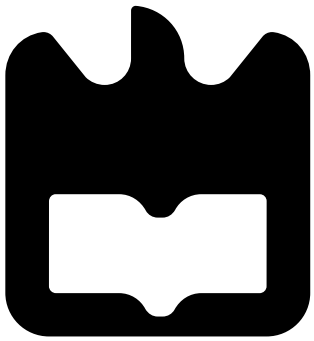
Universidade de Aveiro
2021

Hugo David
Pinto Santos

Sistema de visão e manipulação robótica para marcação a laser



Hugo David
Pinto Santos

Sistema de visão e manipulação robótica para marcação a laser

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Professor Pedro Nicolau Faria da Fonseca e Professor José Nuno Panelas Nunes Lau, Professores do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Professor Doutor António José Ribeiro Neves** Professor Auxiliar da Universidade de Aveiro
vogais / examiners committee	**Professor Doutor António Paulo Gomes Mendes Moreira** Professor Associado com agregação da Faculdade de Engenharia da Universidade do Porto (arguente principal)
	Professor Doutor Pedro Nicolau Faria da Fonseca Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Aproveito esta oportunidade para agradecer aos meus orientadores Professor Pedro Fonseca e Professor Nuno Lau pelo apoio e conhecimentos que me transmitiram ao longo da realização desta dissertação. Agradeço também ao colaborador Miguel Ângelo por possibilitar a realização deste projeto em parceria com a JPM *Industry*.

Ao João Gomes e Pedro Figueiredo devo também um agradecimento pelo companheirismo e interajuda, em longos dias de trabalho, o que contribuiu para lidar com as várias dificuldades que foram aparecendo.

Quero também deixar um especial agradecimento à Inês Lopes por ter acreditado no meu trabalho e por toda a força e incentivo que me foi dando ao longo destes anos, ajudando-me a lidar com os vários desafios que foram surgindo.

Por fim, quero agradecer à minha família, pai e mãe, por toda a confiança depositada em mim, além dos valores que me passaram como a persistência e empenho a aplicar em todas as fases da minha vida.

Aproveito ainda para agradecer a todos aqueles que contribuíram para que eu iniciasse e concluísse este percurso académico.

Palavras-chave Marcação a laser, Visão, Reconhecimento de objetos, OpenCV, Recursos locais, Robótica, ROS, Planeadores de trajetórias, Manipuladores

Resumo Nos últimos anos tem-se verificado um crescimento populacional que é acompanhado pela indústria através da automatização dos processos industriais. A robótica industrial e a visão computacional surgem como uma solução viável para dar resposta às exigências da produção massiva que nos dias de hoje se fazem sentir. Assim, conscientes dos progressos destas tecnologias, a JPM *Industry* e a Universidade de Aveiro lançaram o projeto Adaptmark, que tem como objetivo investigar e desenvolver um sistema robótico de marcação a laser de peças metálicas.

Assim, esta dissertação propõe um sistema dedicado à marcação a laser de peças planares de diversos formatos. Dividiu-se o sistema geral em dois subsistemas, um responsável pelo reconhecimento das peças e outro pela manipulação das mesmas, de forma a que estas sejam facultadas ao marcador a laser.

O desenvolvimento deste sistema é baseado na biblioteca *Open Source Computer Vision Library* (OpenCV) juntamente com o *Robot Operating System* (ROS), tirando partido das bibliotecas e pacotes já desenvolvidos nestas áreas de investigação como, por exemplo, o MoveIt!, dedicado ao planeamento de trajetórias, entre outras funcionalidades. Além disso, usou-se o Gazebo como simulador responsável por criar um mundo virtual semelhante ao caso real, onde se realizaram algumas validações.

Relativamente ao subsistema de visão, a solução foi apresentada, testada e validada. Este subsistema faz o reconhecimento de peças junto de uma base de dados, assim como estima a sua posição, orientação, dimensões e coordenadas de marcação a laser, onde apesar de existirem ainda erros associados este cumprem os requisitos definidos. Além disso, desenvolveram-se, testaram-se e validaram-se mecanismos que permitem o incremento da eficiência computacional deste subsistema. Em relação ao subsistema de manipulação, testou-se e validou-se, tanto em ambiente real como simulado, um conjunto de planeadores de trajetórias assim como a manipulação das peças. Quanto à marcação a laser propriamente dita, obtiveram-se resultados ajustados com os requisitos definidos tendo-se obtido um sistema final funcional. Todos os testes e resultados experimentais realizados são apresentados de forma detalhada nesta dissertação.

Keywords Laser Marking, Vision, Object Recognition, OpenCV, Local Resources, Robotics, ROS, Trajectory Planners, Manipulators

Abstract

In recent years there has been a population growth that is accompanied by the industry through the automation of industrial processes. Industrial robotics and computer vision emerge as a viable solution to respond to the demands of mass production that are felt nowadays. Thus, aware of the progress of these technologies, JPM Industry and University of Aveiro started the Adaptmark project, which aims to investigate and develop a robotic laser marking system for metal pieces.

This dissertation proposes a system dedicated to laser marking planar metal pieces of different shapes. The general system was divided into two subsystems, one responsible for recognizing the metal pices and the other for manipulating them, in order to make them available to the laser marker.

The development of the system is based on the *Open Source Computer Vision Library* (OpenCV) together with *Robot Operating System* (ROS), taking advantage of the libraries and packages already developed with research areas, such as MoveIt!, dedicated to the trajectories planning, among other features. Furthermore, the Gazebo simulator was used to create a virtual world similar to the real world, where some validations were performed.

Regarding the vision subsystem, the solution was presented, tested and validated, where it recognizes metal pieces in a database, as well as estimating their position, orientation, dimensions and laser marking coordinates, where despite the existence of associated errors, it accomplish the requirements defined. Furthermore, the mechanisms that allow the increment of the computacional efficiency of this subsystem were discovered, tested and validated. Regarding the test of the manipulation subsystem, a set of trajectory planners was validated, as well as the manipulation of the metal pieces. As for laser marking itself, results adjusted to the defined requirements were obtained, resulting in a functional final system. All tests and experimental results performed are presented in detail in this dissertation.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Siglas

AGAST *Adaptive and Generic Corner Detection Based on the Accelerated Segment Test.*

