A **adjacência entre dois píxeis** ocorre quando ambos são vizinhos e quando têm propriedades semelhantes. Para a vizinhança de dois píxeis podem ser considerados mais ou menos píxeis à volta do píxel central:

- **Vizinhança-4**, se considerar 4 vizinhos em torno do píxel central (Figura 23a);
- **Vizinhança-8**, se considerar 8 vizinhos em torno do píxel central (Figura 23b).

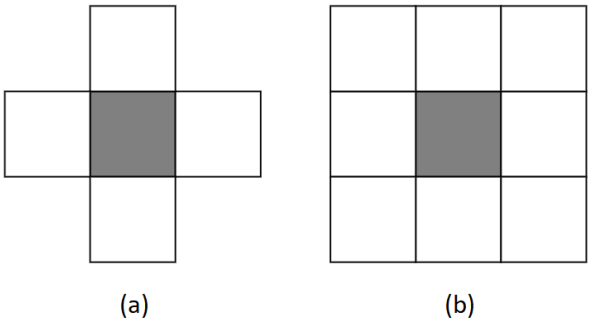


Figura 23: Vizinhanças dos píxeis: (a) Vizinhança-4 (V_4); (b) Vizinhança-8 (V_8)

Utilizar uma vizinhança de 4 ou de 8 poderá resultar em imagens processadas completamente diferentes, sendo que dependendo da aplicação será escolhido o melhor método. O objetivo de utilizar menos vizinhos é acelerar o tempo de processamento do algoritmo, uma vez que menos dados serão processados. O princípio de funcionamento consiste no facto de se o valor de tons de cinza não mudar numa pequena vizinhança, esta ficará dentro do objeto. Pelo contrário, se o valor de tons de cinza mudar consideravelmente, uma das bordas do objeto atravessa a vizinhança. Desta maneira, é possível reconhecer áreas de valores e bordas de cinza constantes [55].

2.2.3. Pré-processamento

O termo "pré-processamento" refere-se a ações executadas nas imagens ao nível do píxel, que tanto podem ter como objetivo reduzir o espaço de memória da imagem, como diminuição do ruído ou técnicas mais elaboradas. Pretende-se assim aperfeiçoar os dados da imagem, suprimindo distorções (ruído) indesejadas ou melhorando propriedades visuais. As **correções de brilho** e as **transformações em tons de cinza** são os dois tipos de pré-processamento, de ajustes de luminosidade dos píxeis [48]:

- **Correções de brilho** – Alteram a intensidade da luminosidade dos píxeis, tendo em consideração o seu brilho inicial, bem como a sua posição na imagem;
- **Transformações em tons de cinza** – Alteram o brilho de uma imagem, independentemente da sua localização.

A **binarização**[8] consiste numa técnica, em que a figura em questão será convertida em dois valores de píxeis diferentes, em que o valor "1" representa os objetos no primeiro plano (*foreground*) e o valor "0" o plano de fundo (*background*) [56]. Outro nome utilizado para esta técnica de binarização é **técnica de limiar**. Este método consiste no

[8] Transformar uma figura numa imagem binária.

processo de divisão dos píxeis das imagens em classes de objeto e fundo, com base na relação entre o valor do tom de cinza de um píxel e o limiar do parâmetro significativo, para separar o objeto do fundo [57]. Com esta técnica obtém-se uma imagem binária, em que o valor necessário para a armazenar em memória, será muito inferior ao seu original, conseguindo mesmo assim identificar o que é necessário para a aplicação.

Com a finalidade de melhorar a qualidade de uma imagem, surgem a combinação das áreas de **melhoramento de imagem**, **restauração de imagem** e **modificação da imagem geométrica** [58] [59]:

- **Aperfeiçoamento de imagem** – Inclui técnicas que aumentam a capacidade de reconhecimento de objetos numa imagem. O intuito pode ser melhorar a visão para o observador humano, bem como a conversão de uma imagem num formato mais útil para o seu processamento;
- **Restauração da Imagem** – É um método utilizado com o propósito de restaurar uma imagem degradada. São utilizadas várias técnicas de filtragem para chegar a esta finalidade;
- **Modificação da imagem geométrica** – É um processo que inclui mecanismos de ampliação, redução, rotação e distorção não-linear da imagem.

2.2.3.1. Filtragem de Imagem

As imagens são filtradas utilizando uma variedade de métodos lineares ou não lineares. A filtragem permite uma variedade de tarefas úteis de processamento de imagem, por exemplo, para minimizar a quantidade de ruído indesejável numa dada figura ou para neutralizar os efeitos de desfocagem [60]. A **suavização** é utilizada para remover o ruído e melhorar a qualidade visual da figura. Os filtros aplicados em processamento de imagem podem sem feitos através de convolução ou com transformadas de *Fourier*.

Uma **máscara** representa um filtro espacial, ou seja, resulta numa matriz que opera diretamente sobre a imagem, através da **convolução**. Esta máscara irá ser aplicada a todos os píxeis de uma imagem, a fim de obter uma nova figura. Uma das máscaras mais utilizada é a matriz 3x3, aplicada em vizinhanças-4 ou vizinhanças-8, consistindo numa transformação píxel a píxel.

Na Figura 24, é possível visualizar o processo da aplicação de um filtro, consistindo numa máscara 3x3. Consoante o valor da máscara (W_1 a W_9) obtém-se um filtro com características específicas.

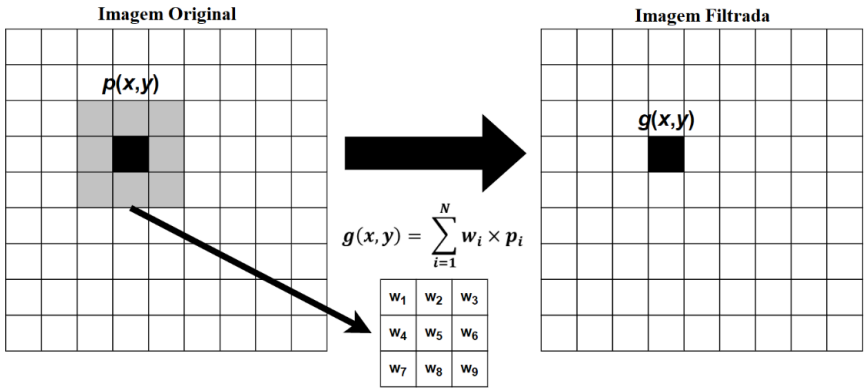


Figura 24: Processo de aplicação de um filtro 3x3

O **filtro de média (vizinhança média)** representa o método de filtragem mais simples, contendo a finalidade de efetuar uma suavização média numa figura. Consiste num filtro linear, que tem o objetivo de suprimir o ruído gaussiano. Cada píxel na respetiva imagem irá ser substituído pela média dos píxeis na sua proximidade imediata. O ruído, caso exista, irá ser integrado com o resto da imagem [60].

O **filtro da mediana** consiste numa técnica de filtragem digital (suavização) não linear, com o intuito de remover ruído, reduzir a desfocagem das bordas e, paralelamente, preservá-las. A ideia geral é substituir o píxel atual da imagem pela mediana dos píxeis na sua vizinhança [48]. Ao passo que os filtros lineares suprimem o ruído gaussiano, o mesmo não acontece para o ruído binário [55]. Na Figura 25 é possível visualizar o efeito da aplicação de um filtro de mediana para remover o ruído de uma imagem.

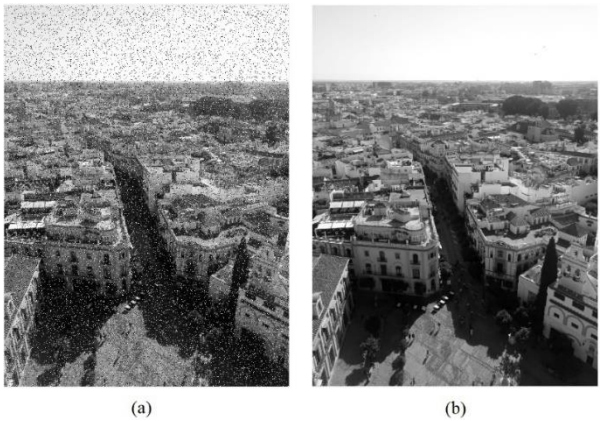


Figura 25: Remoção de ruído com um filtro de mediana: (a) Com ruído; (b) Filtrada

Consoante o tipo de aplicação, poderão ser utilizados outros filtros, como por exemplo o **filtro gaussiano**. Este filtro corresponde a um filtro passa-baixo que reduz os componentes de alta frequência, realizando uma desfocagem da imagem para que a máscara não detete ruído desnecessário e, conjuntamente, preserve os contornos. O seu funcionamento consiste na convolução da imagem com um filtro gaussiano, expresso na seguinte equação [61]:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (6)$$

O *x* e o *y* correspondem aos eixos de localização e σ ao desvio padrão da distribuição gaussiana.

2.2.3.2. Deteção de Bordas / Arestas

As bordas têm características visuais bastante importantes. Estas são sujeitas a qualidades significativas, como os aspetos geométricos e físicos dos objetos. Estes atributos do objeto são diretamente transferidos para a imagem, porque as variações relevantes ao objeto, produzem alterações na escala de cinzentos da respetiva figura [62].

A deteção de bordas consiste numa coleção de métodos de pré-processamento, utilizados para localizar diferenças nas funções de intensidade dos tons de cinza. A importância deste conjunto de técnicas, é o facto de poder identificar os limites de vários objetos, eliminando a restante informação [63] [55]. Os operadores de deteção de bordas mais comuns são o ***Roberts***, o ***Sobel*** e o ***Prewitt*** [48] [64]:

- **Operador *Roberts*:**

O operador Roberts é o mais simples e fácil de calcular, visto que apenas utiliza duas máscaras 2x2. Uma máscara consiste na outra rodada em 90º, como visualizado na Figura 26.

G_x	
1	0
0	-1

G_y	
0	-1
1	0

Figura 26: Máscaras para o operador *Roberts*

A principal desvantagem deste operador, é o facto de como apenas alguns píxeis são utilizados para estimar o gradiente, irá existir uma elevada sensibilidade ao ruído.

- **Operador *Sobel***:

O operador *Sobel* utiliza máscaras 3x3, como é evidente na Figura 27. Semelhantemente ao operador *Roberts*, a segunda máscara corresponde a uma rotação de 90º do antecedente.

G_x		
-1	0	1
-2	0	2
1	0	1

G_y		
1	2	1
0	0	0
-1	-2	-1

Figura 27: Máscaras para o operador *Sobel*

- **Operador *Prewitt***:

O operador Prewitt é bastante semelhante ao Sobel, uma vez que tira partido da máscara 3x3 em direções ortogonais. As suas máscaras são as seguintes (Figura 28):

G_x		
-1	0	1
-1	0	1
-1	0	1

G_y		
1	1	1
0	0	0
-1	-1	-1

Figura 28: Máscaras para o operador *Prewitt*

Este é um operador simples para calcular as arestas de uma imagem. Através dos seus filtros é possível obter os gradientes segundo *x* e *y*.

Independentemente do operador utilizado, a imagem resultante é uma imagem binária (B&W – *Black and White*) em que "1" representa o objeto e "0" o fundo.

Após ser aplicado o detetor de bordas à imagem, será agora possível realizar operações para melhorar ou suavizar a imagem obtida, com intenção de extrair as características dos objetos. Para este efeito podem ser utilizados **operadores morfológicos**, que consistem em operações de preservação, eliminação ou inversão dos píxeis da imagem. Alguns dos operadores mais utilizados são:

- **Erosão** – Remove os píxeis mais exteriores da região da imagem;
- **Dilatação** – Acrescenta uma camada de píxeis adicional à região;
- **Abertura** – Consiste numa erosão que é seguida por uma dilatação;
- **Fecho** – Consiste numa dilatação que é seguida por uma erosão.

Nesta dissertação irá ser aplicado um operador morfológico de dilatação. O conceito essencial desta operação é aumentar as regiões brancas na imagem, removendo simultaneamente ruído, ou seja, irá aumentar a área do objeto.

2.2.3.3. Extração de Características

Por forma a ser possível identificar e isolar um objeto do outro, é necessário aplicar a técnica da **etiquetagem**. As etiquetas são colocadas sobre a imagem de B&W obtida pelo detetor de bordas. De acordo com a vizinhança-4 ou 8, é reconhecido um objeto e atribuído uma etiqueta. Este processo ocorre para todos os elementos da imagem B&W. É assim obtida uma figura com números de 1 a *N*, em que *N* representa o número de objetos da imagem.

Após realizada a identificação dos diversos objetos, é possível extrair os atributos que se irão moldar em classificações de vários elementos. Esta classificação pode ser caracterizada pelo número de píxeis, área, região envolvente, ou muitas outras que poderão ser obtidas através do comando ***regionprops***. Esta estratégia permite ao utilizador calcular as várias características de um objeto, sendo estas: dimensões dos objetos, área, posição, orientação, entre outras.

Algumas das medidas que serão utilizadas nesta dissertação são o ***convex hull*** e o ***convexity defects***.

Convex Hull[9], ou envoltória convexa, corresponde a um limite convexo que se ajusta em torno dos pontos ou de uma forma, sendo que a *convex* representa uma forma que não contém ângulos interiores maiores do que 180º e *hull*[10] o exterior do objeto. Na Figura 29, é possível verificar a vermelho o *convex hull* de duas formas distintas [65].

[9] Envoltória convexa
[10] Envoltória ou casco.

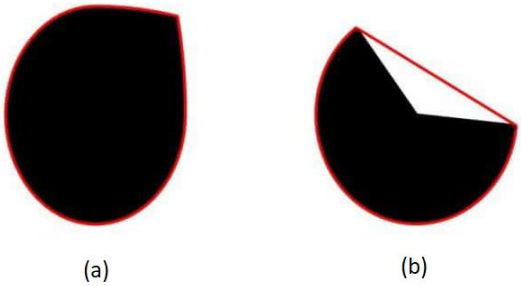


Figura 29: *Convex Hull* de duas formas (*LearnOpenCv*): (a) Convexa; (b) Côncava

De modo a que o *convex hull* consiga incluir todos os pontos na envoltória e, simultaneamente, preservar a convexidade, são gerados *convexity defects*[11], que representam qualquer tipo de desvio do limite do *convex hull* em forma de um vetor (este vetor tem um início e um fim que podem ser manipulados) [66]. Essencialmente, os *convexity defects* são a diferença entre o *convex hull* e o contorno do objeto.

2.3. Plataformas Programáveis

A fim de integrar, sequencialmente, os componentes do robô móvel com os elementos de processamento de imagem e programá-los conforme a aplicação em questão, são utilizadas plataformas programáveis (microcontroladores, microcomputadores ou computadores de placa única).

Para este efeito, uma das opções mais eficientes, acessível e abrangente em termos das funcionalidades disponíveis é o ***Raspberry Pi***. Consiste num microcomputador que corre o sistema operativo *Linux*, contendo também, sistematicamente, uma série de pinos programáveis. Essencialmente é utilizado como um computador de tamanho reduzido, permitindo ao utilizador controlar os seus diversos componentes elétricos. Este controlador foi escolhido para esta dissertação, por ser mais abrangente e mais adequado para realizar o processamento de imagem do que as outras opções que abordarei a seguir, e também devido à falta de experiência que eu apresento com esta plataforma, levando a que queira aprofundar o meu conhecimento.

Uma alternativa popular na robótica ao *Raspberry* é o ***Arduíno***, que tem a vantagem de ser menos complexo e mais direto no desenvolvimento de um robô móvel. No entanto,

[11] Defeitos de Convexidade

em termos de processamento de imagem, não é o dispositivo mais recomendado e consoante os modelos, ter-se-ia de utilizar mais que um *Arduíno*.

Para além das plataformas anteriormente mencionadas, existem ainda outras opções disponíveis. No Departamento de Engenharia Eletrotécnica de Energia e Automação, no Instituto Superior de Engenharia de Lisboa, foram já desenvolvidos uma série de projetos que envolvem uma variedade de plataformas programáveis:

- Sistema simulador de armazenamento automático, com auxílio de um ***Raspberry Pi***, por Daniel Carvalho [67];
- Interface remota genérica para braços robóticos, utilizando uma placa de ***Arduíno***, por André Mira [68];
- Robô semiautónomo "μ-Rato", que dispõe de **microcontroladores** "**16F876**" da *Microship*, para o controlo e comando do dispositivo, por António Garcia, Manuel Carvalho e Francisco da Cruz [69];
- Robô móvel de condução autónoma, que utiliza um **computador portátil** para programar os diversos elementos do robô, por César Coutinho [70].

3. Desenvolvimento de um Sistema Semiautónomo

Neste capítulo, são apresentados os métodos teóricos fundamentais para o desenvolvimento da aplicação desta dissertação e, subsequentemente, o sistema elaborado e implementado. Juntamente, são detalhados todos os componentes de *hardware* escolhidos para o robô móvel, clarificando a seleção de cada um deles. Além disso, é demonstrado o programa de processamento de imagem desenvolvido, esclarecendo o seu funcionamento.

O algoritmo de processamento de imagem foi desenvolvido com recurso à linguagem de programação *python*, empregando, simultaneamente, as bibliotecas *OpenCV e numPy*. Pretende-se que através da mão seja possível dar indicações ao robô móvel. Assim, o principal objetivo do programa de processamento de imagem, consiste em detetar o número de dedos na mão do utilizador e, consoante o número dos mesmos, realizar o controlo do robô.

Para a realização do robô móvel foi utilizada uma plataforma robótica com os respetivos componentes, sensores e câmera USB (*Universal Serial Bus*), que serão explorados no decorrer deste capítulo. Nesta dissertação é utilizado um *Raspberry Pi*, também colocado na plataforma, que é encarregue de realizar o funcionamento e controlo do sistema completo.

Na Figura 30, é possível visualizar um diagrama da arquitetura utilizada para a implementação de todos os passos anteriormente descritos, nesta dissertação.

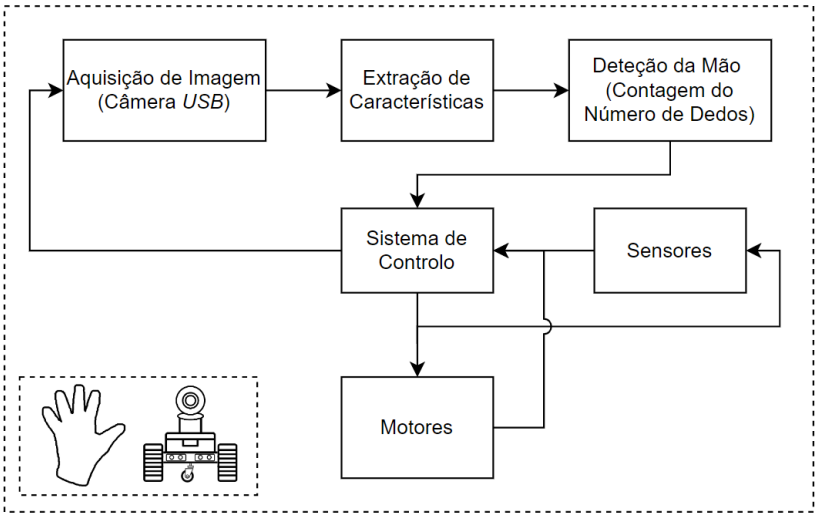


Figura 30: Diagrama da arquitetura para desenvolvimento da dissertação

O sistema começa pela aquisição de imagem e a extração das características relevantes para a contagem do número de dedos da mão. Esta informação é processada no *Raspberry Pi*, que de igual modo dispõe da informação dos sensores. Consoante a informação recolhida, será dada ordem aos motores com vista a deslocar o robô móvel. Na Figura 31 é possível visualizar o robô móvel desenvolvido nesta dissertação.

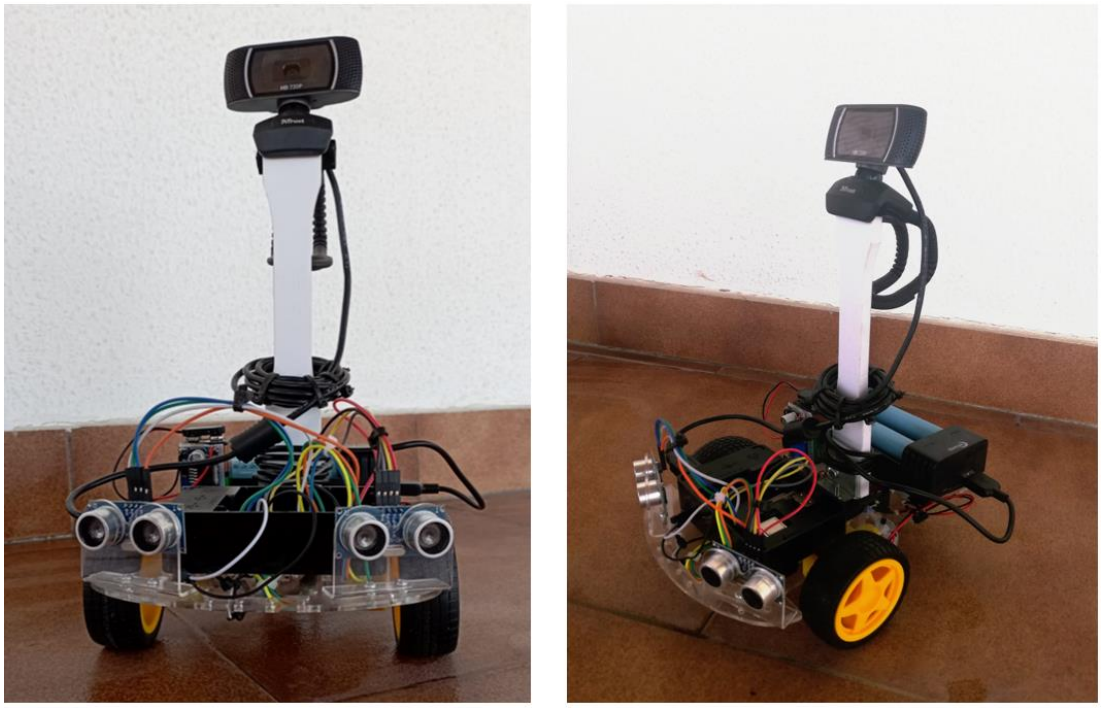


Figura 31: Robô móvel desenvolvido

3.1. Componente Robótica do Sistema

Nesta fase é detalhada a composição do robô móvel, mencionando todos os componentes utilizados e explicando o motivo da escolha de cada um. Além disso, é explicado em pormenor a metodologia dos elementos, como estes funcionam e, em alguns casos, como são alimentados no sistema de energia.

3.1.1. Composição e Estrutura do Robô

A plataforma do robô consiste num chassi feito de acrílico resistente e transparente, que possui vários orifícios para montar os diversos componentes (Figura 32). O chassi tem um comprimento de 200 mm e uma altura de 100 mm.

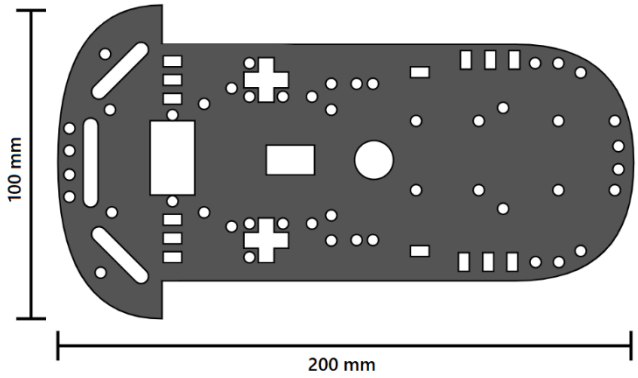


Figura 32: Estrutura do chassi do robô e respetivas dimensões

Como pode ser visualizado na Figura 33a, na parte superior do chassi são colocados dois sensores ultrassónicos, para que o robô evite colisões contra possíveis obstáculos à sua frente. Aqui também se colocam o *Raspberry Pi*, uma câmera USB com o respetivo suporte, uma pilha de 9V, um carregador com as baterias de 3.6V inseridas e duas rodas motoras.

Na Figura 33b, estão representados os componentes presentes na parte inferior do chassi, que irão ser explorados num tópico seguinte. Estes componentes consistem nas duas rodas motoras mencionadas anteriormente, os respetivos motores DC (*Direct Current*), uma ponte H e uma roda livre.

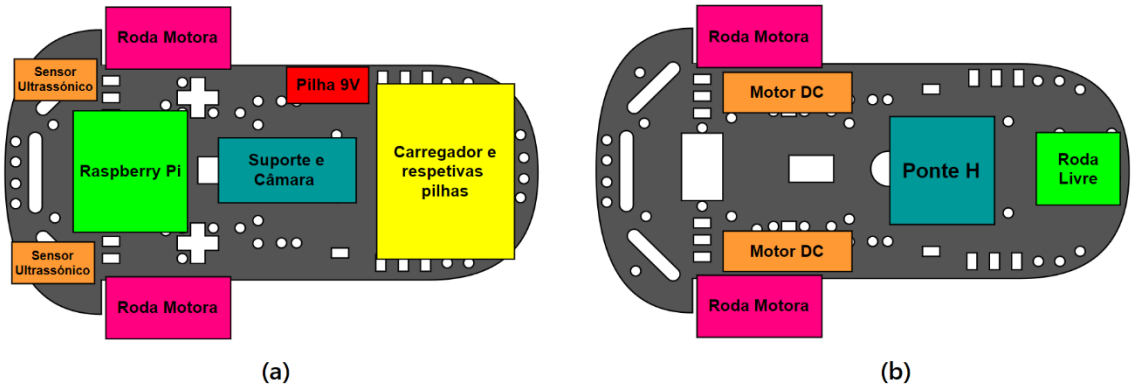


Figura 33: Componentes do chassi: (a) Parte superior; (b) Parte inferior

Com o fim de colocar a câmera do robô num meio elevado, de modo a poder ter uma zona de visão mais alargada, foi utilizado um suporte com comprimento 30 mm, altura 200 mm e largura 8 mm. A câmera é encaixada numa das extremidades do suporte, enquanto a outra é presa ao chassi através de parafusos e de um perfil galvanizado em forma de L.

3.1.1.1. Configuração das Rodas

A configuração das rodas escolhida para esta dissertação foi o triciclo (duas rodas motoras e uma livre). Optou-se por esta solução porque, considerando as configurações que fornecem estabilidade estática, esta é a menos complexa e contém um custo inferior. Comparando com a solução das quatro rodas motoras (ou quadriciclo), são ambas bastante eficientes, mas a configuração do triciclo não requer uma calibração das rodas tão complexa como a do quadriciclo. Tendo estes fatores em consideração, é a melhor opção para esta dissertação (Figura 34).

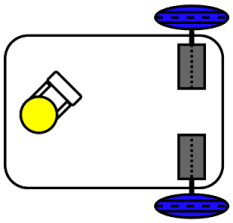


Figura 34: Configuração do robô, duas rodas motoras e uma roda livre

As rodas motoras utilizadas no robô consistem em pneus de borracha, utilizados, maioritariamente, para projetos com *Arduíno* ou *Raspberry Pi*, também conhecidas como "*smart car wheels*". Estas rodas têm 65 mm de diâmetro e 27 mm de largura. Foram escolhidas por serem as ideais para a configuração escolhida de três rodas e por serem uma opção barata (Figura 35a).

A roda livre utilizada no robô, contem uma esfera de metal e uma placa de aço grossa de 1 mm. Tem um corpo redondo de 20 mm de altura, um diâmetro de 15 mm e um peso de 37 g. Esta esfera permite ao robô movimentar-se de uma maneira flexível em todas as direções (Figura 35b).

Figura 35: Rodas do robô: (a) Rodas motoras; (b) Roda livre esférica

Inicialmente a roda livre utilizada era uma roda castor, mas durante os testes de movimento do robô, esta causava problemas quando se alterava a direção, porque ficava presa. Após alguns testes com a roda esférica, estes problemas desapareceram.

3.1.1.2. Sistema de Energia

Para realizar a alimentação do *Raspberry Pi* e dos sensores ultrassónicos, são utilizadas duas baterias ião lítio recarregáveis (Figura 36), cada uma com uma tensão nominal de 3,63 V e capacidade de 2150 *mAh*, da marca *Samsung*. Estas baterias juntamente com um carregador, irão fornecer uma tensão de 5 V (à saída do carregador), que é o ideal para a alimentação de ambos os componentes. As especificações podem ser visualizadas na Tabela 1.

Figura 36: Baterias *Li-Ion* 18650 da *Samsung*

Tabela 1: Especificações das baterias *Li-Ion* 18650

Dimensões	
Comprimento:	18,4 *mm*
Diâmetro:	65 *mm*
Peso:	44 *g*
Parâmetros	
Tensão nominal:	3,63 *V*
Capacidade nominal:	2150 *mAh*
Capacidade de carregamento:	1075 *mAh*
Corrente máxima de descarga:	10 *A*
Tensão máxima de carregamento:	4,2 V
Tensão mínima de corte:	2,75 V

Juntamente com as baterias, é utilizado um carregador da marca *Nimo* (Figura 37), que é responsável por assegurar 5 V à saída do dispositivo, de modo a alimentar os componentes anteriormente mencionados e facilitar o carregamento das baterias. As especificações deste carregador estão disponíveis na Tabela 2.

Figura 37: Carregador para 2x baterias de *Li-Ion*

Tabela 2: Especificações do carregador das baterias

Dimensões	
Comprimento:	110 *mm*
Largura:	60 *mm*
Altura:	30 *mm*
Peso:	72 *g*
Parâmetros	
Tensão de alimentação:	5 *V*
Corrente de alimentação:	2000 *mA*
Tensão de saída:	5 A
Corrente de saída:	1000 *mA*
Tipo de baterias:	Lítio

Para evitar o uso de uma breadboard e separar as alimentações dos componentes, é utilizada uma pilha recarregável de 9 V da marca *Duracell* (Figura 38). Esta pilha irá apenas alimentar a ponte L298N, responsável pelo funcionamento dos motores. As especificações desta pilha estão evidenciadas na Tabela 3. Considerando que são só necessários 5 V para a alimentação, é utilizado um *buck converter* para reduzir a tensão da pilha.

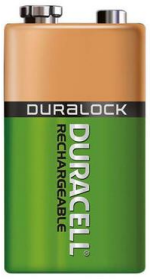


Figura 38: Pilha recarregável *Duracell* de 9V

Tabela 3: Especificações da pilha de 9 *V*

Dimensões	
Comprimento:	26,5 *mm*
Largura:	17,5 *mm*
Altura:	48 *mm*
Parâmetros	
Tensão nominal:	9 *V*
Capacidade nominal:	170 *mAh*

Como referido anteriormente, é utilizado um *buck converter* para reduzir a tensão da pilha de 9 V para 5 V. O componente escolhido consiste num *buck converter step down module* LM2596 *DC-DC* (Figura 39). As especificações deste conversor encontram-se na Tabela 4.

Figura 39: *Buck converter LM2596*

Tabela 4: Especificações do *buck converter LM2596*

Dimensões	
Comprimento:	45 *mm*
Largura:	20 *mm*
Altura:	14 *mm*
Peso:	11 *g*
Parâmetros	
Tensão de entrada:	3 – 40 *V*
Tensão de saída:	1.5 – 35 *V* (ajustável)
Corrente de saída:	2 *A* – 3 *A*
Frequência de comutação:	150 *kHz*
Temperatura de operação:	-40 – 85 *ºC*
Eficiência de conversão:	92 *%*

O *buck converter* LM2596, consiste num regulador de tensão *step-down* (*buck*), que diminui a tensão de alimentação do sistema para a sua carga, aumentado a corrente. Este componente foi desenvolvido para microcontroladores, como o *Arduíno* ou *Raspberry Pi*, e consegue conduzir uma corrente máxima de 3 A. Na Figura 40 é possível visualizar os componentes elétricos e como é constituído um *buck converter*. Na Figura 41 observa-se a pilha de 9 V com o conversor.

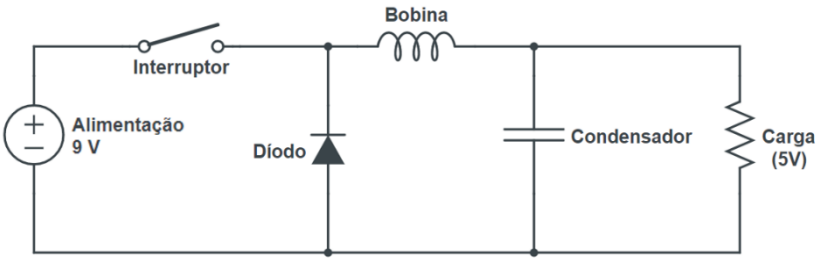


Figura 40: Diagrama do circuito do buck converter

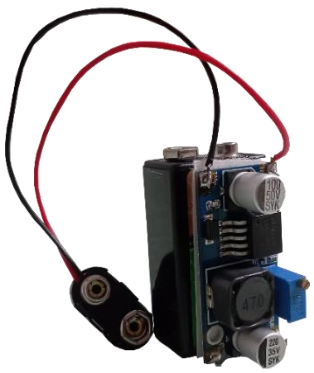


Figura 41: Pilha de 9V com *buck converter*

Na Figura 42, é possível visualizar como todos os componentes elétricos do robô são alimentados.

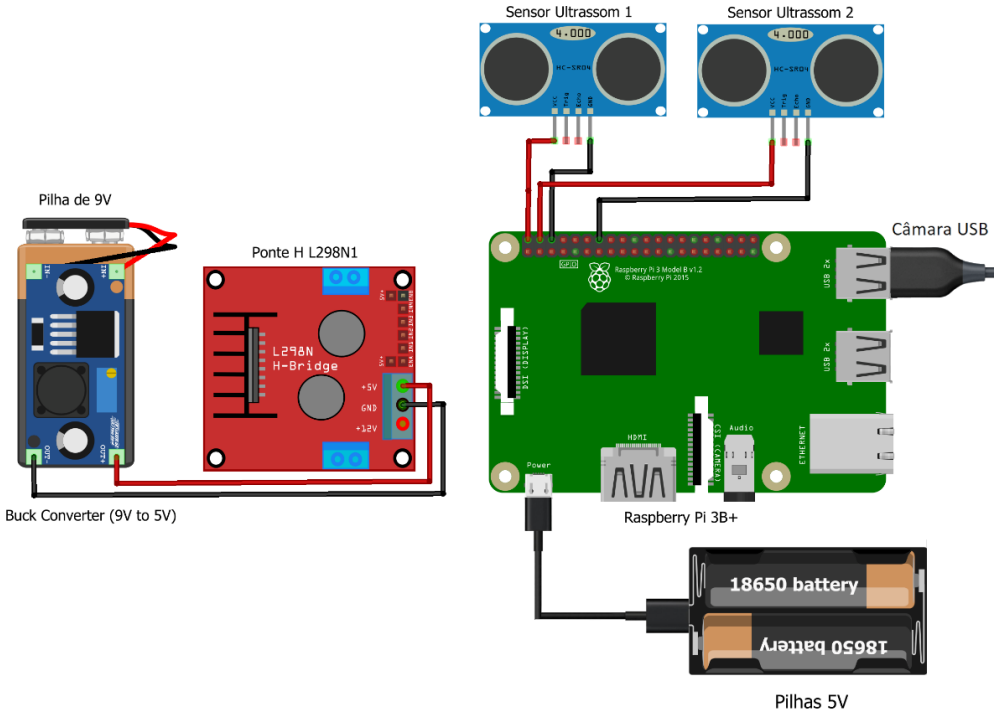


Figura 42: Alimentação de todos os componentes elétricos do robô

Foram escolhidas as baterias ião lítio recarregáveis, por terem uma excelente relação qualidade-preço e para facilitarem o carregamento, sem ser necessária qualquer desmontagem. Além disso, outra vantagem é o baixo peso do conjunto das baterias e do carregador. Com o propósito de evitar o problema de baixa tensão, comum aos *Raspberry Pi*, optou-se pela utilização de uma pilha recarregável de 9 V para a alimentação da ponte H. Numa fase inicial, talvez devido a situações incorretas de alimentação, houve um *Raspberry Pi*, disponibilizado pela orientadora, que ficou danificado. Por este motivo, o sistema dispõe de fontes separadas de energia, para sinais de potência e de controlo, com a finalidade de evitar possíveis avarias ao controlador.

3.1.1.3. Motores

Para possibilitar a locomoção do robô, são utilizados **dois motores de engrenagem DC** (Figura 43), colocados na parte de baixo do chassi. As especificações destes motores podem ser visualizadas na Tabela 5. Optou-se pela utilização deste tipo de motores por serem uma das opções mais baratas, flexíveis e suficientes que estão disponíveis para a aplicação nesta dissertação.

Figura 43: Motor de engrenagem DC

Tabela 5: Especificações do motor de engrenagem DC

Dimensões	
Comprimento:	65 *mm*
Largura:	20 *mm*
Altura:	20 *mm*
Parâmetros	
Tensão de Operação:	3 – 6 *V DC*
Corrente sem Carga:	150 – 200 *mA*
Binário (Torque):	3.50 – 8.0 *N/cm*
Velocidade sem Carga:	90 – 200 *RPM*
Caixa de Redução	
Redução:	1:48
Peso:	30 *g*

Modulação por largura de pulso, ou **PWM** (*Pulse Width Modulation*), permite ao utilizador controlar certos dispositivos (neste caso, os motores), através do controlo da energia fornecida a estes equipamentos. São, maioritariamente, utilizados para manipular a velocidade. Essencialmente, esta técnica funciona através da modulação do fator de ciclo[12] de uma onda quadrada, em que quando está ON a potência é aplicada na sua totalidade e quando está OFF não há potência na carga. Através da manipulação do fator de ciclo é possível controlar a percentagem de tempo que está ON e OFF. O cálculo deste fator está expresso na equação 8, em que L corresponde à largura de pulso e T ao período:

$$Duty\ Cycle = 100\ \times \frac{\mathrm{L}}{\mathrm{T}} \tag{8}$$

É importante salientar que a frequência se mantém igual, enquanto a largura de pulso irá variar para diferentes valores de fator de ciclo. O fator de ciclo compreende valores entre 0 e 1, por esse motivo, para esta dissertação, irá ser referido como *duty cycle*, para ser estudado em percentagem e ser mais percetível. Na Figura 44, é possível verificar o efeito de diferentes tipos de *duty cycles* numa onda quadrada de período T.

[12] Fator de ciclo corresponde à fração de tempo em que um sistema está no estado *ON* (fornecimento de energia).

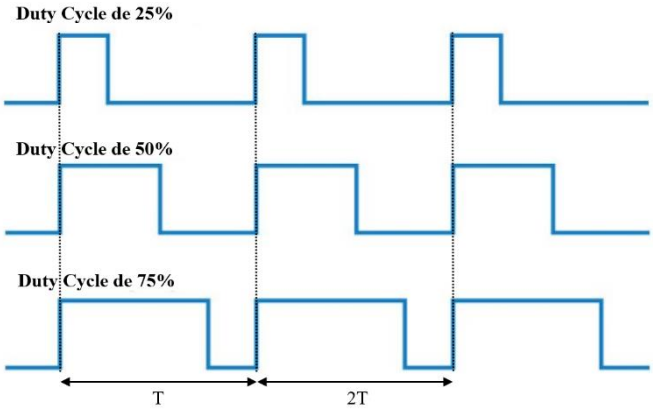


Figura 44: Exemplo das diferentes percentagens de *duty cycle*

Para controlar os dois motores DC é utilizado o **módulo ponte H L298N** (Figura 45), permitindo o controlo do sentido de rotação e da velocidade através dos pinos PWM do *Raspberry Pi*. As especificações da ponte H estão disponíveis na Tabela 6.