BiTRRT *Bi-directional Transition-based Rapidly-exploring Random.*

BRIEF *Binary Robust Independent Elementary Features.*

BRISK *Binary Robust Invariant Scalable Keypoints.*

CAD *Computer Aided Design.*

CCD *Charge Coupled Device.*

CHOMP *Covariant Hamiltonian Optimization for Motion Planning.*

CMOS *Complementary Metal Oxide Semiconductor.*

CPU *Central Processing Unit.*

DART *Dynamic Animation and Robots Toolkit.*

FAST *Features from Accelerated Segments Test.*

FED *Fast Explicit Diffusion.*

FK *Forward kinematics.*

FLANN *Fast Library for Approximate Nearest Neighbor.*

GIL *Global Interpreter Lock.*

GUI *Graphical user interface.*

HDR *High Dynamic Range.*

IDL *Interface Definition Language.*

IFR *International Federation of Robotics.*

IK *Inverse kinematics.*

IRIS *Intelligent Robotics and Systems.*

KDL *Kinematics and Dynamics Library.*

LSH *Locality Sensitive Hashing.*

ODE *Open Dynamics Engine.*

OGRE *Open Source 3D Graphics Engine.*

OMPL *Open Motion Planning Library.*

OpenCV *Open Source Computer Vision Library.*

OpenGL *Open Graphics Library.*

ORB *Oriented FAST and Rotated.*

PID *Proportional Integral Derivative .*

PLA *Poliatic Acid.*

PRM *Probabilistic Roadmap Method.*

RANSAC *Random Sample Consensus.*

RGB *Red-Green-Blue.*

ROS *Robot Operating System.*

RRT *Rapidly-exploring Random Trees.*

SBL *Single-query Bi-directional Lazy collision checking planner.*

SBPL *Search-Based Planning Library.*

SCARA *Selective Compliance Assembly Robot Arm.*

SDF *Simulation Description Format.*

SIFT *Scale Invariant Feature Transform.*

SPARS *SPArse Roadmap Spanner algorithm.*

SQP *Sequential Quadratic Programming.*

SRDF *Semantic Robot Description Format .*

STOMP *Stochastic Trajectory Optimization for Motion Planning.*

SURF *Speeded-Up Robust Features.*

URDF *Robot Description Format.*

XML *eXtensible Markup Language.*

Capítulo 1

Introdução

1.1 Enquadramento

Nos últimos anos tem-se verificado um crescimento populacional e estima-se que em 2100 se atinga a marca dos 10.9 biliões de pessoas [1]. Com este crescimento está inerentemente associado o aumento das exigências da sociedade e, por esse motivo é necessário que a indústria acompanhe esta evolução, sendo necessário recorrer à automatização dos processos industriais. Contudo, acompanhar estas evoluções industriais, dada a necessidade do aumento de produtividade é para algumas indústrias um grande desafio.

Associado ao objectivo da produção em massa, é também essencial nunca colocar em causa a garantia da qualidade do produto ou serviço prestado, indo sempre de encontro às exigências dos consumidores. Tendo como base estes objetivos, é também importante ressalvar que, através das tecnologias disponíveis, pode-se, assim, contribuir para a melhoria das condições de trabalho, assim como, da sua segurança.

A robótica industrial surge como uma solução viável para dar resposta às exigências da produção massiva que nos dias de hoje se fazem sentir. Segundo a *International Federation of Robotics* (IFR) estima-se que em 2020 foi estabelecido um recorde de 2.7 milhões de robôs industriais a operar em todo o mundo, registando assim um aumento de 12% relativamente ao ano anterior [2], evidenciando o elevado investimento da indústria em robôs.

No ramo da robótica industrial, uma das áreas que mais desafios tem apresentado são os sistemas de visão por computador que necessitam de acompanhar a elevada cadência da produção em série. Desta forma, os sistemas de visão terão de ser dinâmicos, flexíveis, rápidos, robustos e o mais autónomos possível de forma eliminar não só o tempo despendido na intervenção humana como o erro humano associado.

Conscientes dos progressos destas tecnologias, a JPM Industry e a Universidade de Aveiro lançaram o projeto Adaptmark, que tem como objetivo investigar e desenvolver um sistema robótico de marcação laser em peças planares de variados formatos.

1.2 Objetivos

Esta dissertação tem como objetivo investigar e desenvolver um sistema robótico de marcação a laser de peças constituídas, essencialmente, por aço inoxidável, caracterizado pela sua elevada cadência, autonomia de operação e adaptabilidade.

O sistema geral será dividido em dois subsistemas. O subsistema de visão é responsável

pelo reconhecimento de peças planares de diferentes formatos, estimando a sua posição, orientação e dimensões, assim como, as coordenadas da marcação a laser. O subsistema de manipulação, fundamentalmente constituído por um manipulador robótico e um marcador a laser, será responsável pela manipulação da peça, de forma a recolhê-la, facultá-la ao marcador laser e, por fim, colocá-la na zona de recolha.

Dado o objetivo apresentado, o desenvolvimento desta dissertação segmentado em dois subsistemas apresenta uma inegável dependência, entre eles, uma vez que, apenas será possível satisfazer os requisitos estabelecidos quando os dois subsistemas se encontrarem totalmente funcionais. A figura 1.1 ilustra a totalidade do sistema e como ele se apresenta.

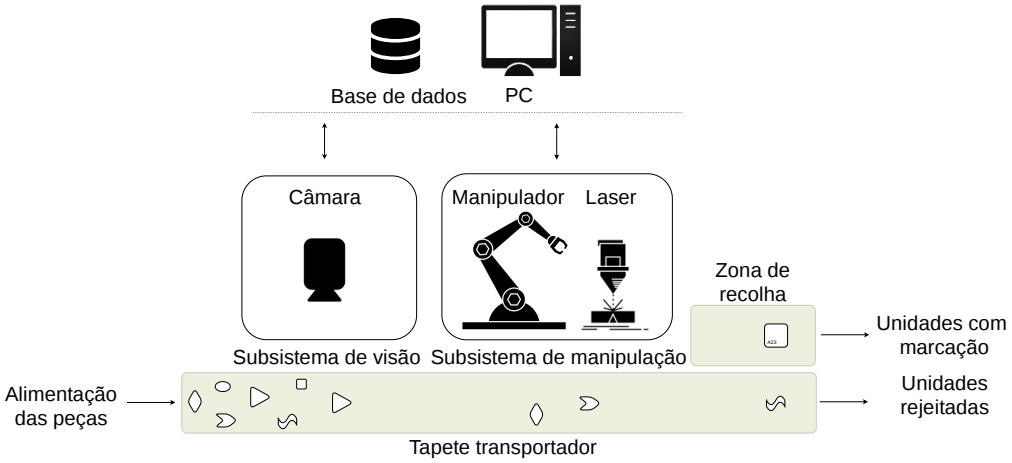


Figura 1.1 – Sistema geral de marcação a laser.

Com base no sistema geral delineado foram definidos, no início e durante a dissertação, alguns requisitos em conjunto com os orientadores desta dissertação e com a empresa JPM Industry, a saber:

- Usar uma câmara RGB localizada sobre o tapete transportador e orientada perpendicularmente sobre o mesmo;
- Identificar peças de diferentes formatos e com um espectro de cor bastante homogéneo. Além disso, estimar a sua localização, orientação e coordenadas da marcação a laser. Os erros máximos recomendados, sugeridos numa fase final da dissertação, devem rondar os 0.2 cm, 0.2° e 0.2 cm, respetivamente.
- Reconhecer peças com uma distância mínima entre elas de 1 cm;
- Base de dados estruturada com grupos de categorias de peças que devem rondar entre 50 a 400 peças. Numa base de dados constituída por, aproximadamente, 400 peças o tempo máximo proposto, numa fase final do trabalho, pode rondar um tempo de pesquisa de 4 segundos.
- De cada peça presente na base de dados faz parte uma imagem modelo previamente capturada e armazenada pelo operador, sendo que esta deve estar o mais próximo possível das condições de trabalho do sistema final. Por exemplo, a distância entre a câmara e a peça, deve ser mantida;

- Usar um manipulador e uma garra com especificações adequadas à situação problema, de modo, a manusear facilmente as peças. O tempo máximo associado a cada peça, relativamente à manipulação, deverá ser no máximo 8 segundos. O requisito temporal foi também sugerido numa fase final da dissertação.