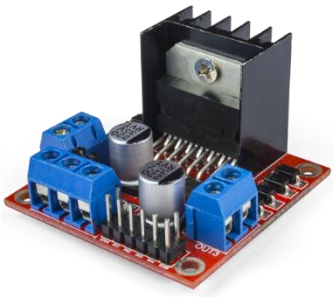


Figura 45: Ponte *H L298N*

Tabela 6: Especificações da ponte *H L298N*

Dimensões	
Comprimento:	43 *mm*
Largura:	43 *mm*
Altura:	27 *mm*
Parâmetros	
Tensão de Operação:	3 – 35 *V*
Tensão Lógica	5 *V*
Corrente de Operação Máxima:	2 – 4 *A*
Corrente Lógica:	0 – 36 *mA*
Limites de Temperatura:	-20 – +135 *ºC*
Peso	30 *g*
Potência Máxima:	25 *W*

Na Figura 46 verificam-se as ligações e pinos que abrangem uma ponte H e como é realizada a alimentação dos motores.

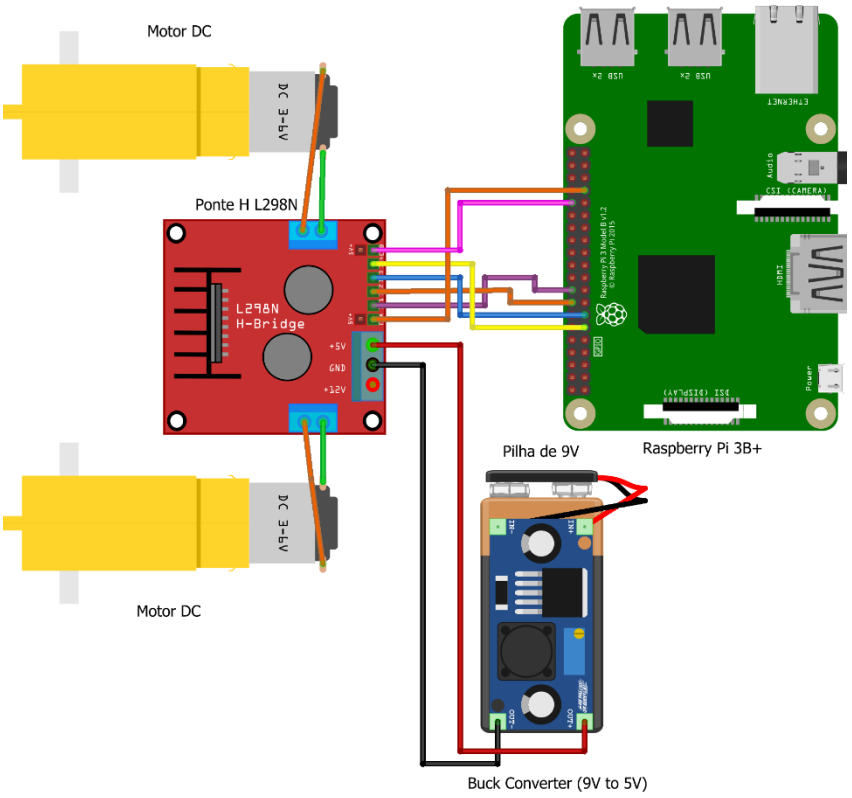


Figura 46: Diagrama do circuito da ponte H e respetivos motores no *Raspberry Pi*

Na Tabela 7, evidencia-se uma descrição mais detalhada dos componentes da imagem. O *Raspberry Pi* tem dois canais de PWM, sendo estes, PWM0 (GPIO 12 e 18) e PWM1 (GPIO 13 e 19). Para o controlo da velocidade dos motores foram utilizados os pinos GPIO 12 para o motor B e GPIO 13 para o motor A. Para o controlo da direção dos motores, utilizaram-se as entradas GPIO 23 e 24 para o motor A e GPIO 17 e 27 para o motor B.

Para os parâmetros de PWM empregues, optou-se por uma frequência de 120 *Hz* para ambos os motores, com um *duty cycle* diferente dependendo do movimento do robô. Para este tipo de motores, aumentar ou diminuir a frequência não tem um grande impacto no seu funcionamento. Por este motivo, este valor foi selecionado por ser o mais utilizado dos estudados durante o estado da arte.

Para criar uma instância PWM, é utilizada a função do *OpenCV* "gpio.PWM". Esta função emprega dois argumentos, sendo o primeiro o canal em que o pino será utilizado como saída PWM (por exemplo pino 12 para o motor A) e o segundo a frequência do canal, que como anteriormente definido, são 120 *Hz*. Logo após com auxílio da função "pwm.start", é possível inicializar o sinal PWM com um valor de *duty cycle* (por exemplo,

para andar para frente é utilizado um duty cycle de 22% no motor A e de 19% no motor B).

Tabela 7: Componentes do circuito da ponte H, com descrição detalhada

Componente	Descrição
Motor A e B	São os conectores para a ligação dos dois motores DC anteriormente mencionados.
V_{CC}	Onde é conectada a fonte de alimentação. Como a bateria utilizada contém uma tensão de 5 *V*, é conectada ao pino 5 *V*.
Ground	Massa da bateria, ligada ao pino do *GND*.
EN_A	Responsável pelo controlo *PWM* do motor A.
IN_1 e IN_2	Pinos de entrada para o motor A. Controlam a direção deste motor.
EN_B	Responsável pelo controlo *PWM* do motor B.
IN_3 e IN_4	Pinos de entrada para o motor B. Controlam a direção deste motor.

3.1.1.4. Sensores Ultrassónicos

Para ser possível haver reconhecimento do ambiente que rodeia o robô, foram instalados dois sensores ultrassónicos. Estes dispositivos irão controlar o deslocamento e a orientação, enquanto, simultaneamente, evitam possíveis obstáculos.

De modo que o robô consiga executar as suas funções sem que ocorram colisões com o ambiente, são utilizados sensores de distância. Foi criado um sistema que possibilita o desvio de obstáculos, utilizando **dois sensores ultrassónicos HC-SR04** (Figura 47), colocados na frente do chassi. As especificações deste sensor encontram-se na Tabela 8.

Figura 47: Sensor ultrassónico HC-SR04

Tabela 8: Especificações do sensor ultrassónico HC-SR04

Dimensões	
Comprimento:	45 *mm*
Largura:	20 *mm*
Altura:	15 *mm*
Parâmetros de Operação	
Tensão de Operação:	5 *V*
Corrente de Operação:	15 *mA*
Frequência de Operação:	40 *kHz*
Limites	
Máximo Alcance:	400 *cm*
Mínimo Alcance:	2 *cm*
Ângulo de Medição:	15 graus
Sinal de Disparo	
Sinal de Disparo de Entrada:	10 *µs TTL pulse*
Sinal de Disparo de Saída:	Pos. *TTL pulse*

Como pode ser visualizado na Figura 48, a operação destes sensores é realizada através de quatro ligações, que consistem na alimentação a 5 V (com uma corrente de 15 mA), a massa, o *trigger* (sinal enviado) e o *echo* (sinal retornado).

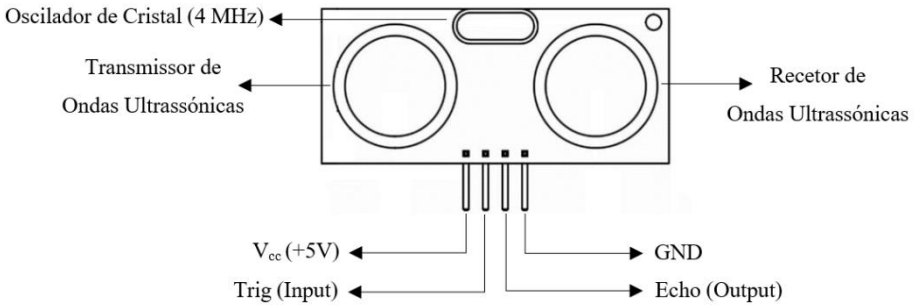


Figura 48: Ligações e composição de um sensor ultrassónico

Estes sensores utilizam o método de medição *Time of Flight*, que como referido durante o estado da arte, é emitido um sinal (pulso ultrassónico) pelo sensor, sendo, posteriormente, refletido por um objeto e retornado ao emissor (Figura 49).

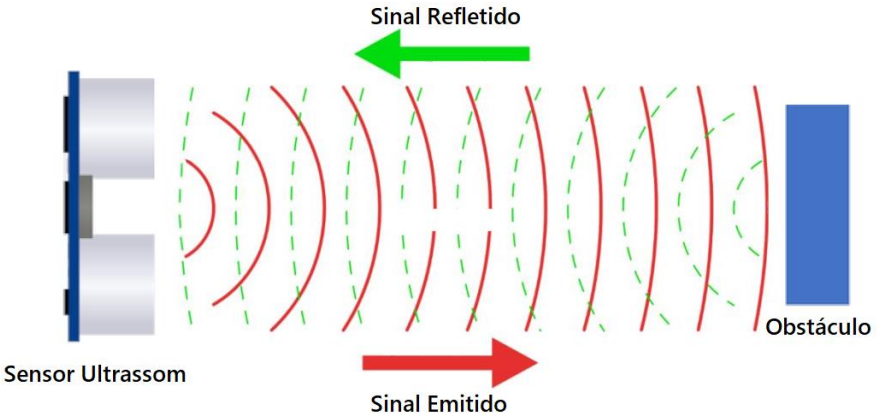


Figura 49: Princípio de funcionamento de um sensor ultrassom

Utilizando a próxima equação, é possível determinar a distância entre o sensor e o respetivo objeto, considerando o intervalo de tempo entre a emissão e o retorno do sinal, e a velocidade de propagação, que neste caso consiste na velocidade do som (34300 cm/s).

$$d = \frac{v \times \Delta t}{2} \tag{7}$$

Comparativamente a sensores de infravermelhos, os ultrassónicos são pouco afetados pelo calor ou pela pressão atmosférica e apresentam um preço de mercado relativamente baixo. [71] O facto de serem pouco complexos e de não ser necessária uma placa adicional para a sua utilização, faz com que este tipo de sensores seja o ideal para esta dissertação.

3.2. Plataforma Programável

No sentido da realização desta dissertação, foi utilizada uma plataforma programável (ou microcontrolador) denominada ***Raspberry Pi***. Um *Raspberry* consiste num computador de tamanho reduzido ("computador do tamanho de um cartão de crédito"), desenvolvido pela *Raspberry Pi Foundation*, que corre o sistema operativo *Linux*. Essencialmente, fornece ao utilizador as mesmas ferramentas que um computador normal numa escala inferior, mais barata e com menor poder de processamento, para aprender a programar, fazer automação doméstica ou desenvolver projetos de *hardware*.

Para além de ser uma ótima ferramenta para programação, também tem uma série de pinos GPIO (general purpose input/output), permitindo a manipulação de vários componentes elétricos, como por exemplo, sensores ou *leds*.

Nesta dissertação optou-se pela utilização do *Raspberry Pi model 3B+* (Figura 50), para evitar-se a compra de um novo modelo, considerando que a orientadora disponha deste. As especificações deste dispositivo encontram-se na Tabela 9.

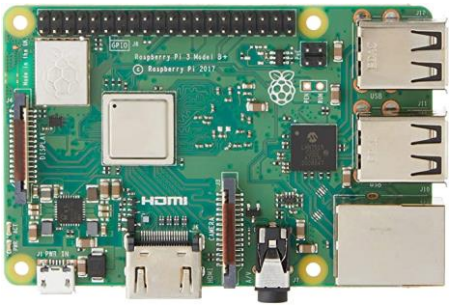


Figura 50: Plataforma programável, *Raspberry Pi Model 3B+*

Tabela 9: Especificações do *Raspberry Pi model* 3B+

Dimensões	
Comprimento:	82 *mm*
Largura:	56 *mm*
Altura:	20 *mm*
Peso:	50 *g*
Hardware	
CPU:	*Broadcom BCM 2837BO quad-core A53* (*ARMv8*) 64-*bit* (1.4 *GHz*)
GPU:	*Broadcom Videocore-IV*
RAM:	*1 GB LPDDR2 SDRAM*
Networking:	*Gigabit Ethernet, 2.4 GHz e 5 GHz 802.11b/g/n/ac Wi-Fi*
Bluetooth:	*Bluetooth 4.2, Bluetooth Low Energy (BLE)*

Armazenamento:	*MicroSD*
GPIO:	40 pinos
Ports:	*HDMI*, *audio-video jack*, 4x *USB* 2.0, *Ethernet*, *CSI*[13], *DSI*[14]

O *Raspberry Pi* é o microcontrolador ideal para o desenvolvimento desta aplicação, porque consegue, simultaneamente, controlar a locomoção do robô e realizar o processamento de imagem requerido nesta dissertação. Para além destes fatores, também permitiu-me explorar o conhecimento de outro microcontrolador e desenvolver conhecimentos em outra linguagem de programação, *python*, alargando assim os meus conhecimentos apreendidos nas unidades curriculares do mestrado. Uma opção alternativa ao *Raspberry* poderia ser, por exemplo, um *Arduíno*, no entanto, este controlador apesar de também conseguir controlar a movimentação, não conseguiria realizar o processamento de imagem, pelo que teria de utilizar o programa *MATLAB*.

Na Figura 51, é possível visualizar os 40 pinos disponíveis de GPIO. Os pinos atribuídos a cada componente elétrico irão ser explorados num tópico seguinte.

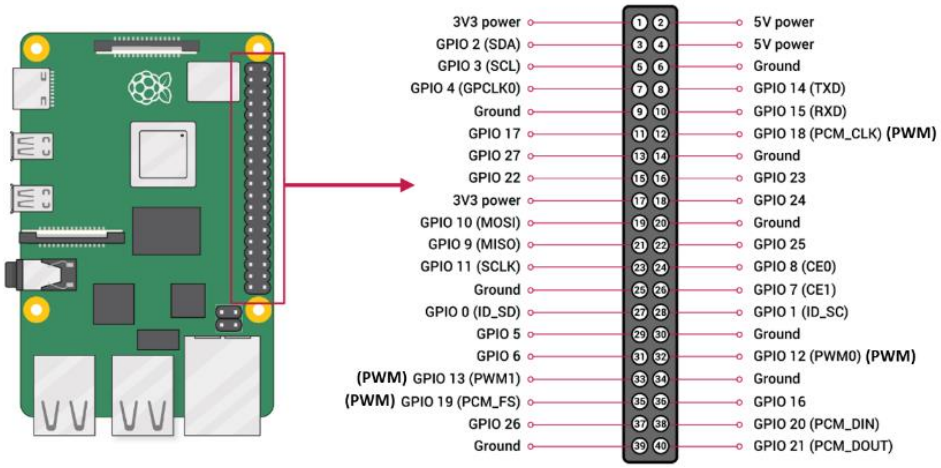


Figura 51: Pinos GPIO do *Raspberry Pi 3B+*, *Raspberry Pi Foundation*

[13] *Camera Serial Interface*.
[14] *Display Serial Interface*.

3.3. Processamento de Imagem

Com o objetivo de possibilitar a navegação do robô móvel, proposto para esta dissertação, foi concebido um algoritmo por método de processamento de imagem. Este algoritmo é o passo mais importante deste trabalho, considerando que está responsável pela navegação e automatização do robô.

Para este efeito, foi concebido um algoritmo de visão computacional, com auxílio do *Raspberry Pi*, que consiste na deteção da mão do utilizador e, subsequentemente, a contagem do número de dedos. Consoante o número detetado, o robô irá desempenhar diferentes funções.

Como referido anteriormente, o algoritmo foi desenvolvido em *python* com as bibliotecas do *OpenCV* e *NumPy*. *OpenCV* é uma biblioteca informática multiplataforma *open source*[15], utilizada, maioritariamente, para projetos e aplicações no campo de visão computacional, possuindo vários módulos de processamento de imagem. *NumPy* é uma biblioteca informática utilizada apenas para *python*, cujo objetivo é fornecer ao utilizador uma variedade de rotinas e funções para efetuar operações matemáticas, lógicas, de ordenação ou seleção e diversas aplicações com matrizes. É importante salientar que o algoritmo desenvolvido teve por bases alguns programas disponibilizados desenvolvidos em *OpenCV*[16].

O processamento de imagem está divido em várias etapas (dependendo da finalidade em questão), no entanto, para esta dissertação e, como pode ser visualizado no diagrama da Figura 52, está dividido em seis fases: aquisição, digitalização, processamento, segmentação, representação e finalmente, deteção da imagem, que neste caso será da mão do utilizador.

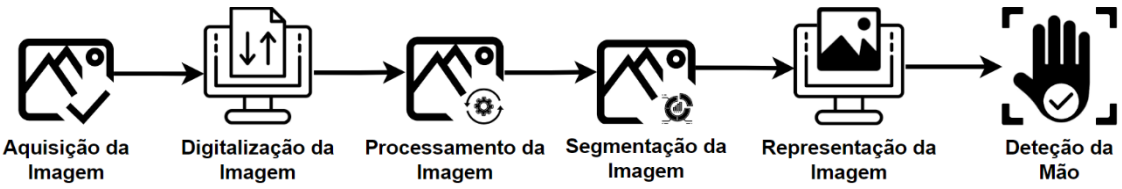


Figura 52: Diagrama de processamento de imagem para deteção de uma mão

[15] *Software* de código aberto. Qualquer utilizador pode inspecionar, modificar e melhorar o código.
[16] https://medium.com/analytics-vidhya/hand-detection-and-finger-counting-using-opencv-python-5b594704eb08, acedido a 26-05-2022

3.3.1. Aquisição e Digitalização

Como mencionado previamente, o primeiro passo a ser executado em processamento de imagem é a aquisição da mesma. Essencialmente, consiste num processo de obtenção e conversão de uma imagem num conjunto de dados numéricos que podem ser manipulados conforme a escolha do utilizador.

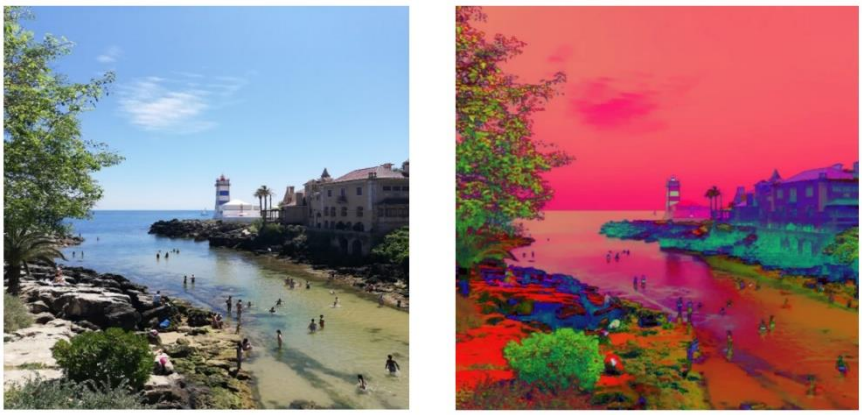
Para realizar a aquisição da imagem, foi instalada uma *webcam trust trino HD 720p* (Figura 53) no robô. Esta câmera foi escolhida por pertencer à orientadora e assim evitar-se adquirir uma diferente (apesar de ter uma boa relação qualidade-preço), mas também por ser suficiente para a aplicação desta dissertação. As especificações desta *webcam*, encontram-se na Tabela 10.

Figura 53: *Webcam Trust Trino HD 720p*

Tabela 10: Especificações da *webcam USB, Trust Trino HD 720p*

Dimensões	
Comprimento:	68 *mm*
Largura:	70 *mm*
Altura (com suporte):	81 *mm*
Profundidade:	62 *mm*
Comprimento do cabo:	143 *cm*
Peso:	88 *g*
Parâmetros e Imagem	
Resolução:	1280 x 720 px *HD*
Interface:	*USB* 2.0
Framerate máxima:	30 *FPS*
Ângulo de visão:	52 graus

Logo após a aquisição da imagem a partir da câmera USB, esta é convertida para o sistema de cores HSV (matiz, saturação e brilho). Esta conversão é essencial para realizar segmentação com base em cor. As três matrizes, "*hue*", "*saturation*" e "*value*" representam, respetivamente: cor (matiz), a quantidade que essa respetiva cor está misturada com o branco (saturação) e a quantidade que está misturada com o preto (brilho). Na Figura 54 está evidenciado um exemplo da diferença de uma imagem em RGB e HSV. Poderia ter usado o mapa de cores RGB ou HSV, mas através do HSV consegue-se uma melhor identificação das cores, tendo em vista a aplicação desta dissertação. Também se poderia trabalhar apenas com a escala de cinzentos, mas trazia limitações, principalmente em aplicações futuras.