- Fazer a marcação a laser numa posição específica da peça. A proposta inicial passar por o marcador a laser estar fixo, no entanto, é também proposto que se analise a viabilidade de ter o marcador a laser acoplado ao manipulador robótico;

O desenvolvimento deste sistema será baseado no *Robot Operating System* (ROS), juntamente com o *Open Source Computer Vision Library* (OpenCV), retirando partido das bibliotecas e pacotes já desenvolvidos nestas áreas de investigação.

1.3 Estrutura do documento

Este documento está dividido nos seguintes capítulos:

- Capítulo 2 - Reconhecimento de objetos: Este capítulo apresenta o estado de arte no âmbito do reconhecimento de objetos. Inicialmente fez-se uma pesquisa por alguns processos relacionados com a problemática desta dissertação e, posteriormente, realizou-se um estudo intensivo de extratores e descritores de recursos, assim como, dos respetivos métodos de correspondências.

- Capítulo 3 - Manipuladores robóticos: Neste capítulo é apresentado o estado de arte dos manipuladores robóticos e das garras correntemente usadas. Além disso, aborda-se domínios como a calibração, a cinemática e os planeadores de trajetórias. Por fim, é especificado o manipulador robótico usado nesta dissertação.

- Capítulo 4 - ROS: Neste capítulo é descrito de uma forma detalhada o que é o ROS e quais são as ferramentas de desenvolvimento disponibilizadas. Depois, ainda se aborda o propósito dos simuladores e quais as suas vantagens. Por último, retrata-se o MoveIt!, como um conjunto de pacotes dedicados, por exemplo, ao cálculo da cinemática inversa ou ao planeamento de trajetórias de manipuladores, entre outras funcionalidades.

- Capítulo 5 - Metodologia e resultados no reconhecimento de objetos: Neste capítulo é apresentado o trabalho prático realizado direcionado para o subsistema de reconhecimento de peças, começando por introduzir o sistema físico real em que este trabalho se enquadra. Ao longo do capítulo são comparadas as diferentes soluções abordadas e justificadas as decisões tomadas para os problemas encontrados no decorrer do trabalho. São também apresentadas as experiências e os resultados obtidos, seguido da discussão dos mesmos.

- Capítulo 6 - Metodologia e resultados na manipulação robótica: Neste capítulo é apresentado o trabalho prático realizado direcionado para o subsistema de manipulação robótica de peças, tanto em ambiente simulado como real. Ao longo do capítulo são comparadas as diferentes soluções abordadas e justificadas as decisões tomadas para os problemas encontrados no decorrer do trabalho. São também apresentadas as experiências e os resultados obtidos, seguido da discussão dos mesmos.

- Capítulo 7 - Conclusão e trabalho futuro: Para terminar, neste capítulo, é realizada uma análise geral sobre o trabalho desenvolvido, retirando conclusões sobre o resultado final obtido e o que poderá ser feito no futuro.

Capítulo 2

Reconhecimento de objetos

O reconhecimento de objetos é uma técnica de visão computacional que consiste em identificar objetos em imagens ou vídeos. Quando os humanos observam uma fotografia, é fácil de identificar pessoas, objetos, cenas e detalhes visuais. O objetivo destes algoritmos é atribuir ao computador capacidades que a visão humana possui, conferindo-lhe uma grande nível de compreensão da informação que a imagem disponibiliza. O reconhecimento de objetos abrange não só a identificação dos mesmos, como também, a estimativa da localização e orientação do objeto numa imagem.

Neste capítulo são abordadas diversas tecnologias e metodologias promissoras que têm ganho cada vez mais relevo nos últimos anos, na área na visão computacional com vista ao reconhecimento de objetos. Serão também apresentados alguns estudos de processos que se focam nesta problemática e seus respetivos resultados. O conhecimento de todos os conceitos seguidamente explorados são considerados fundamentais para a compreensão e validação da solução final.

2.1 Visão por computador

A visão computacional é a área científica responsável por tirar proveito da elevada capacidade de processamento dos computadores, aplicando-a na perceção de imagens ou vídeos. Com o avançar dos anos e a crescente potencialidade que se tem vindo a registar, o ramo da engenharia tem tentado tirar o máximo proveito destas tecnologias com vista a automatizar tarefas que o sistema visual humano realiza [3].

A visão por computador apenas é possível devido ao aprimorado processamento de imagem que nos dias de hoje é possível. Este processamento permite obter, não só, uma imagem digital com as características desejadas, como também a possibilidade de extrair informação útil que, por sua vez, será processada por um algoritmo com uma finalidade específica. Como espectável, a maioria dos algoritmos de processamento de imagem podem ser aplicados a um vídeo, uma vez que um vídeo nada mais é que um conjunto de imagens com uma determinada taxa de atualização.

As tarefas pelas quais a visão computacional pode ficar responsável são das mais diversas áreas como mapeamento 3D, deteção e reconhecimento de objetos, modulação de imagem 3D, entre outros. Este género de funções só é possível com a junção de vários conceitos de diversas áreas como geometria, física, estatística e teoria de aprendizagem [4].

2.2 Câmaras digitais

A imagem digital é uma representação de uma imagem bidimensional recorrendo a um conjunto de *bits* codificados de forma a permitir, por meios digitais, a sua representação, transferência, manipulação e armazenamento. Existem, essencialmente, dois tipos de imagens digitais, nomeadamente, o tipo matricial, também designada *raster* ou *bitmap*, e o tipo vetorial. A imagem de tipo matricial baseia-se numa matriz formada por um conjunto de pontos, chamados de píxeis, que se organizam por linhas e colunas, e que consoante os valores que lhes são atribuídos podem representar uma imagem que pode ser monocromática, em escala de cinza ou colorida. Usualmente os píxeis coloridos são formados pelo padrão *Red-Green-Blue* (RGB), que recorre a três números inteiros para representar o espectro de cores. As imagens adquiridas por câmaras são tipicamente do tipo matricial. A imagem vetorial usa primitivas geométricas como pontos, linhas, curvas e polígonos, as quais são baseadas em expressões matemáticas, de forma a representar imagens. Os desenhos técnicos, muito usados no ramo da engenharia, são um exemplo deste tipo de imagens vetoriais.

Pode-se encontrar uma grande variedade de tipos de câmaras para atender aos requisitos de uma aplicação específica. É possível encontrar câmaras com alta resolução e longo tempo de exposição para uso científico, câmaras de infravermelhas, por exemplo, para aplicações militares e de resgate, câmaras *High Dynamic Range* (HDR) para cenários de iluminação rígida, câmaras estéreo e 3D para extrair informações espaciais de um ambiente, entre outras. De uma forma breve as câmaras digitais podem-se dividir em 3 componentes:

- Lente - Responsável por fazer convergir os feixes de luz para o sensor da câmara, criando uma representação bidimensional do mundo;

- Sensor - Componente digital onde a imagem é projetada, encarregue de gerar uma matriz que avalia a quantidade de luz em cada posição (píxel). A qualidade das imagens depende essencialmente das especificações do sensor, tais como a tecnologia eletrónica *Charge Coupled Device* (CCD) e *Complementary Metal Oxide Semiconductor* (CMOS), dimensões do sensor, resolução (número de píxeis) ou velocidade (tempo necessário para avaliar a luz em cada píxel e armazená-la) [5].

- Filtro - Dispositivo colocado entre a lente e o sensor capaz de filtrar os comprimentos de onda da luz. Isso permite que cada píxel avalie a quantidade de luz em uma faixa específica de comprimentos de onda. No caso das câmaras RGB a faixa situa-se dentro da gama do visível e as cores escolhidas são vermelho, verde e azul (RGB), uma vez que depois de adquiridas podem ser conjugadas de forma a criar todo o espectro de cores possível.

2.3 Calibração da câmara

Uma câmara é encarregue de projetar pontos tridimensionais reais em pontos bidimensionais numa imagem e, independentemente da aplicação que se pretenda dar, é sempre necessário ter em conta a calibração da mesma. O processo de calibração consiste em estimar os parâmetros da lente e do sensor e também da relação entre eles. Uma vez calculados os parâmetros, é possível criar um modelo para determinar o valor de cada píxel a partir de uma representação 3D do ambiente visualizado pela câmara.

2.3.1 Modelo da câmara Pinhole

O modelo da câmara *pinhole* descreve a relação matemática entre as coordenadas de um ponto no espaço tridimensional e a sua projeção no plano de imagem de uma câmara ideal, pois a abertura da câmara é descrita como um ponto, e não é considerado uma lente de focagem da luz com as inerentes distorções. Alguns dos efeitos que o modelo da câmara não leva em consideração podem ser compensados, por exemplo, aplicando transformações adequadas nas coordenadas da imagem.

A figura 2.1 ilustra o modelo descrito com o distância focal (f), a distância da câmara ao objeto (Z), a dimensão do objeto (X) e a dimensão do objeto projetado no plano imagem (x) e dada a trigonometria existente pode-se considerar a seguinte relação:

$$-\frac{x}{f} = \frac{X}{Z} \tag{2.1}$$

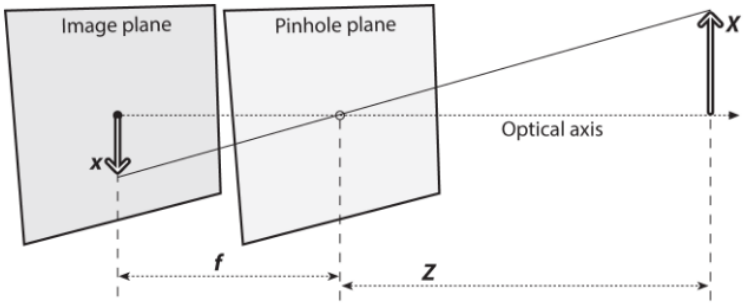


Figura 2.1 – Ilustração do modelo da câmara *Pinhole* [6].

2.3.2 Parâmetros de calibração

Os parâmetros de calibração de câmara são essencialmente constituídos pelos parâmetros intrínsecos e extrínsecos [7]:

- Parâmetros intrínsecos - são responsáveis pela formação da imagem e incluem duas características físicas da própria câmara, a saber, a distância focal e a posição do sensor. A distância focal diz respeito à distância entre a lente e o plano de projeção, logo, considerando o tamanho do sensor fixo, quanto menor o distância focal maior será o ângulo de abertura do sensor. A posição do sensor também faz parte deste parâmetros devido à rotação ou desalinhamento que possa existir entre o centro óptico, ponto no eixo da lente que quando atravessado por qualquer feixe de luz não sofre qualquer desvio, e o centro do sensor. O modelo ideal da câmara não tem em consideração a distorção da lente, logo deve-se incluir tanto a distorção radial como a tangencial, sob a forma de coeficientes de distorção, a fim de obter uma representação linear do cenário tridimensional.

- Parâmetros extrínsecos - referem-se à rotação e translação da câmara relativamente a um sistema de coordenadas global. A calibração deste tipo de parâmetros será mais desenvolvida no capítulo 3;

Posteriormente à calibração da câmara, é possível avaliar a precisão dos parâmetros estimados com base no erro de reprojeção. Este é um erro geométrico que corresponde à distância entre um ponto projetado e observado na imagem, como se pode observar na figura 2.2 [8]. Ou seja, este erro permite quantificar o quão precisa uma estimativa de um ponto 3D recria a projeção real do ponto.

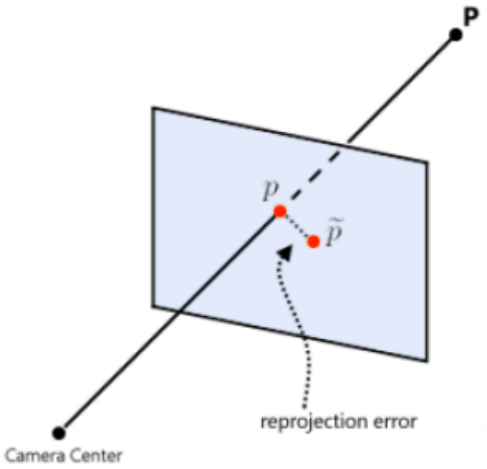


Figura 2.2 – O erro de reprojeção é a distância entre o ponto da imagem projetada p o ponto da imagem observada $\tilde{p}$ [8].

De modo a estimar os parâmetros, são necessários pontos tridimensionais em coordenadas reais e o seus pontos bidimensionais correspondentes. Por conveniência, de forma a facilitar os cálculos, as coordenadas são convertidas de coordenadas cartesianas para homogéneas. A seguinte equação relaciona as coordenadas de um ponto 3D (X,Y,Z) com as coordenadas na imagem projeção desse ponto (x,y), com base nas coordenadas homogéneas:

$$x = \frac{u}{w} \qquad y = \frac{v}{w} \tag{2.2}$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.3}$$

onde P é matriz projeção, que é constituída por duas matrizes, a matriz intrínseca (K) e a matriz extrínseca $[R|t]$:

$$\underbrace{P}_{\text{Matriz projeção}} = \underbrace{K}_{\text{Matriz intrínseca}} \times \underbrace{[R|t]}_{\text{Matriz extrínseca}} \tag{2.4}$$

A matriz intrínseca é dada por:

$$K = \begin{bmatrix} f_x & \lambda & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.5}$$

onde f_x e f_y são a distância focal nos eixos x e y, c_x e c_y são o centro ótico da imagem plano em x e y, e λ é o coeficiente de enviesamento entre os eixos.