(a) (b)

Figura 54: Imagem RGB *vs* HSV: (a) RGB; (b) HSV

O processo de conversão entre uma imagem RGB e uma HSV é um processo executado através do seguinte conjunto de equações (considerando que os parâmetros RGB contêm valores entre 0 e 255):

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

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

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

Como mencionado anteriormente, o objetivo do processamento de imagem é conseguir identificar a mão do utilizador e o número de dedos apresentados, levando a que seja necessário uma câmera USB. Para este efeito, irá ser essencial colocar a câmera numa posição alta, com o fim de possibilitar uma utilização simples, flexível e que não origine problemas de obstrução da zona a ser analisada.

Como é demonstrado na Figura 55, o suporte e a câmera têm, aproximadamente, 20 cm e 5,5 cm de altura, respetivamente, logo, a câmera, fica a 25,5 cm de altura em relação ao chassi.

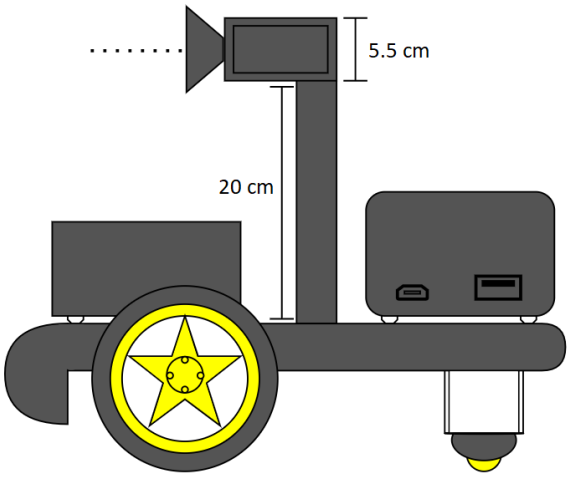


Figura 55: Posição da câmera no robô

3.3.2. Pré-Processamento

O pré-processamento de imagem, visa aperfeiçoar os dados, executando funções de filtragem para suprimir ruído, ou suavizar a imagem adquirida, antes da extração das características relevantes.

O processamento de imagem concretizado durante esta dissertação é bastante suscetível a ruído, o que pode vir a dificultar a extração das características requeridas. Tendo isto em conta, irão ser aplicados alguns filtros para reduzir ao máximo os níveis de ruído presentes na imagem e, conjuntamente, suavizar os componentes importantes. Estes filtros consistem numa operação morfológica de dilatação e num filtro gaussiano (*gaussian blur*).

O primeiro filtro a ser aplicado consiste numa operação morfológica de **dilatação**. Essencialmente a área da mão irá aumentar, com o propósito de remover possível ruído. Ruído esse que provém da iluminação do ambiente, que não é sempre constante. A matriz filtro (*kernel*) utilizada, irá ser uma matriz 2D de convolução 3x3. Foi empregue apenas uma iteração desta operação (quanto maior o número de iterações, mais dilatada fica a imagem). Na Figura 56 é evidenciado o efeito da dilatação no algoritmo de deteção da mão. Como se pode observar, o contorno dos dedos fica mais aprimorado e a parte interna da mão fica mais uniforme.

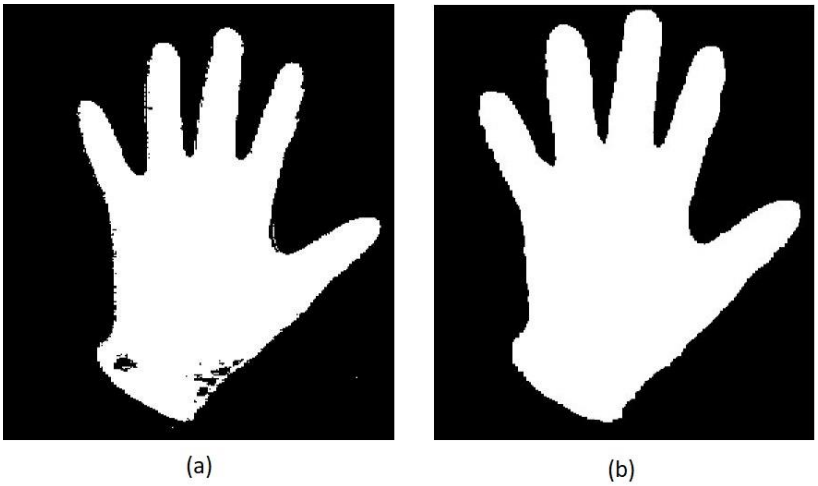


Figura 56: Efeito da dilatação na máscara: (a) Sem dilatação; (b) Com dilatação

O segundo filtro a ser aplicado é um filtro gaussiano. Tem o objetivo de realizar uma suavização à imagem sem desfocar os contornos da mão, deste modo pequenos pormenores irrelevantes não são detetados. É utilizado um filtro de convolução 5x5. Na Figura 57, está disponível um exemplo do efeito que este filtro tem numa imagem, permitindo detetar as árvores, folhagem e os animais, mas não os pormenores da folhagem ou os pormenores das penas dos animais, que seria informação em excesso. Convém realçar que em visão computacional nem sempre se deseja extrair a maior informação, mas sim a informação necessária e suficiente para a tomada de decisão.

Figura 57: Exemplo da aplicação do filtro gaussiano: (a) Original; (b) Filtrada

3.3.3. Segmentação

O principal objetivo da segmentação da imagem, é decompor a imagem em segmentos, que permitem diminuir a sua complexidade, para que se torne possível identificar os objetos desejados. Para isso, será necessário obter os contornos da imagem e atribuir etiquetas a cada objeto, de modo a ser possível extrair as suas características, que neste caso consiste na mão do utilizador. Essencialmente é importante saber distinguir a mão do resto dos objetos.

Como referido anteriormente, a imagem original foi convertida para o espaço de cores HSV. Para ser possível capturar os contornos da mão, foi necessário, inicialmente, encontrar os limites inferiores e superiores para cada um dos parâmetros de HSV.

Nesta dissertação irão ser estudados dois algoritmos para a deteção da mão, um com uma luva de nitrilo azul calçada e outro apenas com a mão. Para este efeito, é necessário encontrar os limites para a cor azul, de forma a conseguir detetar a luva, e outros limites para a cor de pele, para conseguir detetar a mão sem luva. Os parâmetros base, encontrados para ambos os algoritmos podem ser evidenciados na Tabela 11.

Tabela 11: Limites inferiores e superiores de *HSV* para a luva azul e para a mão

	Luva Azul		
	H	S	V
Limite Inferior:	90	100	50
Limite Superior:	119	255	255

	Sem Luva		
	H	S	V
Limite Inferior:	5	50	50
Limite Superior:	13	255	255

Após serem encontrados os limites corretos para os dois algoritmos, será agora possível determinar os contornos da mão. Os contornos consistem numa curva que une todos os pontos contínuos que contêm a mesma cor ou intensidade na fronteira da máscara utilizada. São utilizados, maioritariamente, para análise de formas e reconhecimento de objetos.

Para encontrar os contornos da máscara criada anteriormente e efetuar a deteção do número dos dedos, é utilizada a função "*findContours*" do *OpenCV*, que irá obter todos os contornos da imagem em questão. O resultado desta operação pode ser visualizado na Figura 58.

Após serem definidos todos os contornos, é necessário retirar os que contêm a maior área, porque é possível assumir que o contorno da mão será o maior representado. Deste modo, elimina-se qualquer situação de ruído, devido à iluminação e aos limiares, para identificar a luva azul ou a mão. O próximo passo consiste em aproximar e suavizar os contornos. Para este efeito são utilizadas as funções "*arcLength*" (calcula um perímetro de contorno)

e "*approxPolyDp*" (aproxima um polígono com outro com menos vértices para que a distância entre eles seja menor ou igual à precisão especificada).

Figura 58: Contornos da mão utilizando a função "*findContours*"

Encontrado os contornos da mão, o próximo passo será definir o *convex hull* e encontrar os *convexity defects* para a deteção do número de dedos. *Convex hull*, corresponde a um limite convexo que se ajusta em torno dos pontos ou de uma forma. Para calcular o *convex hull* de uma mão humana, a palma e os dedos irão compreender a área envolvente, enquanto os espaçamentos entre estes últimos irão ser os *convexity defects*. É assim desenhado à volta da mão, de forma que todos os pontos do contorno da mão estejam dentro do *hull*, como pode ser visualizado na Figura 59.

Para obter os espaçamentos entre os dedos é necessário utilizar os *convexity defects*. Na Figura 59, a vermelho está representado o *convex hull*, a cinzento o contorno da mão e as setas pretas correspondem ao desvio dos contornos dos limites da envoltória.

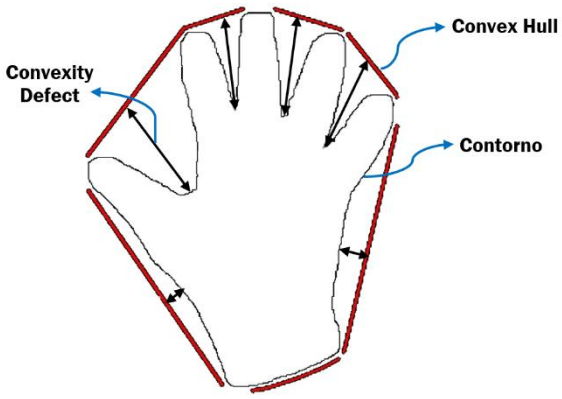


Figura 59: *Convexity Defects* de uma mão humana

A representação de imagem é fundamental para uma boa deteção e reconhecimento de objetos. Durante a segmentação, as regiões ou características importantes são etiquetadas, para durante o passo atual, poderem ser representadas ou manipuladas, conforme a decisão do utilizador. Para esta dissertação, estas características correspondem à contagem do número de dedos da mão.

O método utilizado para a contagem dos dedos, é através dos *convexity defects* anteriormente definidos. Para além dos *convexity defects* correspondentes aos espaçamentos entre os dedos, dependente da posição e orientação do pulso, existem também outros, como é evidenciado na Figura 59. Para que o código consiga identificar apenas os espaçamentos entre os dedos como *convexity defects*, será utilizada uma técnica através do ângulo envolvente. Os *convexity defects*[17] retornam uma matriz onde cada linha irá conter os valores do **ponto de partida**, **ponto final**, **ponto mais distante** e **distância aproximada ao ponto mais longínquo**. A partir destes valores é possível definir os lados que os espaçamentos entre os dedos fazem com o *convex hull*, como se verifica na Figura 60.

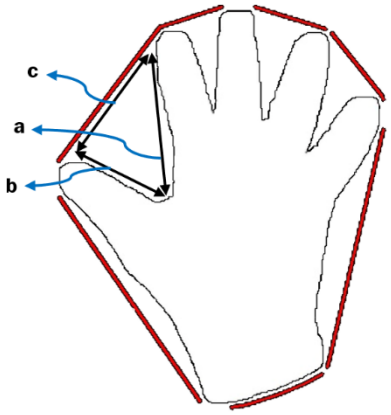


Figura 60: Lados do triângulo entre o espaçamento entre dois dedos e o *convex hull*

Logo após, é calculada a área do triângulo resultante, através das seguintes equações:

$$s = \frac{a + b + c}{2} \tag{8}$$

$$\text{area} = \sqrt{s \times (s - a) \times (s - b) \times (s - c)} \tag{9}$$

[17] https://docs.opencv.org/3.4/d5/d45/tutorial_py_contours_more_functions.html, acedido a 13-02-2022

Seguidamente, é calculada a distância entre um ponto do triângulo e o *convex hull* com auxílio da equação 10. Esta distância também será útil para remover pontos próximos da linha envolvente que normalmente contêm muito ruído.

$$dist = \frac{2 \times area}{a} \tag{10}$$

Finalmente, é aplicada a regra do cosseno na equação 11 para se obter o ângulo em questão. Esta lei relaciona os comprimentos dos lados de um triângulo com o cosseno de um dos seus ângulos.

$$angulo = \text{acos}\left(\frac{b^2 + c^2 - a^2}{2 \times b \times c}\right) \tag{11}$$

Ignorando ângulos maiores que 90º e pontos muito próximos da envoltória, é possível agora marcar os espaçamentos entre os dedos e realizar a contagem conforme o número de espaçamentos detetado na mão, como é verificado na Figura 61.

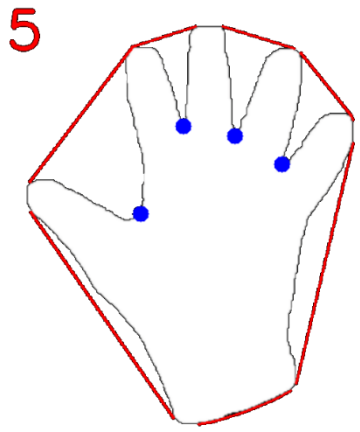


Figura 61: Contagem do número de dedos através dos *convexity defects*

Estes passos anteriormente mencionados e explicados, podem ser evidenciados no diagrama da Figura 62, que representa o processo deste programa para detetar o número de dedos da mão.

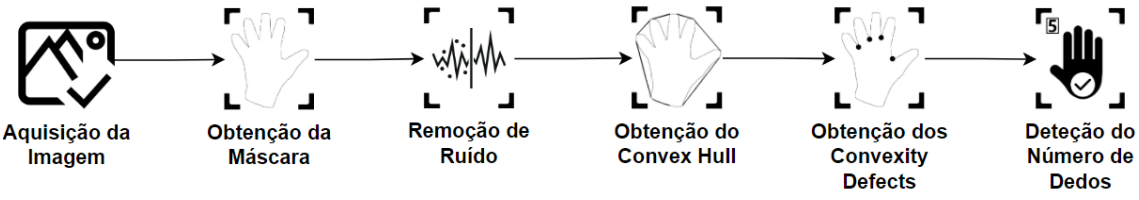


Figura 62: Diagrama do programa para deteção do número de dedos de uma mão

3.4. Deteção de Obstáculos

Para evitar possíveis colisões no percurso do robô móvel, foi desenvolvido um algoritmo de deteção de obstáculos, com recurso aos sensores ultrassónicos colocados na parte da frente do chassi.

Estes sensores possuem um transmissor ultrassónico (o pino *trig*) que emite um pulso de alta frequência com o objetivo de propagar-se pelo ar até encontrar um objeto e, subsequentemente, refleti-lo de volta para ser captado pelo recetor ultrassónico (o pino *echo*).

Inicialmente, é aplicado um pulso de 1 nanossegundo para acionar os sensores. Seguidamente, é necessário verificar se o pino *echo* foi ativado (significando que foi detetado um objeto). Para este efeito são utilizados dois ciclos, o primeiro irá fixar o tempo atual em que o pino está desligado e o segundo o tempo em que o pino está ligado. Recorrendo a estes dois ciclos, é possível determinar a duração que o pulso ultrassónico demorou a ser enviado e recebido de volta, efetuando uma subtração dos dois tempos. Considerando que a velocidade do som no ar é aproximadamente 34300 cm/s, é possível determinar a distância entre o sensor e o obstáculo, multiplicando esta velocidade pela duração anteriormente mencionada e dividindo por 2 (esta divisão é necessária porque a duração corresponde ao tempo em que o sinal foi enviado e recebido de volta, logo só é necessária metade da velocidade).

Após alguns testes desta rotina, foram detetados casos em que o programa ficava preso nos ciclos. Para resolver este problema foi adicionado um *timeout*, que após um determinado tempo irá forçar a saída do ciclo[18].

3.5. Algoritmo de Controlo

Este algoritmo é responsável pela seleção das aplicações disponíveis que irão controlar a movimentação do robô móvel, como também pela utilização em simultâneo de todos os componentes presentes nesta dissertação. A parte fundamental deste algoritmo, é utilizar a deteção da mão e dedos do utilizador como um comando, e dependendo do número de

[18] https://raspberrypi.stackexchange.com/questions/95332/ultrasonic-sensor-stops-working-a-few-minutes-later, acedido a 27-05-2022

dedos apresentados, efetuar diferentes operações, como pode ser visualizado na ilustração da Figura 63.

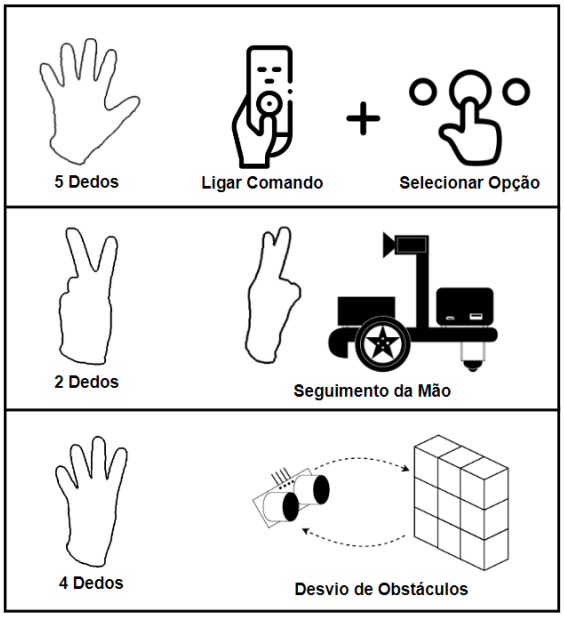


Figura 63: Ilustração das diferentes opções de controlo disponíveis

Inicialmente, após o algoritmo ser executado, o comando encontra-se desligado e à espera de ser ativado. Para ligar o comando, o utilizador necessita de utilizar 5 dedos.

Após o comando ser ligado, o utilizador tem duas opções, movimentação por **seguimento da mão** ou movimentação com **desvio de obstáculos**.

Para evitar uma má deteção do número de dedos, a fim de que cada uma das opções disponíveis seja executada, é necessário manter a mão com o número de dedos desejado em frente da câmera durante 1 segundo, para confirmar que foi a seleção desejada. Após escolhida a opção, para voltar ao modo comando, é necessário utilizar novamente 5 dedos.

3.5.1. Movimentação por Seguimento da Mão (Método A)

Nesta solução, o robô irá movimentar-se seguindo a mão do utilizador. É acedida quando o utilizador utiliza 2 dedos após a ligação do comando. O funcionamento desta operação é relativamente simples, considerando que irá seguir os movimentos da mão enquanto esta estiver à frente da câmera. Uma vez que a janela da câmera tem um comprimento de 637 píxeis, irá ser dividida em 3 secções:

- **Intervalo de $0 \leq x \leq 159$** – Quando a mão se encontra entre estes limites, significa que a mão do utilizador está à esquerda do centro da janela, logo o robô irá deslocar-se lentamente para a direita para compensar o movimento e recentrar a mão;
- **Intervalo de $159 < x < 478$** – Na ocasião em que está neste intervalo, a mão está no centro da janela, logo o robô irá simplesmente movimentar-se para a frente;
- **Intervalo de $478 \leq x \leq 637$** – Neste instante, irá ter um comportamento semelhante ao primeiro intervalo, mas para o lado oposto. A mão está à direita do centro da janela, por esse motivo, o robô irá movimentar-se para a esquerda.

Quando o robô móvel se movimenta para muito perto da mão do utilizador, este irá andar para trás até estar a uma determinada distância da mão. Esta distância é definida através da área da mão e o movimento ocorre quando esta for superior ao limite predefinido de 80000 píxeis. Para não haver leituras incorretas, todos os contornos detetados com uma área inferior a 10000 píxeis são descartados.

Também foi implementado um simples desvio de obstáculos. Quando os sensores ultrassónicos detetarem um objeto, este irá simplesmente andar para trás até estar a uma distância segura do objeto. Não foi implementado um contorno do obstáculo, porque este modo consiste no seguimento da mão, o que possibilita que haja um desvio realizado pelo utilizador na presença de um objeto no trajeto.