Resolvendo a equação 2.3 podem-se desenvolver as seguintes expressões:

$$\begin{cases} u = f_x \cdot X + c_x \cdot Z \\ v = f_y \cdot Y + c_y \cdot Z \\ w = Z \end{cases} \tag{2.6}$$

Convertendo as coordenadas homogéneas novamente em coordenadas cartesiana e sabendo que w = Z, então:

$$\begin{cases} x = \frac{f_x \cdot X}{Z} + c_x \\ y = \frac{f_y \cdot Y}{Z} + c_y \end{cases} \tag{2.7}$$

Assumindo que o referencial do centro ótico está centrado na origem dos eixos, pode-se deduzir as seguintes expressões:

$$\begin{cases} x = \frac{f_x \cdot X}{Z} \\ y = \frac{f_y \cdot Y}{Z} \end{cases} \tag{2.8}$$

onde se pode verificar que a equação 2.1 do modelo da câmara *Pinhole* são conforme estas expressões deduzidas.

Distorção na calibração

A matriz da projeção não tem em consideração a distorção da lente, dado que a câmara *pinhole* ideal não tem lente, logo não tem distorções associadas. De forma a representar com precisão o modelo da câmara real, o modelo usado deve incluir tanto a distorção radial como a tangencial, representadas nas figuras 2.3 e 2.4, respetivamente [7]. Estas distorções devem ser compensadas, a fim de se obter uma representação sem distorções do cenário tridimensional.

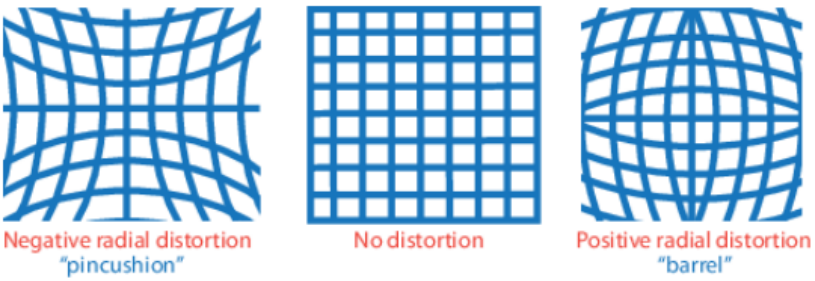


Figura 2.3 – Representação das distorções radiais *Pincushion* e *Barrel* [7].

A distorção radial ocorre quando os raios de luz se curvam mais conforme mais distantes se encontram do centro ótico. Os coeficientes de distorção radial modelam este tipo de distorção e os pontos distorcidos (x_{dist_radial}, y_{dist_radial}) são dados pelas seguintes expressões:

$$x_{dist\_radial} = x(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6), \quad r^2 = x^2 + y^2$$
$$y_{dist\_radial} = y(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6), \quad r^2 = x^2 + y^2$$

onde x, y são as coordenadas dos pontos sem distorção, k_1, k_2, k_3 são os coeficientes de distorção radial da lente.

A distorção tangencial ocorre quando a lente e o plano da imagem não são totalmente paralelos (Figura 2.4). Os coeficientes de distorção tangencial que modelam este tipo de distorção e os pontos distorcidos ($x_{dist\_tangencial}$, $y_{dist\_tangencial}$) são dados pelas seguintes expressões:

$$x_{dist\_tangencial} = x + [2 * p_1 * x * x + p_2 * (r^2 + 2 * x^2)], \quad r^2 = x^2 + y^2$$

$$y_{dist\_tangencial} = y + [p_1 * (r^2 + 2 * y^2) + 2 * p_2 * x * y], \quad r^2 = x^2 + y^2$$

onde x e y são as coordenadas dos pontos sem distorção, p_1 e p_2 são os coeficientes de distorção tangencial da lente.

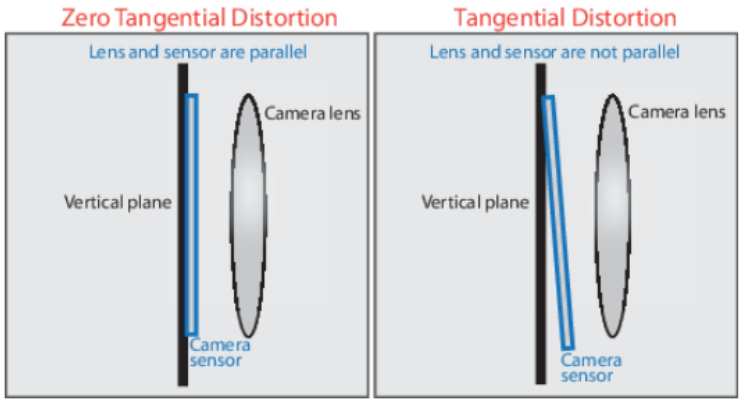


Figura 2.4 – Representação da distorção tangencial [7].

O OpenCV possui métodos de calibração da câmara a fim de calcular os parâmetros intrínsecos, reduzindo o efeito de distorção causado pelas lentes. Usando um conjunto de imagens de calibração com um tabuleiro de xadrez, que deve ser capturado em diversos ângulos, é calculada a matriz intrínseca (equação 2.5) e a matriz dos coeficientes da distorção (D):

$$D = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{2.9}$$

O tabuleiro de xadrez é bastante usado para esta finalidade pois o seu padrão quadriculado é distinto e fácil de detetar numa imagem, uma vez que os cantos dos quadrados estão bem definidos o que permite localizá-los de forma robusta no padrão xadrez.

2.4 OpenCV

O *Open Source Computer Vision Library* (OpenCV) é uma biblioteca multiplataforma, totalmente livre tanto para uso académico como comercial, que visa acelerar o desenvolvimento de aplicações na área de visão computacional [9].

A estrutura do OpenCV pode ser divida em módulos como, por exemplo, processamento de imagem e vídeo, aplicações de tempo real, estruturas de dados, álgebra linear, interface gráfica do utilizador (GUI) e de controle de periféricos externos como rato e teclado, além dos milhares de algoritmos disponíveis. É uma biblioteca centrada no desenvolvimento de algoritmos nas vastas áreas da visão computacional, tais como sistemas de reconhecimento

facial, identificação de objetos, reconhecimento de movimentos, robótica móvel, realidade aumentada, reconstrução 3D, entre outros [9].

Inicialmente toda a biblioteca foi desenvolvida nas linguagem de programação C e C++, mas atualmente dá suporte a linguagens como Java, Python e Visual Basic. A versão 1.0 foi lançada em 2006 e nos dias de hoje já se encontra na versão 4.5.2.

2.5 Processamento de imagem de baixo nível

O processamento de imagem de baixo nível consiste num diversificado conjunto de operações que podem ser realizadas ao nível dos píxeis de forma a extrair informações úteis para um processamento de alto nível posterior. Nesta secção, as operações mais importantes para esta dissertação serão detalhadas.

2.5.1 Filtros baseados na convolução e correlação

Tal como um sinal analógico ou digital, as imagens podem ser filtradas por uma grande variedade de filtros, entre os quais filtros passa-baixo, dedicados a remoção do ruído, filtros passa-alto, de forma a realçar os limites, entre outros. Este tipo de filtros são baseados na convolução ou correlação de uma imagem com uma matriz normalmente denominada como *kernel*, que pode assumir não só várias dimensões como também diversas combinações, podendo ser lineares ou não. Um *kernel* linear significa que o ponto com as coordenadas (x,y) resultado da aplicação do mesmo pode ser expresso como uma soma ponderada dos pontos que o rodeiam incluindo ele próprio [6]. Tanto a filtragem baseada na convolução como na correlação são caracterizadas como lineares, no entanto, o que as distingue é a configuração do *kernel*, uma vez que, no caso da convolução, o *kernel* é rodado 180°, o que faz com que os resultados obtidos possam ser bastante diferentes.

Desta forma diversos filtros podem ser facilmente desenvolvidos e aplicados usando um simples *kernel* como, por exemplo, *kernels* dedicados a fazer o desfoque (*blur*) da imagem, outros destinados a realçar determinadas características da imagem. Uma outra aplicação possível passa por fazer correspondência entre duas imagens distintas mas que se podem correlacionar, logo a correlação entre duas imagens é calculada, sendo que uma delas é representada sobre a forma de um *kernel*.

O OpenCV disponibiliza funções que recorrem a este tipo de filtros sendo possível aplicar a diversas finalidades. As funções *filter2D*, *matchTemplate* e *blur* são exemplos disso mesmo.

2.5.2 Deteção de limites

Dependendo da aplicação, o formato de cada objeto pode ser é um dos fatores mais relevantes na caracterização do objeto e, por isso, normalmente, qualquer tipo de atributos adicionais são dispensáveis, logo é necessário simplificar e reduzir a complexidade das imagens antes da aplicação de qualquer método de correspondência de imagens. Assim a deteção de limites da imagem reduz significativamente a quantidade de dados e filtra informações inúteis, preservando as propriedades estruturais importantes de uma imagem [10].

Vários algoritmos tem sido propostos nos últimos anos com vista a determinar os limites de um objeto numa imagem digital como, por exemplo, o *Sobel*, *Roberts Cross*, *Laplacian*, *Canny*, entre outros. Segundo um artigo dedicados à comparação deste algoritmos [10], pode-se afirmar que o algoritmo *Canny Edge*, apesar de ser computacionalmente mais exigente, é

o que melhor desempenho apresenta nos mais diversos cenários, mesmo em condições com bastante ruído e, por essas razões, é o algoritmo de deteção de limites que será apresentado e testado.

Canny Edge

Após uma indispensável pré-filtragem do ruído da imagem usando um filtro gaussiano, o processo inicia-se pela determinação do gradiente de intensidade da imagem, que é conseguida com um filtro *sobel* [10]. O operador *sobel* consiste num par de *kernels*, de dimensão 3x3, que permite estimar o gradiente na direção x e y, como é ilustrado nas seguintes equações, onde I refere-se à imagem e * representa a convolução bidimensional:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I \tag{2.10}$$

De seguida, tendo a primeira derivada da direção horizontal (G_x) e vertical (G_y), usando a distância euclidiana, também conhecida por norma L2, pode-se obter a amplitude do gradiente da borda e a respetiva direção com as seguintes expressões:

$$gradiente(x,y)_{L2} = \sqrt{G_x^2 + G_y^2} = \sqrt{\left(\frac{dI}{dx}\right)^2 + \left(\frac{dI}{dy}\right)^2} \tag{2.11}$$

$$\hat{A}ngulo(\theta) = tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.12}$$

A forma mais usual é calcular o gradiente usando a distância de Manhattan, também conhecida por norma L1, pois é um processo mais rápido. Neste caso o cálculo do gradiente é dado pela seguinte expressão:

$$gradiente(x,y)_{L1} = \left|\frac{dI}{dx}\right| + \left|\frac{dI}{dy}\right| \tag{2.13}$$

Depois de calculado o gradiente, o processo consiste em analisar todos os píxeis e os que corresponderem a máximos locais são então candidatos a arestas. Este processo é feito píxel a píxel de forma a determinar os que melhor caracterizam os contornos da imagem e, de seguida, são submetidos a dois algoritmos, um de supressão não máxima, outro baseado no limite de histerese dos píxeis.

O método de supressão não máxima consiste na eliminação de píxeis cujos valores não são máximos locais na direção perpendicular à borda, ou seja, no sentido do gradiente. Assim, fica-se apenas com os píxeis das bordas que correspondem aos máximos locais. Relativamente ao segundo algoritmo, definindo-se um limite inferior e superior, os píxeis são sujeitos a um criterioso algoritmo, que se baseia no seguinte raciocínio: se o gradiente associado ao píxel tiver um valor maior do que o limite superior, ele será aceite como um píxel pertencente à borda, no entanto se estiver abaixo do limite inferior, então ele simplesmente é rejeitado. No entanto, se o gradiente associado a esse píxel estiver entre os limites, ele será aceite somente se nos seus píxeis vizinhos existir um píxel que já pertença à borda [10]. A título de exemplo, na figura 2.5b pode-se observar o resultado final da aplicação deste algoritmo na figura 2.5a.

(a) (b)

Figura 2.5 – Resultado do *Canny Edge* num objeto exemplar com que se pretende trabalhar.

2.5.3 Operações Morfológicas

A erosão e a dilatação são duas operações morfológicas de processamento de imagem, ou seja, o processamento é feito píxel a píxel. Estas operações são destinadas a ser aplicáveis tanto a imagens binárias como a imagens em escala de cinzentos.

Os dois métodos são definidos pelo *kernel* que percorre toda a imagem. O *kernel* pode assumir diversas formas, sendo a mais usual uma matriz quadrada com apenas '1'.

Na erosão, o píxel da imagem será considerado '1' apenas se todos os píxeis sob o *kernel* forem '1'. Se não, o píxel é considerado '0'. Desta forma é possível, por exemplo, eliminar pequenos ruídos na imagem produzidas pelo *Canny Edge*. Quanto à dilatação, segue exatamente o mesmo raciocínio, mas no sentido inverso. Ou seja, o píxel é considerado '1' se pelo menos um píxel sob o *kernel* for '1'. Este processo é útil como forma de garantir que os limites dos objetos sejam fechados, o que reduz as possíveis falhas.

2.5.4 Contornos

O reconhecimento de objetos é uma das áreas mais exploradas na visão computacional e a análise de contornos é particularmente importante nesse contexto. Apesar dos algoritmos de detetor de limites, como o *Canny Edge*, possam ser usados para encontrar os píxeis dos limites que fragmentam os diferentes segmentos de uma imagem, eles não dão nenhuma indicação sobre estas margens como entidades em si mesmas. Assim, os contornos não só identificam os limites de um objeto numa imagem, como também permitem a caracterização de todos os limites identificados, atribuindo-lhes uma identidade específica sob a forma de uma hierarquia.

O OpenCV apresenta uma função designada *findContours* que faz exatamente esse processo. Esta função cria uma lista de pontos bidimensionais, da biblioteca *Numpy*, que representam os limites de um objeto numa imagem. Estes pontos são caracterizados pelas respetivas coordenadas (x,y) que resultam na caracterização de uma determinada assinatura de uma imagem binária. A função permite ainda extrair um conjunto diversificado de propriedades que caracterizam a imagem binária em análise. Algumas das propriedades mais usadas são apresentadas de seguida.