Na situação em que o robô não deteta a mão do utilizador passado 3 segundos (a partir da área), este irá rodar sobre si mesmo infinitamente até encontrá-la. Logo após, irá voltar ao estado normal e seguir a mão. Se durante a rotação, os sensores encontrarem um obstáculo, irá andar para trás, para uma posição em que não haja deteção de objetos e, logo após, continua a rodar.

Estas situações podem ser evidenciadas no fluxograma da Figura 64.

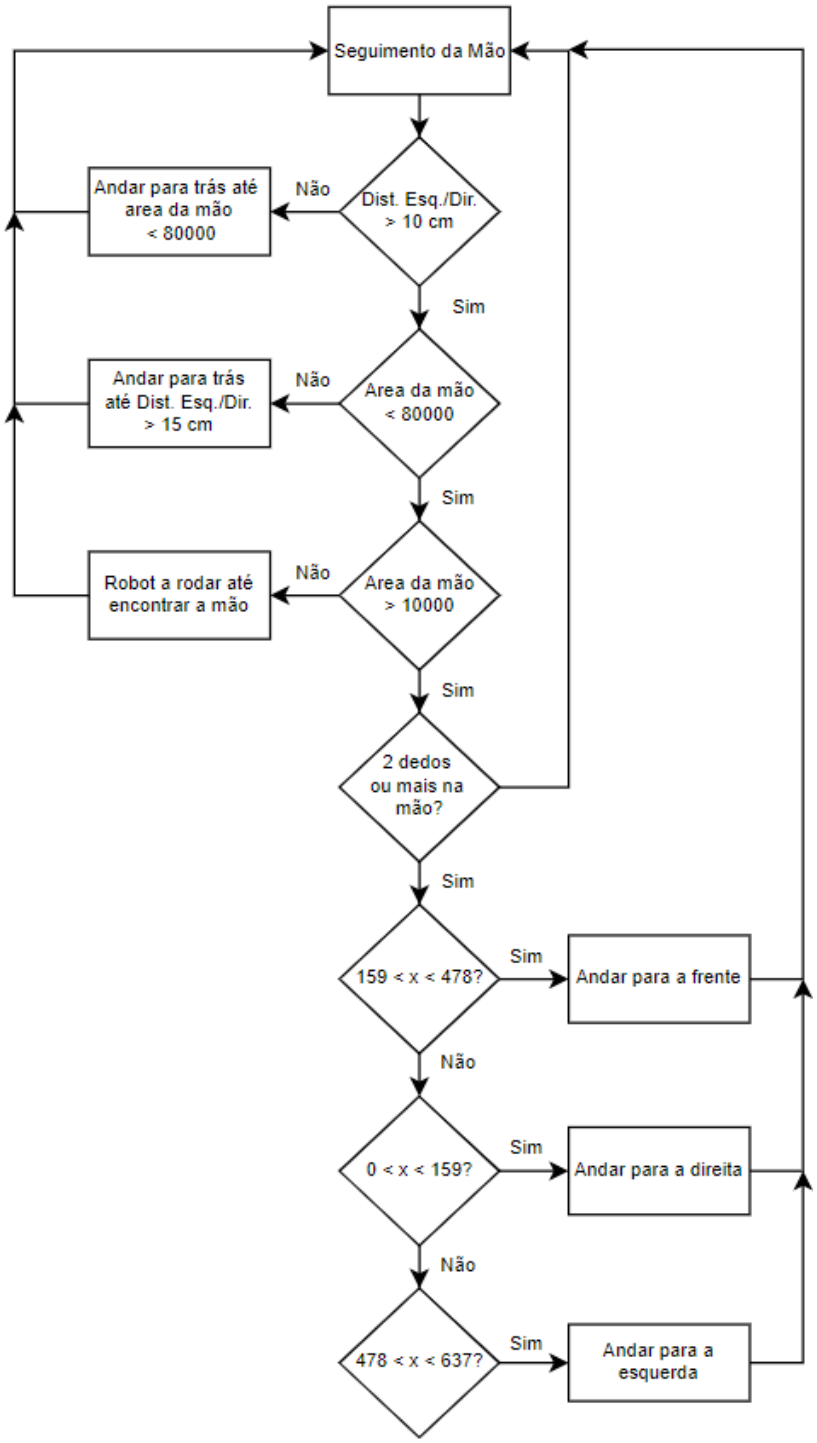


Figura 64: Fluxograma do funcionamento do algoritmo de seguimento da mão

3.5.2. Movimentação por Desvio de Obstáculos (Método B)

Esta solução resulta na movimentação do robô móvel para a frente até encontrar um obstáculo e desviar-se dele. É acedida quando o utilizador utiliza 4 dedos após a ligação do comando.

Inicialmente é executada a rotina de deteção de obstáculos anteriormente descrita, para ser possível os dois sensores ultrassónicos começarem a medir as distâncias até aos objetos. Em seguida, o robô irá andar para a frente até que a distância de um dos sensores seja inferior a 10 cm. Inicialmente o robô móvel irá desligar os motores para, posteriormente, andar para trás até a distância ser maior que 15 cm. Logo após, dependo de qual das distâncias dos sensores for maior, pode acontecer uma das seguintes ações:

- **Distância do sensor da direita maior do que o da esquerda**, o robô irá fazer uma curva para a direita. No caso em que umas das distâncias dos sensores for inferior a 8 cm ou superior a 25 cm, irá repetir o processo anteriormente descrito;
- **Distância do sensor da esquerda maior do que o da direita**, o robô irá executar as mesmas ações da opção anterior, mas irá realizar uma curva para a esquerda.

Após ter contornado o obstáculo, irá continuar a andar para a frente até que encontre um novo obstáculo e seja preciso repetir o processo.

Na situação em que o robô móvel deteta a mão do utilizador (a partir da área), este irá parar. Esta condição tem o objetivo de suspender o movimento, para que seja possível o utilizador ligar o comando novamente.

As situações abordadas podem ser evidenciadas no fluxograma da Figura 65.

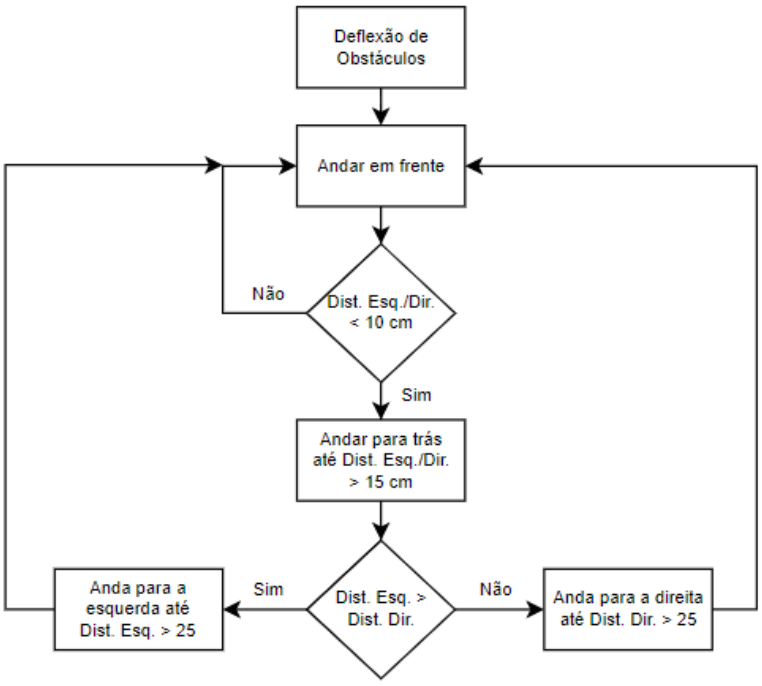


Figura 65: Fluxograma do funcionamento do algoritmo de desvio de obstáculos

4. Resultados Obtidos

Neste capítulo, irão ser apresentados os valores obtidos com o sistema implementado. Primeiramente, irão ser demonstrados os valores de calibração do sistema, ou seja, os valores limiares para a obtenção dos contornos da mão. Posteriormente, serão exibidos os resultados obtidos no processo de obtenção do número de dedos.

De seguida, irá ser realizada uma comparação dos métodos da luva azul e mão sem luva, começando por exibir algumas situações em ambientes diferentes e o comportamento de cada técnica, para logo após determinar qual é o mais indicado para esta dissertação.

Além disso, será estudado a eficácia do sistema de desvio de obstáculos e das diferentes velocidades que o robô móvel pode assumir (pretendendo decidir qual a ideal para cada situação).

Por último, irá ser comparado o melhor método de deteção da mão elaborado, com o algoritmo comercial do mediapipe e apresentado as limitações disponíveis do sistema elaborado.

4.1. Importância dos valores de HSV

Nesta fase são explicados os valores utilizados para a segmentação da mão do utilizador com o algoritmo da luva azul e o sem luva, como anotado na Tabela 11. Este estudo foi realizado numa situação perfeita para ambos os algoritmos, em que o fundo consiste numa parede branca num ambiente bem iluminado, ou seja, num ambiente controlado de iluminação. Na Figura 66 é possível observar os resultados obtidos para a luva azul e na Figura 67 para a mão sem luva.

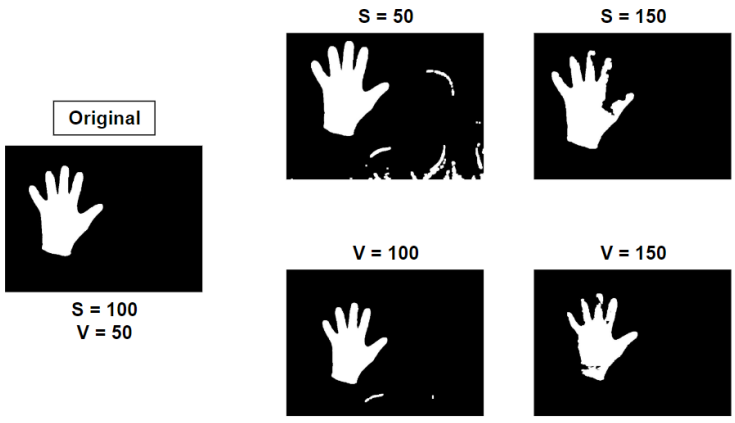


Figura 66: Valores de saturação e brilho na segmentação da luva azul

Analisando a figura acima, é possível observar que utilizando uma saturação inferior à escolhida, ficaria presente algum ruído, o que poderia dificultar a deteção da mão. Isto é, haveria objetos do fundo que poderiam ser considerados como dedos de uma mão. Para uma saturação superior, contempla-se que a luva começa a desaparecer. Quando é introduzido um valor de brilho maior que o original, verifica-se que não tem um impacto muito grande. Este fenómeno acontece porque a imagem estudada, é um caso ideal e numa situação de pouca iluminação, o valor do brilho tem um maior efeito.

A utilização de uma luva, é a situação em que se tem mais controlo da mão, uma vez que a cor da luva seria sempre igual. O mesmo já não acontece quando se deteta uma mão sem luva, em que existem várias tonalidades de cor de pele. De reparar que, de igual modo, opta-se sempre por mostrar a palma da mão para a câmera, por ser o gesto mais natural.

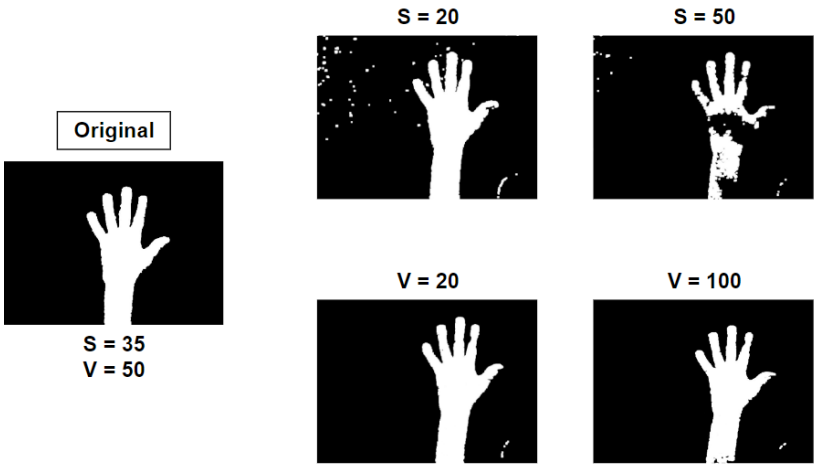


Figura 67: Valores de saturação e brilho na segmentação da mão sem luva

Examinando a Figura 67, diminuindo a saturação em relação ao original, observa-se que a máscara fica sujeita a algum ruído indesejável. Na situação oposta, ao aumentar a saturação, a máscara começa a perder o detalhe da mão. Quando é introduzido um valor de brilho superior ou inferior ao original, verifica-se que não tem impacto, pela mesma razão anterior.

Também é importante salientar que os valores para os dois métodos são fixos, o que significa que uma pessoa com uma cor de pele um pouco mais clara ou escura, não iria obter os mesmos resultados para o método sem luva, que obtive com a minha mão.

O processamento de imagem, como detalhado anteriormente, irá ter o objetivo de detetar a mão do utilizador e, subsequentemente, o número de dedos apresentado. As fases necessárias para este efeito estão presentes na Figura 68 (nesta figura só é detalhado o processamento de imagem para a luva azul, no entanto, para o método sem luva a única diferença seria os valores de HSV da máscara). Estas imagens são as obtidas pelo sistema numa situação real.

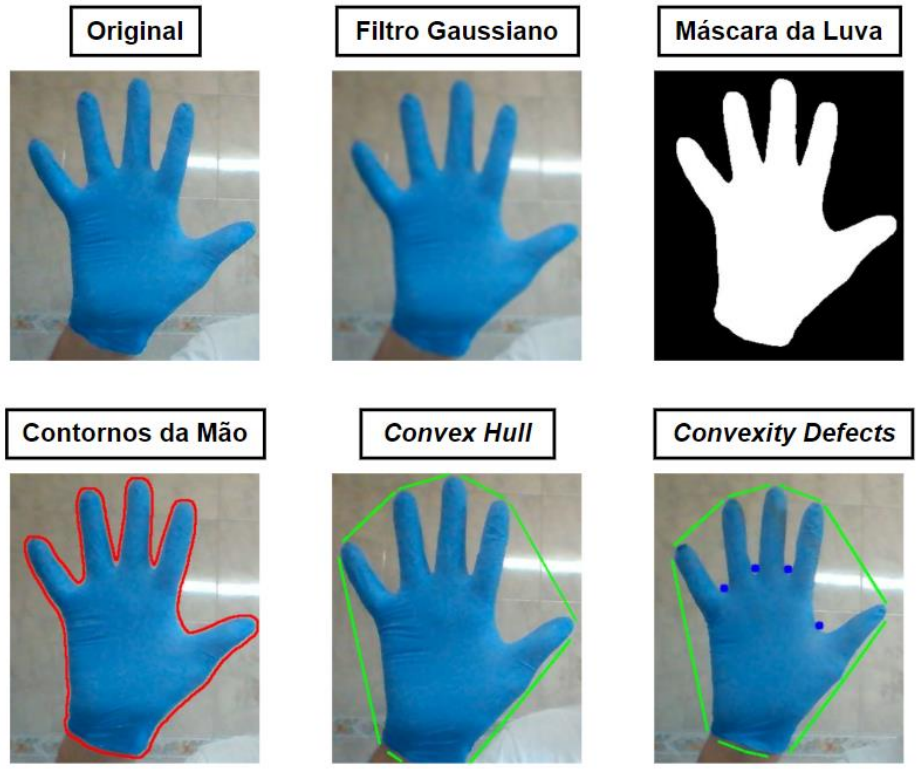


Figura 68: Fases para a deteção da mão do utilizador e do número de dedos

4.2. Importância da mão com ou sem luva

Após encontrados os valores de HSV da minha cor de pele, para possibilitar a deteção da mão, rapidamente deparei-me com um problema com este método. Os tons encontrados são muito comuns em qualquer ambiente disponível, causando conflitos no programa e leituras erradas, considerando que este irá assumir que grandes áreas indesejadas, com os valores introduzidos, fazem parte da mão do utilizador. Isto é, facilmente o sistema confundia o tom da pele com uma parede ou partes em madeira.

O método encontrado para resolver o problema anteriormente mencionado foi utilizar uma luva cirúrgica azul. Este procedimento é mais eficaz e preciso do que a mão sem

luva, porque reduz os conflitos com outros fatores com os mesmos valores de HSV, considerando que no ambiente, a cor azul da luva é menos comum do que a cor da pele.

Nesta fase, na Figura 69, foi estudado a deteção da mão do utilizador nos dois métodos em 4 ambientes com iluminações diferentes, sendo o primeiro uma parede branca sem iluminação, o segundo a mesma parede com a iluminação ligada, o terceiro uma porta castanha com luz exterior e o quarto o mesmo ambiente anterior, mas com uma luz amarelada:

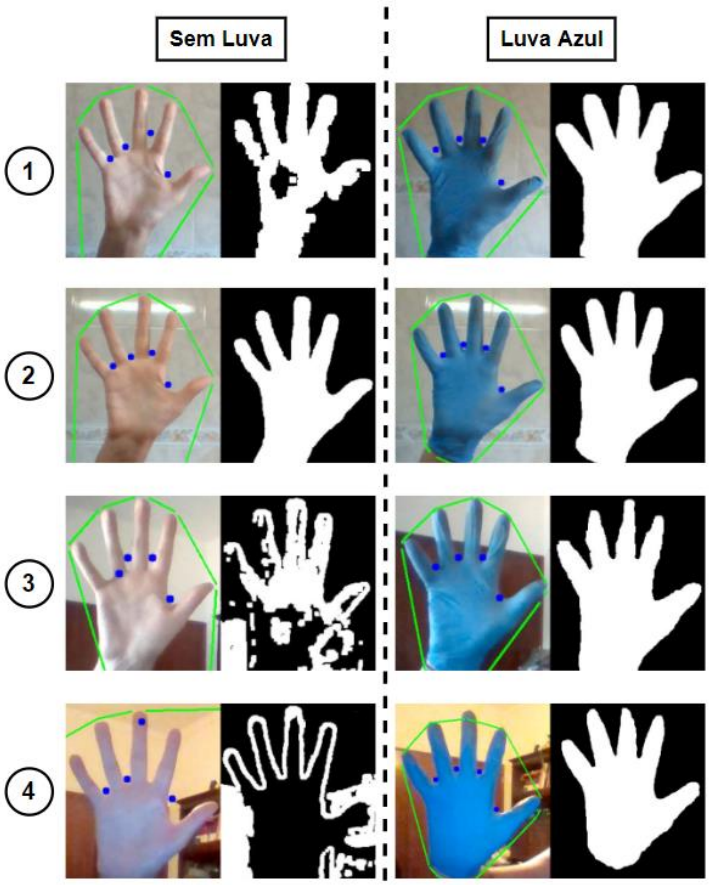


Figura 69: Testes dos dois algoritmos em quatro ambientes diferentes

Analisando a figura anterior, é possível retirar as seguintes conclusões para os dois algoritmos:

- O **método da mão sem luva** irá funcionar corretamente se houver uma parede branca (poder-se-ia também supor que a mão do utilizador está ao nível do peito deste e este usa uma camisa de cor contrastante da pele humana) por detrás da mão e se tiver boa iluminação. Nos casos em que estes fatores não se verificam, o algoritmo irá conter

muito ruído, o que irá impossibilitar a determinação do número de dedos correto e dificultar bastante a deteção da mão.

- Para o **método da mão azul**, os dois fatores anteriormente mencionados, também são importantes para o funcionamento perfeito deste algoritmo, no entanto, em casos de pouca iluminação, ou simplesmente casos não ideais, ainda consegue cumprir o seu objetivo corretamente. Há sempre exceções em que este algoritmo não irá funcionar corretamente, principalmente num ambiente localizado no exterior, onde poderia haver um objeto da mesma cor da luva, porém em ambientes fechados é o melhor método.

Na próxima fase, é estudada a eficácia dos dois algoritmos e as situações em que estes conseguem detetar a mão do utilizador e, consequentemente, demonstrar corretamente o número de dedos apresentado. Estes testes foram realizados em vários ambientes com luminosidades diferentes. Os resultados foram registados na Tabela 12.