Propriedades

Os momentos representam características de alto nível que caracterizam os contornos. Assim os momentos $m_{p,q}$ são definidos como a soma da intensidade I dos N píxeis da imagem, onde, por sua vez, o valor de cada píxel (x,y) é multiplicado pelo fator $x^p y^p$, como representado na seguinte expressão [11]:

$$m_{p,q} = \sum_{i=1}^{N} x_i^p y_i^q I(x_i, y_i) \tag{2.14}$$

Tratando-se de uma imagem binária, o m_{00} consiste simplesmente no número de píxeis diferentes de zero. Assim, da divisão dos momentos m_{10} e m_{01} por m_{00} resulta, na prática, no valor médio das coordenadas dos píxeis selecionados, ou seja, as coordenadas do centróide. Assim o centróide de intensidade pode ser dado por [11]:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{2.15}$$

Em imagens binárias, a área também pode ser calculada usando os momentos pois m_{00} retorna o número de píxeis iguais a 1, e esta é métrica usada.

Duas outras características muito usadas são *Rotated Rectangle* e o *Ellipse Fitting* e podem ser usadas com a mesma finalidade. Por um lado, no *Rotated Rectangle* o objetivo passa por retornar o retângulo mais pequeno em que o contorno fica circunscrito e, através do comprimento, largura e respetivo ângulo em relação ao eixo vertical, será possível determinar a orientação do contorno. Por outro lado, *fitting* de uma elipse segue um raciocínio ligeiramente diferente em que não retorna propriamente a elipse com a menor área em que o contorno fica circunscrito, mas sim a elipse que dá a melhor aproximação do conjunto de pontos que caracteriza os contornos. Mais uma vez, será possível obter uma estimativa da orientação do contorno.

2.6 Processos de reconhecimento de objetos

O desafio de reconhecimento de objetos é um problema já muito aprofundado e combina uma série de algoritmos que apresentam as suas vantagens e desvantagens consoante a aplicação a que estão sujeitos. Nesta secção, são abordados alguns processos que se enquadram no âmbito desta dissertação no que diz respeito ao reconhecimento de objetos.

2.6.1 Reconhecimento de objetos baseado em histogramas de cor

O reconhecimento de objetos baseado em histogramas de cor foi, na área da visão computacional, das primeiras abordagens que se realizaram e, atualmente, é ainda muito usada em aplicações específicas, dadas as suas limitações. Esta técnica baseia-se em representar o nível de cor, para cada canal de cor, de forma individual, sob a forma de um histograma, como ilustrado na figura 2.6. O método de correspondência baseia-se na comparação das intensidades de cada uma das componentes de dois histogramas que representam cenários diferentes. Logo histogramas diferentes do mesmo cenário representarão histogramas muito semelhantes em cada uma das componentes.

Estudou-se um artigo que recorre a este método e apresenta um algoritmo de deteção rápida de objetos com base em histogramas de cores e padrões binários locais. O método

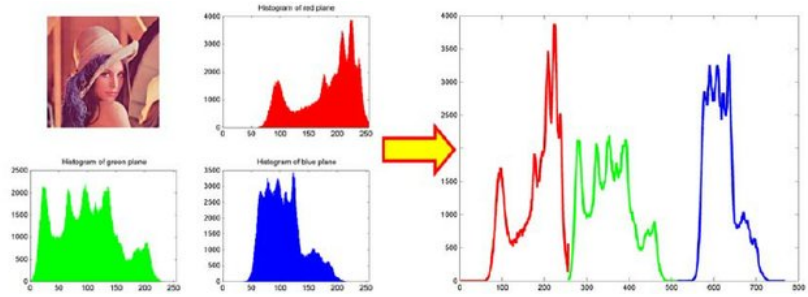


Figura 2.6 – Histograma de cor RGB para uma imagem exemplo [12].

proposto, invariante a transformações de escala e rotações, consiste em duas etapas: usa os histogramas de cores para determinar os candidatos a objetos alvo e, posteriormente, faz a deteção mais precisa usando os histogramas baseados nos padrões binários locais detetados [13]. O processo obteve resultados bastante satisfatórios do ponto de vista de uma aplicação robótica, de elevada cadência, com tempos inferiores a 0.5 segundos. No entanto, este procedimento fornece apenas uma caracterização muito debilitada de uma imagem, uma vez que imagens com histogramas semelhantes podem resultar de cenários completamente distintos.

Esta abordagem, apesar de no artigo desenvolvido apresentar resultados adequados, devido às limitações apresentadas não será uma solução para a situação problema desta dissertação. Isto porque os espaços de cores das imagens resultantes apresentariam histogramas praticamente iguais, dada a homogeneidade das peças, no que diz respeito às cores constituintes.

2.6.2 Reconhecimento de objetos baseado numa imagem modelo

O método de correspondência baseado num modelo é um dos principais algoritmos de pesquisa numa imagem digital, apresentando uma grande utilidade para diversas aplicações de visão computacional.

A correspondência baseia-se num processo de pesquisa elementar e iterativo onde, através de uma imagem modelo, é realizada a pesquisa sobre a imagem original. Este processo de pesquisa é efetuado, normalmente, píxel a píxel, de forma a que a cada iteração seja possível comparar a correspondência entre a imagem modelo e a imagem original [14], como se pode verificar na figura 2.7. O cálculo da correspondência pode ser feito com recurso a diversos métodos como, por exemplo, o método das diferenças quadradas, o cálculo da correlação, ou o cálculo do coeficiente de correlação [6].

Estudou-se um artigo que propõe um algoritmo chamado *Fast-Match*, que é um método rápido para detetar correspondências aproximadas considerando transformações afins 2D [15]. Baseia-se na métrica soma de diferenças absolutas para calcular o erro associado e para cada potencial transformação calcula esse erro usando um algoritmo linear que examina aleatoriamente apenas um conjunto limitado de píxeis. Este algoritmo destaca-se dos métodos clássicos de correspondências baseados em modelo pois difere na forma como descarta informações irrelevantes, não fazendo uma análise intensiva e desnecessária, além de que é invariante a transformações afins. Os testes foram realizados usando o *Zurich Building Database* [15] com uma resolução das imagens de 640x480 píxeis, como ilustrado na figura 2.8, e num computador com um CPU da Intel i7 de 2.7 MHz.

Esta abordagem apresenta tempos cada vez menores quanto mais a imagem modelo se

Figura 2.7 – Processo de correspondências baseado numa imagem modelo que percorre toda a imagem original [6].

Figura 2.8 – Ilustração dos resultados obtidos na deteção de objetos [15].

aproxima das dimensões da imagem pesquisada. Com a imagem modelo, apenas 10% mais pequena, os tempos de computação médios são de cerca de 2.5 segundos. Assim, este método apresenta resultados, a nível temporal, desajustados para aplicações robóticas com requisitos de temporais bastante restritos.

2.6.3 Reconhecimento de objetos baseado em descritores locais

O reconhecimento de objetos baseado em descritores locais, visa detectar e descrever regiões de interesse numa imagem, mais conhecidos por pontos-chave. Estas características, que devem ser únicas, são consideradas apropriadas para a correspondência de objetos em diferentes cenários [16].