Tabela 12: Eficácia a encontrar a mão e mostrar o número correto de dedos

	Luva Azul		***Sem Luva***	
	Deteção da Mão	**Correto número de dedos**	**Deteção da Mão**	**Correto número de dedos**
***Eficácia*:**	92 %	86 %	68%	52%

Analisando os resultados obtidos na Tabela 12, facilmente se conclui que o método sem luva não é ideal para os objetivos propostos desta dissertação, considerando o facto de apenas conseguir determinar corretamente o número de dedos da mão em metade dos casos estudados. Quanto ao método da luva azul, é possível afirmar ser a técnica ideal, conseguindo detetar o correto número de dedos, até em situações com pouca iluminação. De salientar que estes valores foram obtidos em situações extremas de iluminação, com o intuito de avaliar a robustez do algoritmo.

Considerando os resultados obtidos nas duas fases, a utilização da luva cirúrgica azul será o método principal para esta dissertação, enquanto o procedimento da mão sem luva tem o objetivo de representar o algoritmo numa perspetiva ideal em que não existe nenhum conflito ou ruído.

4.3. Deteção de Obstáculos

Neste subcapítulo será estudado a eficácia do sistema de deteção de obstáculos em várias situações disponíveis, para cada uma das opções de movimentação do robô móvel. Na Figura 70 é possível evidenciar um exemplo do funcionamento do desvio de obstáculos

Figura 70: Robô móvel a detetar e desviar-se dum obstáculo

Na Tabela 13 é possível visualizar os resultados obtidos para os dois métodos de movimentação do robô móvel. Os testes consistem na colocação de diversos obstáculos de várias formas e tamanhos no percurso do robô.

Tabela 13: Eficácia da deteção e desvio de obstáculos nas opções de movimentação

	Method A		***Method B***	
	Detection	**Prevention**	**Detection**	**Contouring**
***Effectiveness*:**	100 %	100 %	100 %	86 %

Como é evidente na tabela, o robô não tem nenhuma dificuldade na deteção do objeto presente no seu percurso. Durante o método A, considerando que para prevenir o embate com um obstáculo irá apenas andar para trás, foi obtida uma eficácia de 100%. Durante o método B, na maior parte dos testes realizados, com diferentes obstáculos colocados em diferentes ambientes, foi obtida uma eficácia de 86% no correto contorno do objeto. O robô móvel não conseguiu realizar o contorno corretamente em algumas situações testadas, em que ficava preso num ciclo entre andar para a frente e para trás.

4.4. Importância de velocidades diferentes na deteção de obstáculos

Em seguida, serão estudadas as diferentes velocidades para cada uma das situações, a fim de determinar qual das opções é a melhor.

Como mencionado anteriormente, os motores do robô são controlados com PWM e ambos têm uma frequência de 120 Hz. Para o robô andar corretamente para a frente, a uma velocidade aceitável para realizar as ações desta dissertação, foi determinado que a roda direita tem um *duty cycle* de 22% e a esquerda de 19%. Esta diferença, entre os dois motores, deve-se ao facto de ser difícil que as rodas sejam exatamente iguais, considerando que um dos lados pode potencialmente ter mais peso que o outro (apesar de se ter tentado distribuir igualmente o peso pelo chassi). Para efeitos de teste, assume-se que para andar para a frente é necessário um *duty cycle* médio de 21%.

Nesta etapa, na Tabela 14, é estudada a eficácia de diferentes velocidades nas diferentes etapas do sistema desenvolvido.

Tabela 14: Estudo da eficácia do algoritmo com velocidades diferentes

	Method A		***Method B***	
Duty Cycle	**Hand Tracking**	**Finding the Hand**	**Detect obstacle without impact**	**Bypass the obstacle**
22%:	88%	78%	100%	86%
33%:	64%	26%	92%	96%
44%:	48%	12%	56%	84%

Após os vários testes realizados para os modos disponíveis, com velocidades diferentes, é possível retirar algumas conclusões:

- Para o **método A**, uma velocidade com um *duty cycle* menor, como 22%, é o ideal a seguir a mão, como também a encontrá-la após o robô começar a rodar. Velocidades superiores a esta têm o problema de funcionar demasiado rápido e facilmente perder a mão;
- Para o **método B**, a velocidade com um *duty cycle* de 22%, foi a melhor a detetar o objeto sem que haja um embate, no entanto, foi a que teve o pior desempenho a contornar do obstáculo. A velocidade de 33% foi onde se obteve os melhores resultados e conclui-se que é a mais indicada para este modo.

4.5. Comparação com *Mediapipe*

Para esta fase da dissertação, decidiu-se comparar o algoritmo desenvolvido da luva azul, com um algoritmo comercial bastante conhecido, o rastreamento de mão do *mediapipe*.

Mediapipe é uma *framework open-source*, utilizada para a elaboração de algoritmos de aprendizagem automática para processamento de imagem, áudio, entre outros.[19] Oferece uma variedade de soluções com todo o tipo de finalidades, mas para este trabalho vou-me focar apenas na "***Mediapipe Hands***". Esta técnica utiliza redes neuronais para detetar pontos de referência de uma mão numa imagem, como por exemplo, os nós dos dedos.

Durante o desenvolvimento original desta dissertação, pensei em utilizar a solução anteriormente mencionada, para desenvolver o algoritmo controlador da movimentação do robô móvel, no entanto, após a realização de alguns testes, rapidamente percebi que a versão do *Raspberry Pi* utilizada neste trabalho (*Raspberry Pi model* 3B+) não tem poder de processamento suficiente para correr com *mediapipe*.

Na Figura 71, é possível visualizar uma comparação do número de *frames* por segundo do algoritmo da luva azul e do algoritmo de *mediapipe*, onde facilmente se percebe a desvantagem da utilização deste último.

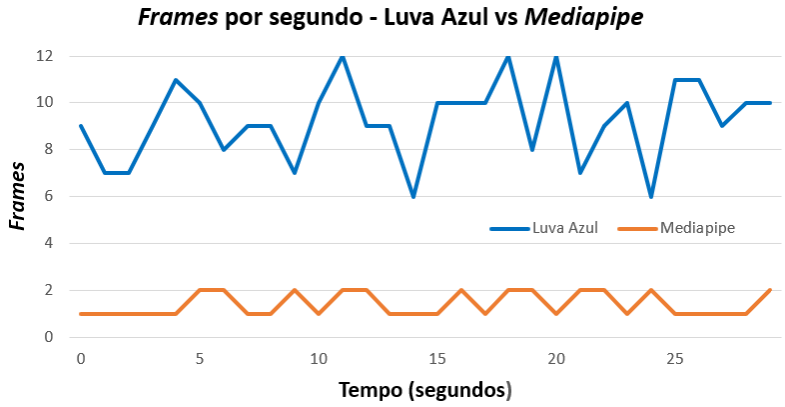


Figura 71: *Frames* por segundo do algoritmo da luva azul e *mediapipe*

Apesar de não ser a melhor alternativa em termos de *performance* para esta dissertação, o *mediapipe hands* continua a ser uma excelente opção para comparar com o algoritmo desenvolvido da luva azul. Na Tabela 15, está disponível a comparação anteriormente

[19] https://learnopencv.com/introduction-to-mediapipe/#What-is-MediaPipe, acedido a 13-08-2022

mencionada, em que é referido a eficácia da deteção da mão e do número de dedos (ou no caso do *mediapipe*, o gesto) do utilizador.

Tabela 15: Comparação da eficácia do algoritmo da luva azul com o do *mediapipe*

	Luva Azul		***Mediapipe***	
	Deteção da Mão	**Correto número de dedos**	**Deteção da Mão**	**Correto gesto identificado**
***Eficácia*:**	92 %	86 %	98%	98%

Analisando os resultados obtidos na tabela anterior, é possível verificar que utilizando o método do *mediapipe hands* se obtém uma melhor eficácia, comparativamente ao método da luva azul, contudo, esta pequena diferença de valores, não contabiliza o baixo *performance* (baixos *frames* por segundo), evidenciado na Figura 71, que provém desta técnica e que praticamente impossibilita o funcionamento correto do programa na plataforma *Raspberry Pi*.

4.6. Limitações do sistema

Durante os testes realizados nesta dissertação foram encontradas algumas limitações, como problemas durante o processamento de imagem, ou situações que ocorreram que não foram contabilizadas durante a elaboração do robô móvel.

Apesar do método da luva azul funcionar corretamente em maior parte dos casos estudados, há limitações associadas a alguns ambientes com iluminação insuficiente ou com demasiado brilho. Além disso, espaços com bastantes objetos de cor azul também pode causar conflitos. Estas duas limitações podem ser observadas na Figura 72.

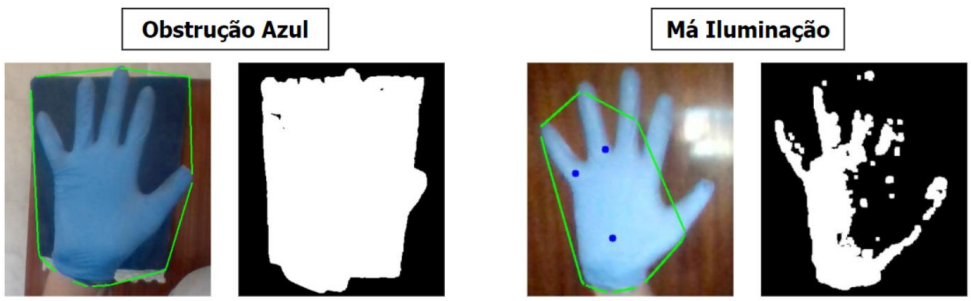


Figura 72: Limitações do método da luva azul

Outra limitação que não foi contabilizada para esta dissertação, foi o facto de o robô móvel não dispor de um sensor ultrassónico na parte de trás. Ainda que não seja um problema muito comum durante a movimentação, foi detetada uma circunstância em que o robô identifica um obstáculo à sua frente, e quando vai realizar a inversão de marcha, colide atrás com outro objeto. Esta colisão faz com que o robô fique permanentemente a tentar andar para trás sem sucesso, devido ao obstáculo. Esta situação descrita pode ser visualizada na Figura 73. Este teste foi executado de modo a demonstrar que ao colocar mais sensores neste sistema haveria outras possibilidades de deslocamento.

Figura 73: Limitação do sistema de desvio de obstáculos

Por último, a maior limitação desta dissertação é o poder de processamento proveniente do *Raspberry Pi model* 3B+. Como demonstrado na Figura 71, o valor médio de *frames* por segundo obtido durante a execução do algoritmo da luva azul é de 9-10 *frames*, o que, consequentemente, significa que o programa consegue correr corretamente, mas não da forma mais adequada. Em várias situações que foram estudadas, muitas vezes ocorre um certo atraso entre a ordem introduzida pelo utilizador e a execução da ordem pelo robô móvel. Uma solução seria otimizar o algoritmo de modo a torná-lo mais rápido, ou impor restrições de iluminação de modo a tornar os casos mais simples.

5. Conclusões

Este capítulo é o culminar do estudo e desenvolvimento que possibilitaram a implementação de um sistema de navegação por processamento de imagem num robô móvel.

Inicialmente, irão ser exibidas as conclusões fundamentais sobre o trabalho desenvolvido e os resultados obtidos. De seguida, serão abordadas as limitações da dissertação. Por fim, é apresentada uma perspetiva de futuros trabalhos, com fundamento nos conteúdos de maior importância.

5.1. Conclusões Gerais

O desenvolvimento de um sistema de navegação por processamento de imagem, para um robô móvel, é complexo e requer o correto e, simultâneo, funcionamento do conjunto de todas as fases que o compõem. Estas etapas consistem no processamento de imagem (aquisição, extração de características e deteção da mão), deteção de obstáculos e o sistema de controlo.

O robô desenvolvido durante esta dissertação é relativamente simples, contendo apenas componentes suficientes para o seu correto funcionamento. A sua simplicidade permite que seja facilmente replicado, considerando também o baixo custo. Por outro lado, facilmente se poderá acrescentar módulos adicionais de modo a ter mais funcionalidades.

O processamento de imagem tem como principal objetivo detetar a mão do utilizador e, subsequentemente, o número de dedos apresentado. Para este fim, foram estudadas duas soluções, a mão com e sem luva. Tendo em consideração o processo esclarecido em tópicos anteriores, para o funcionamento do processamento de imagem, foram obtidos corretamente os resultados esperados para os valores de HSV definidos em cada solução. O sistema sem luva está bastante dependente do ambiente a que está sujeito, conseguindo apenas detetar o número de dedos em metade das situações estudadas; a luva azul teve resultados mais positivos, funcionando corretamente em 86% dos casos. Conclui-se que o método da luva azul é o mais indicado para esta dissertação. A fim de demonstrar o quão fiável é o algoritmo, estes valores foram adquiridos em condições extremas de iluminação. Seria possível obter melhores resultados utilizando uma iluminação dedicada.

Relativamente aos algoritmos de deteção de obstáculos desenvolvidos, é possível concluir que foi um êxito, com uma eficácia de 100%. Os sistemas utilizados para os modos de seguimento de mão e desvio de obstáculos são também bastante eficazes, sendo que no primeiro obteve-se um resultado de 100% e no segundo um valor de 94%.

Com os resultados obtidos durante os testes das velocidades em cada um dos modos de controlo, conclui-se que o ideal para o seguimento da mão é uma velocidade inferior, para que o robô siga a mão de uma forma delicada. Relativamente ao desvio de obstáculos, uma velocidade um pouco maior é o mais adequado, para que o robô consiga desviar-se do obstáculo sem problemas.

Comparativamente a outros algoritmos predominantes, como o *Mediapipe*, não é o mais eficaz. No entanto, para o controlador utilizado (*Raspberry Pi model 3B+*), é o mais indicado, devido às características de memória disponível.

As limitações desta dissertação, consistem na defeituosa deteção do número de dedos devido ao brilho em demasia, à iluminação, ou obstrução de algo azul no ambiente, sendo que os sistemas de processamento de imagem estão muito dependentes da iluminação do espaço. Outros fatores são a falta de um sensor de distância na parte detrás do robô, impossibilitando a deteção de algumas situações testadas. Finalmente, a maior limitação é o poder de processamento proveniente do *raspberry model model 3B+* que estava disponível.

Quanto à linguagem de programação utilizada (*python*), é possível afirmar que foi a indicada para esta dissertação, considerando a sua versatilidade e o facto de ser *open source*, mas também devido à motivação que tinha de aprofundar o meu conhecimento sobre esta. Outra grande vantagem desta linguagem, é a quantidade de bibliotecas que podem ser adicionadas, mais especificamente o O*penCV* e *numPy*. Estas, para além de terem permitido a elaboração do algoritmo de processamento de imagem, possuem inúmeras possibilidades para qualquer outro tipo de aplicação.

Por último, é possível concluir que os resultados obtidos comprovam as metodologias adotadas. O algoritmo de processamento de imagem desenvolvido consegue desempenhar as tarefas propostas, levando assim a um sistema de navegação satisfatório para um robô móvel.

5.2. Trabalho Futuro

Os resultados obtidos nesta dissertação servem como uma base sólida para desenvolvimentos futuros no domínio da robótica móvel. Durante a elaboração do projeto foram sempre consideradas situações modulares com vista a futura expansão. De igual modo, o controlador poderá ser facilmente substituído por outro. Contudo, há algumas atualizações possíveis para o trabalho futuro, utilizando o robô móvel e o algoritmo desenvolvido nesta dissertação.

Começando pelo robô móvel, a adição de quatro rodas omnidirecionais ou *mecanum* possibilitaria que o robô pudesse deslocar-se em todas as direções, o que poderia levar a abordagens mais interessantes para a sua movimentação. No entanto, iria aumentar, consideravelmente, o custo e complexidade.

Poderia ser adicionada uma plataforma extra ao robô, de forma a torná-lo num mecanismo de transporte, onde pode ser colocado qualquer objeto em cima e, com recurso ao algoritmo de processamento de imagem, seria possível transportá-lo onde quer que seja.

Durante o desenvolvimento do robô, a questão energética foi elaborada conforme as dificuldades e inconveniências encontradas. Uma destas é a pilha recarregável de 9V, que não tem uma capacidade suficiente para um funcionamento prolongado. Tendo isto em conta, poderia ser trocada esta pilha por uma alternativa com maior capacidade.

Para melhorar o desempenho geral do controlador, seria ideal utilizar uma versão mais recente do *Raspberry Pi*, sendo recomendável o *Raspberry Pi model 4*. Este *upgrade*, iria melhorar, consideravelmente, o tempo de resposta do sistema, possibilitando a utilização de outros algoritmos. No entanto, tem a desvantagem de ser uma alternativa bastante dispendiosa, e nestes últimos tempos de crise pandémica, esse equipamento não se encontrava disponível. Contudo, com a utilização de uma nova versão mais recente do *Raspberry Pi*, poderia ser adicionado um sistema com o método de SLAM, com o intuito de criar um mapa de navegação, para expandir as possibilidades de deteção de obstáculos, ou adicionar outro modo de controlo ao robô.

6. Referências Bibliográficas

[1] **M. Bowker**, "Robot - Merriam-Webster Dictionary", Merriam Webster, [Online]. Available: https://www.merriam-webster.com/dictionary/robot.

[2] **R. I. Association**, "Robot Info", Robitik Sistem, 2019. [Online]. Available: http://www.robotiksistem.com/robotinfo.html.

[3] **K. S. Fu, R. C. Gonzalez and C. S. G. Lee**, *Robotics - Control, Sensing, Vision and Intelligence*, New York: McGraw-Hill Book Inc, 1987.

[4] **M. Shelley**, "Frankenstein", Londres: Lackington, Hughes, Harding, Mavor & Jones, 1818.

[5] **M. Groover, M. Weiss, R. Nagel and N. Odrey**, *Industrial Robotics, Technology, programming, and applications*, New Delhi: McGraw-Hill Book Inc., 1987.

[6] **R. Murphy and D. Woods**, "Beyond Asimov: The Three Laws of Responsible Robotics", *Intelligent Systems, IEEE*, 2009.

[7] **R. Murphy**, *Introduction to AI Robotics*, Cambridge: MIT Press, 2000.

[8] **R. Murray, Z. Li and S. Sastry**, *A Mathematical Introduction to Robotic Manipulation*, USA: CRC Press, Inc., 1994.

[9] **G. Dudek and M. Jenkin**, *Computational Principles of Mobile Robotics*, New York: Cambridge University Press, 2010.

[10] **U. Nehmzow**, *Mobile Robotics: A Practical Introduction*, London: Springer, 2003.

[11] **W. G. Walter**, "An Imitation of Life", vol. 182, Scientific American, 1950.

[12] **W. G. Walter**, "A Machine that Learns", vol. 51, Scientific American, 1951.

[13] **N. Nilsson**, "Shakey the Robot", Artificial Intelligence Center, SRI International, 1984.

[14] **C. Angle**, “Genghis, a Six Legged Autonomous Walking Robot”, Massachusetts Institute of Technology, 1989.

[15] **P. Ehlert**, “The use of Artificial Intelligence in autonomous mobile robots”, Delft University of Technology, 1999.

[16] **P. Morin and C. Samson**, “Motion Control of Wheeled Mobile Robots”, Berlim: Springer Berlin Heidelberg, 2008.

[17] **L. Bruzzone and G. Quaglia**, “Review article: locomotion systems for ground mobile robots in unstructured environments”, *Mechanical Sciences*, 2012.

[18] **K. Shabalina, A. Sagitov and E. Magid**, “Comparative Analysis of Mobile Robot Wheels Design”, em *11th International Conference on Developments in eSystems Engineering (DeSE)*, 2018.

[19] **M. Crenganis, C. Biris and C. Girjob**, “Mechatronic Design of a Four-Wheel drive mobile robot and differential steering”, em *MATEC Web Conference - 10th International Conference on Manufacturing Science and Education*, 2021.

[20] **N. Rodriguez, L. Ormaechea, R. Cabás, A. Gonzalez, E. Ottaviano and A. Huete**, *Advanced Mechanics in Robotic System*, Springer, 2011.

[21] **I. Doroftei, V. Grosu and V. Spinu**, “Omnidirectional mobile robot - Design and Implementation”, em *Bioinspiration and Robotics Walking and Climbing Robots*, M. Habib, Ed., InTech, 2007.

[22] **F. Rubio, F. Valero and C. Albert**, “A review of mobile robots: Concepts, methods, theoretical framework, and applications”, *International Journal of Advanced Robotic Systems*, 2019.

[23] **K. Kanjanawanishkul**, “Omnidirectional wheeled mobile robots: Wheel types and practical applications”, *International Journal of Advanced Mechatronic Systems*, vol. 6, nº 6, p. 289–302, 2015.

[24] **M. I. Ribeiro**, "Sensores em Robótica - Como é que os robots sentem o ambiente envolvente", Institute For Systems and Robotics - Lisboa, [Online]. Available: http://users.isr.ist.utl.pt/~mir/pub/sensores.pdf.

[25] **J. Zhou**, "Indoor Localization of a Mobile Robot Using Sensor Fusion", Massey University, New Zealand, 2011.

[26] **K. Sims**, "Orientable Single-Distance Codes for Absolute Incremental Encoders", Brigham Young University, 2020.

[27] **P. Limpisathian**, "Design of Low-Cost High Accuracy Microcontroller-Based Revolver Emulator", University of Akron, 2013.

[28] **M. Pakdaman and M. Sanaatiyan**, "Design and Implementation of Line Follower Robot", em *Second International Conference on Computer and Electrical Engineering*, 2009.

[29] **S. Mehta, A. Patel and J. Mehta**, "CCD or CMOS Image Sensor For Photography", em *IEEE ICCSP 2015 conference*, 2015.

[30] **M. Lillestøl**, "Design and test of a CMOS image sensor with global shutter and High Dynamic Range", University of Oslo, 2017.

[31] **W. Lee and C. Cai**, "An Orientation Sensor for Mobile Robots Using Differentials", *International Journal of Advanced Robotic Systems,* vol. 10, p. 1, 2013.

[32] **V. Passaro, A. Cuccovillo**, L. Vaiani, M. De Carlo, C. Campanella, "Gyroscope Technology and Applications: A Review in the Industrial Perspective", *Sensors,* 2017.

[33] **V. Skvortzov, H. Lee, S. Bang and Y. Lee**, "Application of Electronic Compass for Mobile Robot in an Indoor Environment", em *Proceedings IEEE International Conference on Robotics and Automation*, 2007.

[34] **M. Hamid, A. Adom, N. Rahim and M. Rahiman**, "Navigation of mobile robot using Global Positioning System (GPS) and obstacle avoidance system with

commanded loop daisy chaining application method", em *5th International Colloquium on Signal Processing*, 2009.

[35] **J. Sasiadek, Y. Lu and V. Polotski**, "Navigation of Autonomous Mobile Robots - Invited Paper", *Lecture Notes in Control and Information Sciences,* 2009.

[36] **C. Gamallo, C. Regueiro, P. Quintía and M. Mucientes**, "Omnivision-based KLD-Monte Carlo Localization", *Robotics and Autonomous Systems,* 2010.

[37] **S. Malagon-Soldara, M. Toledano-Ayala, G. Soto-Zarazua, R. Carillo-Serrano and E. Rivas-Araiza**, "Mobile Robot Localization: A Review of Probabilistic Map-Based Techniques", *IAES International Journal of Robotics and Automation (IJRA),* 2015.

[38] **Y. Elor and A. Bruckstein**, "A "thermodynamic" approach to multi-robot cooperative localization", *Theoretical Computer Science,* 2012.

[39] **M. Ahmadi, A. Khayatian and P. Karimaghaee**, "Attitude estimation by divided difference filter in quaternion space", *Acta Astronautica,* 2012.

[40] **A. Santos**, "Autonomous Mobile Robot Navigation using Smartphones", Instituto Superior Técnico, Lisboa, 2008.

[41] **S. Thrun**, "Robotic Mapping: A Survey", Carnegie Mellon University, 2002.

[42] **P. Teleweck and B. Chandrasekaran**, "Path Planning Algorithms and Their Use in Robotic Navigation Systems", *Journal of Physics: Conference Series,* 2019.

[43] **K. Karur, N. Sharma, C. Dharmatti and J. Siegel**, "A Survey of Path Planning Algorithms for Mobile Robots", *Vehicles,* 2021.

[44] **D. Ferguson, M. Likhachev and A. Stentz**, "A Guide to Heuristic-based Path Planning", em *International Conference on Automated Planning and Scheduling*, 2005.

[45] **C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. Leonard**, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age", *IEEE Transactions on Robotics,* 2016.

[46] **A. Saman and A. Lotfy**, "An implementation of SLAM with extended Kalman filter", em *6th International Conference on Intelligent and Advanced Systems (ICIAS)*, 2016.

[47] **A. Kour, V. Yv, V. Maheshwari and D. Prashar**, "A Review on Image Processing", *International Journal of Electronics Communication and Computer Engineering*, 2012.

[48] **M. Sonka, V. Hlavac and R. Boyle**, *Image Processing, Analysis, and Machine Vision*, CL Engineering; 2nd edition, 1998.

[49] **N. S. Kumar, A. K. Sangaiah, M. Arun e S. Anand**, *Advanced Image Processing Techniques and Applications*, IGI Global, 2017.

[50] **D. Sugimura, T. Mikami, H. Yamashita and T. Hamato**, "Enhancing Color Images of Extremely Low Light Scenes Based on RGB/NIR Images Acquisition With Different Exposure Times", *IEEE Transactions on Image Processing*, 2015.

[51] **J. Pandya**, "Image Acquisition and Techniques", *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, vol. 5, nº 2, 2021.

[52] **A. Yadav and P. Yadav**, *Digital Image Processing*, University Science Press, 2009.

[53] **V. Mishra, S. Kumar and N. Shukla**, "Image Acquisition and Techniques to Perform Image Acquisition", *SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology*, 2017.

[54] **R. E. Twogood and F. Sommer**, "Digital Image Processing", *IEEE Transactions on Nuclear Science*, 1982.

[55] **B. Jähne**, *Digital Image Processing 6th Edition*, Springer, 2005.

[56] **S. Saha, C. Pal, R. Paul, S. Maity and S. Sau**, "A brief experience on journey through hardware developments for image processing and its applications on Cryptography", 2012.

[57] **A. Sultana and M. Meenakshi**, "Design and development of FPGA based adaptive thresholder for image processing applications", em *IEEE Recent Advances in Intelligent Computational Systems*, 2011.

[58] **W. K. Pratt**, *Digital Image Processing PIKS Scientific Inside*, Wiley-Interscience; 4th edition, 2007.

[59] **A. Koschan and M. Abidi**, *Digital Color Image Processing*, Wiley-Interscience; 1st edition, 2008.

[60] **G. Gupta and R. Chandel**, "Image Filtering Algorithms and Techniques: A Review", *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, nº 10, 2013.

[61] **J. Blackledge**, "Chapter 16 - Segmentation and Edge Detection", em *Detection Digital Image Processing*, Cambridge, Woodhead Publishing, 2005.

[62] **G. d. Vale and Alurir d. Poz**, "O processo de detecção de bordas de Canny: Fundamentos, algoritmos e avaliação experimental", Faculdade de Ciências e Tecnologia - FCT, 2002.

[63] **A. Brustolin**, "Edge Detection Based on Global and Local Parameters of the Image", *Computing Research Repository (CoRR),* 2015.

[64] **A. Jose, D. Merlin, N. Joseph, E. George and A. Vadukoot**, "Performance study of edge detection operators", em *Conference: 2014 International Conference on Embedded Systems (ICES)*, 2014.

[65] **K. Shrimali**, "Convex Hull using OpenCV in Python and C++," LearnOpenCv, [Online]. Available: https://learnopencv.com/convex-hull-using-opencv-in-python-and-c/.

[66] **A, Dhawan and V. Honrao**, "Implementation of Hand Detection based Techniques for Human Computer Interaction", *International Journal of Computer Applications,* vol. 72, 2013.

[67] **D. Carvalho**, "Processamento de Imagem num Simulador de Armazenamento Automático", Instituto Superior de Engenharia de Lisboa, 2016.

[68] **A. Mira**, "Interface Remota Genérica para Braços Robóticos", Instituto Superior de Engenharia de Lisboa, 2021.

[69] **A. Garcia, M. Carvalho and F. da Cruz**, "Robõ μ-Rato", Instituto Superior de Engenharia de Lisboa, 2007.

[70] **C. Coutinho**, "Robótica Móvel – Sistema de Condução Autónoma", Instituto Superior de Engenharia de Lisboa, 2014.

[71] **D. Farrington**, R. Fleming, D. Hill and J. Morrow, "Ultrasonic Mapping Device", Worcester Polytechnic Institute, Worcester, Massachusetts, 2016.

[72] **F. L. Chernousko**, "Locomotion Principles for Mobile Robotic Systems", *Procedia Computer Science*, 2017.

[73] **Q. Li, R. Li, K. Ji and W. Dai**, "Kalman Filter and Its Application", em *8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, 2015.

[74] "Phototaxis (2021)," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Phototaxis.

[75] "Raio (roda) (2022)," Wikipedia, [Online]. Available: https://pt.wikipedia.org/wiki/Raio_(roda).

7. Anexo – Algoritmo desenvolvido

```
# --------------------- Instituto Superior de Engenharia de Lisboa ------------------
# -------- Área Departamental de Engenharia Eletrotécnica Energia e Automação -------
# ------------------------------------------------------------------------------------
# ------- Dissertação - Navegação de um Robô Móvel por Processamento de Imagem ------
# ---------------- Algoritmo de Controlo do Movimento do robô Móvel ----------------
# ------------------------------------------------------------------------------------
# ----------------------- Mestrado em Engenharia Eletrotécnica ----------------------
# ------------------------ Alexandre Duarte Goulão - A47990 -------------------------

import cv2
import numpy as np
import math
import time
import RPi.GPIO as gpio

def callback(x):
    global H_low, H_high, S_low, S_high, V_low, V_high

    # Atribuir o valor da posição da barra a H, S, V, valores altos e baixos
    H_low = cv2.getTrackbarPos('low H', 'controls')
    H_high = cv2.getTrackbarPos('high H', 'controls')
    S_low = cv2.getTrackbarPos('low S', 'controls')
    S_high = cv2.getTrackbarPos('high S', 'controls')
    V_low = cv2.getTrackbarPos('low V', 'controls')
    V_high = cv2.getTrackbarPos('high V', 'controls')

gpio.setwarnings(False)
gpio.setmode(gpio.BCM)
gpio.setup(17, gpio.OUT)            # IN1 RODA DIREITA
gpio.setup(27, gpio.OUT)            # IN2 RODA DIREITA
gpio.setup(13, gpio.OUT)            # ENB RODA DIREITA
gpio.setup(12, gpio.OUT)            # ENA RODA ESQUERDA
gpio.setup(23, gpio.OUT)            # IN1 RODA ESQUERDA
gpio.setup(24, gpio.OUT)            # IN2 RODA ESQUERDA
gpio.setmode(gpio.BCM)

motor1gpio = gpio.PWM(12, 120)      # ENA -> PWM -> |RODA DIREITA| -> Freq 120 Hz
motor2gpio = gpio.PWM(13, 120)      # ENB -> PWM -> |RODA ESQUERDA| -> Freq 120 Hz

def forward():
    dc = 32                         # VALOR DE PWM -> |RODA DIREITA| -> 32%
    dc2 = 29                        # VALOR DE PWM -> |RODA ESQUERDA| -> 29%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, True)           # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, False)          # IN2 -> |RODA DIREITA| -> REVERSE
```

```
    gpio.output(24, True)           # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE

def reverse():
    dc = 31                         # VALOR DE PWM -> |RODA DIREITA| -> 31%
    dc2 = 27                        # VALOR DE PWM -> |RODA ESQUERDA| -> 27%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, False)          # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, True)           # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, False)          # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, True)           # IN2 -> |RODA ESQUERDA| -> REVERSE

def stop():
    gpio.output(17, False)          # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(27, False)          # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(23, False)          # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(24, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE

def right_turn():
    dc = 28                         # VALOR DE PWM -> |RODA DIREITA| -> 28%
    dc2 = 38                        # VALOR DE PWM -> |RODA ESQUERDA| -> 38%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, True)           # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, False)          # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, True)           # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE

def left_turn():
    dc = 38                         # VALOR DE PWM -> |RODA DIREITA| -> 38%
    dc2 = 27                        # VALOR DE PWM -> |RODA ESQUERDA| -> 27%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, True)           # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, False)          # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, True)           # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE

def slow_forward():
    dc = 22                         # VALOR DE PWM -> |RODA DIREITA| -> 22%
    dc2 = 19                        # VALOR DE PWM -> |RODA ESQUERDA| -> 19%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, True)           # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, False)          # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, True)           # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE
```

```
def slow_reverse():
    dc = 21                         # VALOR DE PWM -> |RODA DIREITA| -> 21%
    dc2 = 17                        # VALOR DE PWM -> |RODA ESQUERDA| -> 17%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, False)          # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, True)           # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, False)          # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, True)           # IN2 -> |RODA ESQUERDA| -> REVERSE

def slow_right_turn():
    dc = 18                         # VALOR DE PWM -> |RODA DIREITA| -> 18%
    dc2 = 28                        # VALOR DE PWM -> |RODA ESQUERDA| -> 28%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, True)           # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, False)          # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, True)           # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE

def slow_left_turn():
    dc = 28                         # VALOR DE PWM -> |RODA DIREITA| -> 28%
    dc2 = 17                        # VALOR DE PWM -> |RODA ESQUERDA| -> 17%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, True)           # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, False)          # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, True)           # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE

def hard_right_turn():
    dc = 20                         # VALOR DE PWM -> |RODA DIREITA| -> 20%
    dc2 = 20                        # VALOR DE PWM -> |RODA ESQUERDA| -> 20%
    motor1gpio.start(dc)            # LIGAR O PWM -> |RODA DIREITA|
    motor2gpio.start(dc2)           # LIGAR O PWM -> |RODA ESQUERDA|
    gpio.output(27, False)          # IN1 -> |RODA DIREITA| -> FORWARD
    gpio.output(17, True)           # IN2 -> |RODA DIREITA| -> REVERSE
    gpio.output(24, True)           # IN1 -> |RODA ESQUERDA| -> FORWARD
    gpio.output(23, False)          # IN2 -> |RODA ESQUERDA| -> REVERSE

# Definir os pinos utilizados para os sensores ultrassónicos
gpio_trigger_dir = 20
gpio_echo_dir = 21
gpio_trigger_esq = 5
gpio_echo_esq = 6
# Inicializar variáveis para os sensores ultrassónicos
StartTimeEsq = 0
StopTimeEsq = 0
StartTimeDir = 0
```

```
StopTimeDir = 0
flag = 0
maxTime = 0.04

gpio.setup(gpio_trigger_dir, gpio.OUT)      # TRIG SENSOR DIREITO
gpio.setup(gpio_echo_dir, gpio.IN)          # ECHO SENSOR DIREITO
gpio.setup(gpio_trigger_esq, gpio.OUT)      # TRIG SENSOR ESQUERDO
gpio.setup(gpio_echo_esq, gpio.IN)          # ECHO SENSOR ESQUERDO
gpio.output(gpio_trigger_dir, False)        # FORCAR TRIG SENSOR DIREITO OFF
gpio.output(gpio_trigger_esq, False)        # FORCAR TRIG SENSOR ESQUERDO OFF

# Variáveis
pTime = 0
cTime = 0
capture = 0
areahull = 0
dedos = 0
comando = 0
seguimento = 0
obstaculo = 0
t_check = None

# Variáveis para os prints
start = 0
start2 = 0
start3 = 0
start4 = 0

# Variáveis para dar return no comando
first = 0
second = 0
third = 0
fourth = 0
fifth = 0

cap_region_x_begin = 0.5  # Ponto de início da REGIAO DE INTERESSE
cap_region_y_end = 0.8    # Ponto final da REGIAO DE INTERESSE

# HSV low and high values
H_low = 0
H_high = 179
S_low = 0
S_high = 255
V_low = 0
V_high = 255

# Ligar a camera do robô
cap = cv2.VideoCapture(0)
```

```
# Janela para controlar os valores de HSV
cv2.namedWindow('controls', 2)
cv2.resizeWindow("controls", 400, 200)

# -------------------------- LUVA AZUL ---------------------------
cv2.createTrackbar('low H', 'controls', 90, 179, callback)
cv2.createTrackbar('high H', 'controls', 119, 179, callback)
cv2.createTrackbar('low S', 'controls', 100, 255, callback)
cv2.createTrackbar('high S', 'controls', 255, 255, callback)
cv2.createTrackbar('low V', 'controls', 80, 255, callback)
cv2.createTrackbar('high V', 'controls', 255, 255, callback)
# ------------------------- MÃO SEM LUVA -------------------------
# cv2.createTrackbar('low H', 'controls', 5, 179, callback)
# cv2.createTrackbar('high H', 'controls', 13, 179, callback)
# cv2.createTrackbar('low S', 'controls', 20, 255, callback)
# cv2.createTrackbar('high S', 'controls', 255, 255, callback)
# cv2.createTrackbar('low V', 'controls', 50, 255, callback)
# cv2.createTrackbar('high V', 'controls', 255, 255, callback)

font = cv2.FONT_HERSHEY_SIMPLEX

def sensores():
# ------------------------------- SENSOR DIREITO -------------------------------

    # Aplicar pulso de 1 segundo para acionar sensor direito
    gpio.output(gpio_trigger_dir, True)
    time.sleep(0.00001)
    gpio.output(gpio_trigger_dir, False)

    StartTimeDir = time.time()
    timeoutDir = StartTimeDir + maxTime

    # Guardar o tempo de envio do sensor direito
    while gpio.input(gpio_echo_dir) == 0 and StartTimeDir < timeoutDir:
        StartTimeDir = time.time()

    StopTimeDir = time.time()
    timeoutDir = StopTimeDir + maxTime

    # Guardar o tempo de chegada do sensor direito
    while gpio.input(gpio_echo_dir) == 1 and StopTimeDir < timeoutDir:
        StopTimeDir = time.time()

    # Diferença de tempo entre o envio e a chegada do sensor direito
    TimeElapsedDir = StopTimeDir - StartTimeDir

    # Multiplicar pela velocidade do som (34300 cm/s) e dividir por 2, por ser envio e chegada
    distanceDir = (TimeElapsedDir * 34300) / 2
```

```
# -------------------------------- SENSOR ESQUERDO --------------------------------
    # Aplicar pulso de 1 segundo para acionar sensor esquerdo
    gpio.output(gpio_trigger_esq, True)
    time.sleep(0.00001)
    gpio.output(gpio_trigger_esq, False)

    StartTimeEsq = time.time()
    timeoutEsq = StartTimeEsq + maxTime

    # Guardar o tempo de envio do sensor esquerdo
    while gpio.input(gpio_echo_esq) == 0 and StartTimeEsq < timeoutEsq:
        StartTimeEsq = time.time()

    StopTimeEsq = time.time()
    timeoutEsq = StopTimeEsq + maxTime

    # Guardar o tempo de chegada do sensor esquerdo
    while gpio.input(gpio_echo_esq) == 1 and StopTimeEsq < timeoutEsq:
        StopTimeEsq = time.time()

    # Diferença de tempo entre o envio e a chegada do sensor esquerdo
    TimeElapsedEsq = StopTimeEsq - StartTimeEsq

    # Multiplicar pela velocidade do som (34300 cm/s) e dividir por 2, por ser envio e chegada
    distanceEsq = (TimeElapsedEsq * 34300) / 2

    return distanceDir, distanceEsq

# Função para obter o número de dedos na mão do utilizador
def getDedos():
# Código para encontrar o número de defeitos devido aos dedos -> Código exemplo do OpenCV
# (Utilizado para calcular o angulo que os espaçamentos entre os dedos fazem)
# Tuples são utilizados para armazenar múltiplos itens numa só variável
# Retorna uma matriz onde cada linha tem valores (ponto inicial, final, mais distante e
distancia aproximada ao ponto mais distante)
# Desenha-se um ponto inicial e um ponto final de união de linhas, depois desenha-se um círculo
no ponto mais distante.
    global l, arearatio, dedos

    if arearatio > 0:                  # So entra neste ciclo se for detetada uma mão
        for i in range(defects.shape[0]):
            s, e, f, d = defects[i, 0]  # d -> dist aproximada ao ponto mais distante
            start = tuple(approx[s][0]) # Ponto inicial da matriz
            end = tuple(approx[e][0])   # Ponto final da matriz
            far = tuple(approx[f][0])   # Ponto mais distante

            # Encontrar o comprimento de todos os lados do triangulo do espaço entre os dedos
            a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
            b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
```

```
            c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)

            # Calcular a area do triangulo
            s = (a + b + c) / 2
            area = math.sqrt(s * (s - a) * (s - b) * (s - c))

            # Distancia entre um ponto e o convex hull
            dist = (2 * area) / a

            # Aplicar a regra do cosseno
            angulo = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 57

            # Ignorar ângulos > 90 e ignorar pontos que estão muito próximos do convex hull
            (geralmente têm muito ruído)
            if angulo <= 90 and dist > 30 and areahull > 5000:
                l += 1
                cv2.circle(img, far, 5, [255, 0, 0], -1)

            # Desenhar linhas a volta da mao (sao as linhas do convex hull)
            cv2.line(img, start, end, [0, 255, 0], 2)

    l += 1

    # ------------- Colocar no plot os números de dedos correspondentes -------------

    # Caso detete so 1 espaçamento, pode representar 0 ou 1 dedos
    if l == 1:
        if areacnt < 2000:
            cv2.putText(img, 'Mao nao detetada', (0, 50), font, 1, (0, 0, 255), 3, cv2.LINE_AA)

        else:
            # Se a area não coberta pela mão for inferior a 12, significa que a mão esta
fechada, logo 0 dedos
            if arearatio < 12:
                cv2.putText(img, '0', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
                dedos = 0

            else:
                # Representa 1 dedo
                cv2.putText(img, '1', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
                dedos = 1

    # Caso detete 2 espaçamentos, representa 2 dedos
    elif l == 2:
        cv2.putText(img, '2', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

    # Caso detete 3 espaçamentos, representa 3 dedos
    elif l == 3:
        cv2.putText(img, '3', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
```

```
    # Caso detete 4 espaçamentos, representa 4 dedos
    elif l == 4:
        cv2.putText(img, '4', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

    # Caso detete 5 espaçamentos, representa 5 dedos
    elif l == 5:
        cv2.putText(img, '5', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
        dedos = 5

    # Caso detete mais de 5 espaçamentos, esta a detetar mais alguma coisa para alem da mão
    else:
        cv2.putText(img, 'ERRO! MAIS DE 5', (10, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

    return l, dedos

# Função executada apos a ligação do comando
def comandoLigado():
    # Se o comando tiver ligado
    global comando, start2, l, areacnt, arearatio, dedos, img, seguimento, obstaculo

    if comando == 1:
        if start2 == 0:
            print('\n\n\n\n\n ------------------------------ \n ')
            print('Comando ligado!')
            print('Opções:')
            print('. 2 Dedos -> Seguimento da Mão;')
            print('. 4 Dedos -> Evitamento de Obstáculos. \n')
            start2 = 1

        cv2.putText(img, 'Comando ON', (0, 100), font, 1, (0, 255, 0), 3, cv2.LINE_AA)
        if l == 1:
            if areacnt < 2000:
                cv2.putText(img, 'Mao nao detetada', (0, 50), font, 1, (0, 0, 255), 3,
cv2.LINE_AA)

            else:
                # Se a area não coberta pela mão for inferior a 12, significa que a mão esta
fechada, logo 0 dedos
                if arearatio < 12:      # DEDOS = 0
                    cv2.putText(img, '0', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
                    dedos = 0

                else:                   # DEDOS = 1
                    cv2.putText(img, '1', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
                    dedos = 1

        elif l == 2:                    # DEDOS = 2
            cv2.putText(img, '2', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
```

```
            dedos = 2

        elif l == 3:                    # DEDOS = 3
            cv2.putText(img, '3', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
            dedos = 3

        elif l == 4:                    # DEDOS = 4
            cv2.putText(img, '4', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
            dedos = 4

        elif l == 5:                    # DEDOS = 5
            cv2.putText(img, '5', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
            dedos = 5

        elif l == 6:
            cv2.putText(img, 'ERRO, MAIS DE 5', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

        else:
            cv2.putText(img, 'ERRO! MAIS DE 5', (10, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

        # Verificar se foi detetado 2 dedos durante 1 segundo
        checkSecond()

        # Verificar se foi detetado 4 dedos durante 1 segundo
        checkThird()

    else:
        cv2.putText(img, 'Comando OFF', (0, 100), font, 1, (0, 255, 0), 3, cv2.LINE_AA)
        cv2.putText(img, '"5" para ligar comando', (10, 150), font, 1, (148, 0, 211), 3,
cv2.LINE_AA)

    return seguimento, obstaculo

# Função para que o robô movel execute a movimentação com seguimento da mão do utilizador
def seguimentoMao():
    global distanceDir, distanceEsq, start3, flag, areahull2, direcao, t_check, areahull,
arearatio, x, img, seguimento

    if start3 == 0:
            print('\n\n\n\n\n ------------------------------ \n ')
            print('Seguimento da Mão ligado!')
            print('Opções:')
            print('. Esquerda -> Robô segue para a esquerda;')
            print('. Meio -> Robô segue em frente;')
            print('. Direita -> Robô segue para a direita;')
            print('. Muito perto da Mão -> Robô segue para tras;')
            print('. 5 Dedos -> Voltar ao comando. \n')
            start3 = 1
```

```
    # Obter os valores de distância dos dois sensores ultrassónicos em real time
    distanceDir, distanceEsq = sensores()

    cv2.rectangle(img, (159, 0), (478, 478), (0, 0, 255), 2)
    cv2.putText(img, areahull2, (0, 450), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)

    # Se a distancia dos dois sensores for maior do que 10, não há obstáculo a frente do robô
    if distanceDir > 10 or distanceEsq > 10 and flag == 0:
        # Se a area detetada for entre estes valores, o robô ira seguir a mão normalmente
        if 10000 < areahull < 80000:
            # Variável para contar o tempo que o robô não encontra a mão
            t_check = None
            # So vai funcionar se o utilizador mostrar ao robô pelo menos 2 dedos
            if l >= 2:
                # Andar para a frente
                if 159 < x < 478:
                    cv2.putText(img, 'Frente', (410, 50), cv2.FONT_HERSHEY_COMPLEX, 2, (148, 0,
211), 3, cv2.LINE_AA)
                    slow_forward()

                # Andar para a esquerda
                elif x <= 159:
                    cv2.putText(img, 'Esquerda', (300, 50), cv2.FONT_HERSHEY_COMPLEX, 2, (148,
0, 211), 3, cv2.LINE_AA)
                    slow_right_turn()

                # Andar para a direita
                elif x >= 478:
                    cv2.putText(img, 'Direita', (320, 50), cv2.FONT_HERSHEY_COMPLEX, 2, (148, 0,
211), 3, cv2.LINE_AA)
                    slow_left_turn()
            else:
                # Se não detetar pelo menos 2 dedos não ocorre movimentação
                cv2.putText(img, 'NULL', (450, 50), cv2.FONT_HERSHEY_COMPLEX, 2, (148, 0, 211),
3, cv2.LINE_AA)

        else:
            # Se a area for inferior a 10000, significa que o robô não deteta a mão
            # Se passado 3 segundos não a encontrar, ira rodar ate a detetar
            if areahull < 10000:
                cv2.putText(img, 'NULL', (450, 50), cv2.FONT_HERSHEY_COMPLEX, 2, (148, 0, 211),
3, cv2.LINE_AA)
                if t_check == None:
                    t_check = time.time() + 3   # Verificar se passou 3 segundos

                # Se passarem 3 segundos
                if time.time() > t_check:
                    hard_right_turn()
```

```
                # Se nao passarem 3 segundos
                else:
                    stop()

            # Se o robô estiver muito perto da mão, ira andar para trás
            elif areahull >= 80000:
                cv2.putText(img, 'Reverse', (320, 50), cv2.FONT_HERSHEY_COMPLEX, 2, (148, 0,
211), 3, cv2.LINE_AA)
                slow_reverse()

    # Caso a distância de pelo menos um dos sensores for inferior a 10 cm, ira parar
inicialmente e depois andar para trás
    else:
        stop()
        flag = 1

    # Robô anda para trás ate a distância dos dois ser maior do que 15 cm
    if flag == 1:
        if distanceDir <= 15 or distanceEsq <= 15:
            reverse()
            print('Estou a dar reverse')

        # Quando a distância já e maior do que 15 cm, ira parar durante 1 segundo ate voltar a
seguir a mão
        else:
            stop()
            time.sleep(1)
            flag = 0

    # Verificar se foram mostrado 5 dedos durante 1 segundo para ligar o comando
    # Caso tenha acontecido, volta a ligar o comando e a movimentação é parada
    seguimento = checkFourth()

# Função para que o robô movel execute a movimentação com evitamento de obstáculos
def evitarObstaculos():
    global distanceDir, distanceEsq, start4, flag, img, obstaculo, start

    cv2.putText(img, 'Evitar Obstaculos', (0, 300), cv2.FONT_HERSHEY_COMPLEX, 2, (148, 0, 211),
3, cv2.LINE_AA)
    if start4 == 0:
        print('------------------------------- \n ')
        print('Evitamento de Obstáculos ligado!')
        print('Robô irá seguir em frente até encontrar um obstáculo.')
        print('Ao encontrar um obstaculo, ira recuar e efetuar o correto desvio')
        print('. 5 Dedos -> Voltar ao comando \n')
        start4 = 1

    cv2.rectangle(img, (159, 0), (478, 478), (255, 0, 255), 2)
```

```
    # Obter os valores de distância dos dois sensores ultrassónicos em real time
    distanceDir, distanceEsq = sensores()

    # Passos para colocar as distâncias dos sensores no plot em real time
    distanceDir2 = round(distanceDir, 2)
    distanceEsq2 = round(distanceEsq, 2)

    distanceDir2 = int(distanceDir2)
    distanceEsq2 = int(distanceEsq2)

    distanceDir2 = str(distanceDir2)
    distanceEsq2 = str(distanceEsq2)

    print("Distancia Sensor Direito = %.1f cm" % distanceDir)
    print("Distancia Sensor Esquerdo = %.1f cm \n" % distanceEsq)

    cv2.putText(img, distanceDir2, (500, 450), cv2.FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 3)
    cv2.putText(img, distanceEsq2, (0, 450), cv2.FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 3)

    # Se o robô não detetar a mão do utilizador ira executar a movimentação por desvio de
obstáculos
    if areahull < 20000:

        # Quando encontra um obstáculo a menos de 10 cm, começa por parar
        if (distanceDir <= 10 or distanceEsq <= 10) and flag == 0:
            stop()
            flag = 1

        elif flag == 1:
            # Vai andar para trás ate a distância dos sensores ser maior ou igual do que 15
            if distanceDir <= 15 or distanceEsq <= 15:
                reverse()
                print('Estou a dar reverse')
            # Logo apos irá parar
            else:
                stop()
                flag = 2

        elif flag == 2:
            # Se a distancia do sensor da direita for maior do que o da esquerda ira executar as
funções de flag = 3
            if distanceDir > distanceEsq:
                flag = 3

            # Se a distancia do sensor da esquerda for maior do que o da direita ira executar as
funções de flag = 4
            elif distanceDir < distanceEsq:
                flag = 4
```

```
        # Robô vai andar para a direita
        elif flag == 3:
            right_turn()
            print('Virar para a direita')

            # Vai andar para a direita ate que uma destas condições se verifique
            if (distanceDir <= 5 or distanceEsq <= 5) or (distanceDir >= 25 and distanceEsq >=
25):
                flag = 0

            else:
                if distanceDir == distanceEsq:
                    flag = 0

        # Robô vai andar para a esquerda
        elif flag == 4:
            left_turn()
            print('Virar para a esquerda')

            # Vai andar para a esquerda ate que uma destas condições se verifique
            if (distanceDir <= 5 or distanceEsq <= 5) or (distanceDir >= 25 and distanceEsq):
                flag = 0

            else:
                if distanceDir == distanceEsq:
                    flag = 0

        # Enquanto o robô não encontrar nenhum obstáculo vai simplesmente andar para a frente
        else:
            forward()
            print('Andar para a frente')
    # Caso encontre a mão durante a movimentação, ira parar
    else:
        stop()
        print('Mao encontrada, parado')

    # Verificar se foram mostrados 5 dedos durante 1 segundo para ligar o comando
    # Caso tenha acontecido, volta a ligar o comando e a movimentação é parada
    obstaculo = checkFifth()


# ---------- Funções para confirmar o comando introduzido pelo utilizador -----------


def checkFirst():
    # Timeout para certificar que o comando introduzido nao foi por engano
    # Apos ser introduzido o comando, se o mesmo ficar apresentado durante mais de 1 segundo,
ira ser aceite
    global dedos, comando, dedos, first, l
```

```
    if dedos == 5 and comando == 0:
        if first == 0:
            time.sleep(1)
            l = 0
            first = 1
        if l == 5 and first == 1:
            comando = 1
            dedos = 0
        else:
            dedos = 0
    else:
        first = 0

    return comando

def checkSecond():
    # Timeout para certificar que o comando introduzido não foi por engano
    # Apos ser introduzido o comando, se o mesmo ficar apresentado durante mais de 1 segundo,
ira ser aceite
    global dedos, seguimento, comando, dedos, second, l, start2

    if dedos == 2 and seguimento == 0:
        if second == 0:
            time.sleep(1)
            l = 0
            second = 1
        if l == 2 and second == 1:
            seguimento = 1
            start2 = 0
            comando = 0
            dedos = 0
        else:
            dedos = 0
    else:
        second = 0

    return seguimento

def checkThird():
    # Timeout para certificar que o comando introduzido não foi por engano
    # Apos ser introduzido o comando, se o mesmo ficar apresentado durante mais de 1 segundo,
ira ser aceite
    global dedos, obstaculo, comando, dedos, l, start2, third

    if dedos == 4 and obstaculo == 0:
        if third == 0:
            time.sleep(1)
            l = 0
            third = 1
```

```
        if l == 4 and third == 1:
            obstaculo = 1
            start2 = 0
            comando = 0
            dedos = 0
        else:
            dedos = 0
    else:
        third = 0

    return obstaculo

def checkFourth():
    # Timeout para certificar que o comando introduzido não foi por engano
    # Apos ser introduzido o comando, se o mesmo ficar apresentado durante mais de 1 segundo,
ira ser aceite
    global dedos, fourth, l, start3, seguimento

    if dedos == 5:
        if fourth == 0:
            time.sleep(1)
            l = 0
            fourth = 1
        if l == 5 and fourth == 1:
            dedos = 0
        else:
            stop()
            dedos = 0
            start3 = 0
            seguimento = 0
    else:
        fourth = 0
    return seguimento

def checkFifth():
    # Timeout para certificar que o comando introduzido não foi por engano
    # Apos ser introduzido o comando, se o mesmo ficar apresentado durante mais de 1 segundo,
ira ser aceite
    global dedos, fifth, l, start4, obstaculo

    if dedos == 5:
        if fifth == 0:
            time.sleep(1)
            l = 0
            fifth = 1
        if l == 5 and fifth == 1:
            dedos = 0
        else:
            stop()
```

```
            dedos = 0
            start = 0
            start4 = 0
            obstaculo = 0
    else:
        fifth = 0

    return obstaculo

while True:
    # Ler a imagem em real time da camera
    ret, img = cap.read()

    # Dar flip a imagem para não estar mirrored
    img = cv2.flip(img, 1)

    # Filtro de convolução 5x5
    kernel = np.ones((5, 5), np.uint8)

    # Converter imagem para HSV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Cores representam tons de cor da pele em HSV
    lowerBoundHSV = np.array([H_low, S_low, V_low], dtype=np.uint8)
    upperBoundHSV = np.array([H_high, S_high, V_high], dtype=np.uint8)

    # Extrair a uma máscara que representa os tons de pele da mao
    maskHSV = cv2.inRange(hsv, lowerBoundHSV, upperBoundHSV)

    # Fazer a dilatação da máscara obtida para remover ruido, utilizando o filtro "kernel" e 1
interação
    maskHSV = cv2.dilate(maskHSV, kernel, iterations = 1)
    # Desfocar a máscara resultante com um filtro 3x3
    result = cv2.GaussianBlur(maskHSV, (3, 3), 100)

    # Encontrar contornos da máscara, RETR_TREE -> Modo de recolha de contornos,
CHAIN_APPROX_SIMPLE -> Método de aproximação dos contornos.
    # RETR_TREE extrai todos os contornos e reconstrói uma hierarquia completa de contornos
alinhados.
    # CHAIN_APPROX_SIMPLE comprime segmentos horizontais, verticais e diagonais ao longo do
contorno e deixa apenas os pontos finais.
    contours, _ = cv2.findContours(result, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) > 0:
        # Encontrar contornos da mão (utilizando a area máxima) -> Encontrar o maior contorno
(que será a mão)
        # Lambda cria uma função em linha 'x', que neste caso é utilizada para calcular area do
contorno da mão
        cnt = max(contours, key=lambda x: cv2.contourArea(x))
```

```
        # Associa uma area do retângulo a mão do utilizador
        # Retira os valores de x, y, width e height desse retângulo
        x, y, w, h = cv2.boundingRect(cnt)
        # Aproxima os contornos um pouco
        # arcLength calcula um perímetro de contorno ou um comprimento de curva
        # approxPolyDp aproxima um polígono com outro polígono com menos vértices para que a
distancia entre eles seja menos ou igual a precisão especificada
        epsilon = 0.0005 * cv2.arcLength(cnt, True)
        approx = cv2.approxPolyDP(cnt, epsilon, True)

        # Faz um convex hull da mão
        # Um 'convex' é um objeto sem ângulos interiores maiores que 180º. 'Hull' representa o
exterior da forma do objeto
        # 'Convex Hull' representa o limite mínimo que pode fechar ou embrulhar completamento um
objeto (ou contorno do objeto)
        hull = cv2.convexHull(cnt)

        # areahull -> Define area do hull
        # areacnt -> Define area da mão
        areahull = cv2.contourArea(hull)
        areacnt = cv2.contourArea(cnt)

        # Encontra a percentagem de area que não é coberta pela mão no convex hull
        arearatio = ((areahull - areacnt) / areacnt) * 100

        # Encontra os defeitos no convex hull em respeito a mão
        # returnPoints=False é utilizado para encontrar os defeitos
        # convexityDefects encontra os defeitos de convexidade de um contorno
        hull = cv2.convexHull(approx, returnPoints=False)
        defects = cv2.convexityDefects(approx, hull)
        # l -> Representa o número de espaçamentos entre os dedos
        l = 0

        if start == 0:
            if comando == 0:
                print('\n-------- Comando desligado! Para ligar comando utilize 5 dedos. -------
-\n')
            start = 1

        # Para apresentar o valor da area da mão no ecrã
        areahull2 = int(areahull)
        areahull2 = str(areahull2)

        # Obter o número de dedos na mão do utilizador
        l, dedos = getDedos()

        # Verificar se o comando foi ligado, e não foi apresentado 5 dedos por engano
        comando = checkFirst()
```

```
# Verificar se algum dos modos foi selecionado
seguimento, obstaculo = comandoLigado()

# Seguimento da mão
if seguimento == 1:
    seguimentoMao()

# Evitar obstáculos
if obstaculo == 1:
    evitarObstaculos()

# Obter o número de frames por segundo atuais
cTime = time.time()
fps = 1/(cTime-pTime)
pTime = cTime
cv2.putText(img, f'FPS: {int(fps)}', (0, 250), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)

# ------------------------ Mostrar as janelas ------------------------
cv2.imshow('image', img)
cv2.imshow('Mascara', maskHSV)

# Premir ESC para desligar o código
k = cv2.waitKey(5) & 0xFF
if k == 27:
    cap.release()
    cv2.destroyAllWindows()
    break
```