Desta forma, analisou-se um artigo que consiste em fazer o reconhecimento de objetos procurando usar recursos invariantes locais que, posteriormente, serão manipulados por um robô de serviço (Figura 2.9). Diferentes sistemas de conhecimento de objetos foram construídos usando diferentes combinações de extratores e descritores locais e usando diversos estágios de verificação de transformações geométricas [17].

Este artigo foi desenvolvido segundo alguns pressupostos, uma vez que o objetivo passava por fazer reconhecimento de objetos para aplicações robóticas e, por isso, teve-se em consideração algoritmos exequíveis, para aplicações de elevada cadência, com recursos com-

putacionais limitados, além das limitações impostas pelo próprio robô e câmara como, por exemplo, os graus de liberdade do robô e a resolução do sensor.

Análises em termos de precisão, robustez e eficiência foram desenvolvidas e, com os vários testes realizados, foi possível retirar importantes conclusões como, por exemplo, que os sistemas de reconhecimento de objetos com melhor desempenho são *Oriented FAST and Rotated* (ORB) e *Scale Invariant Feature Transform* (SIFT). Os sistemas baseados em ORB são mais rápidos, enquanto os SIFT são mais precisos. Além disso, o desempenho dos sistemas de reconhecimento de objetos usando recursos invariantes locais depende fortemente dos métodos de verificação de transformação geométrica usados. Todos os conceitos e métodos referidos serão tratados em futuras secções de uma forma detalhada.

Esta metodologia apresenta uma boa alternativa aos métodos anteriormente apresentados. Dá boas indicações de cumprimento de requisitos temporais exigentes, no âmbito da robótica, além de que é robusto na deteção e estimativa de posição do objetos. Uma abordagem semelhante será abordada nesta dissertação.

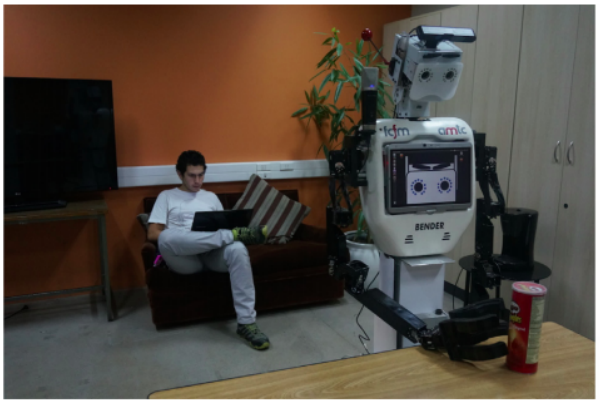


Figura 2.9 – Robô Bender observando um objeto sobre uma mesa [17].

2.7 Extratores e descritores de recursos locais

Na visão computacional existem dois tipos de recursos de imagem que podem ser extraídos, a saber, os recursos globais e os recursos locais. Os recursos globais são, por exemplo, a cor e a textura, pois pretendem descrever uma imagem como um todo e podem ser interpretados como uma propriedade da imagem que envolve todos os píxeis. Os recursos locais visam detectar e descrever pontos-chave ou regiões de interesse numa imagem. Assim estas características são consideradas adequadas para a correspondência de objetos em diferentes cenários [16].

Uma imagem digital que representa um determinado objeto ou cenário pode ser caracterizada por um conjunto de pontos característicos, usualmente conhecidos por pontos-chave, recursos ou *features*, criteriosamente escolhidos para que sejam únicos.

Após a deteção de um ponto-chave de uma imagem num local p(x, y), tendo em conta a escala s e orientação θ, o descritor local faz uma representação da vizinhança local do ponto p. Assim, essa representação tem como objetivo tornar-se invariável às transformações locais da imagem, atribuindo-lhe uma identidade distinta que permite o reconhecimento efetivo para futuras correspondências como ilustrado na figura 2.10 [18] [16]. De seguida são apresentados alguns algoritmos dedicados à extração e descrição desses pontos-chave.

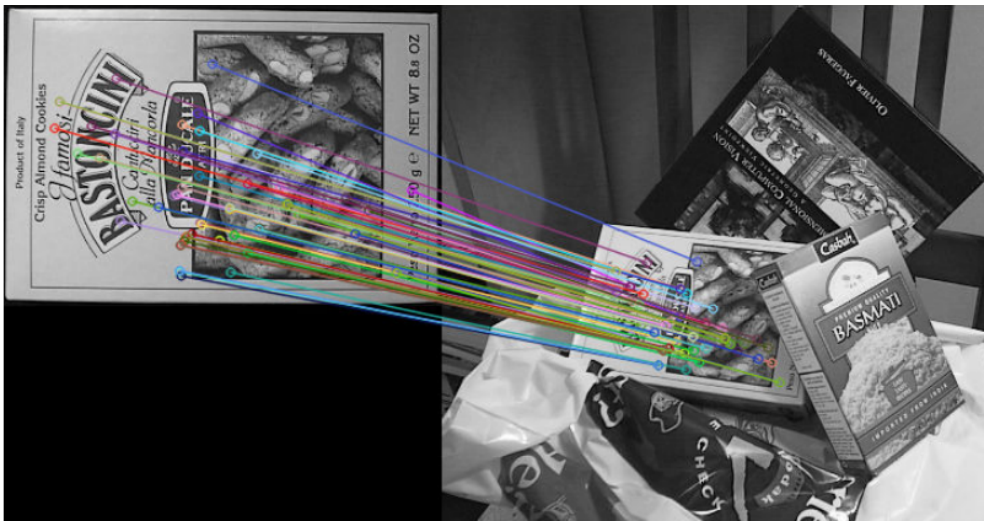


Figura 2.10 – Exemplo da aplicação de um algoritmo de extração e descrição de *features* e respetivas correspondências [9].

2.7.1 SIFT

O método *Scale Invariant Feature Transform (SIFT)* [19] é um extrator e descritor de características locais que são invariantes à escala, como o próprio nome indica, mas também à rotação, luminosidade, ruído, deformações, mudanças de contraste, entre outras deformações. De uma forma concisa este algoritmo pode ser agrupado em quatro estágios com funções específicas [6], [19], [20]:

- Deteção espaço-escala - A identificação dos pontos-chave baseia-se na construção do género de uma pirâmide multi-escala, reduzindo a resolução de uma imagem através da aplicação de um filtro gaussiano, píxel a píxel, seguido do cálculo da diferença gaussiana (DoG). A diferença gaussiana é um algoritmo de aprimoramento de uma imagem na escala de cinza através de uma subtração entre uma imagem com pouca suavização e a mesma imagem com um nível de suavização superior. As imagens são obtidas através da convolução entre a imagem original, em tons de cinza, e um *kernel* gaussiano com diferentes desvios-padrão, de forma a obter distintos níveis de suavização. O resultado desta convolução preserva os limites, entre outros detalhes, duma imagem digital. Com base neste processo são construídos os níveis sucessivos da pirâmide, como se pode verificar na figura 2.11. Com vista à deteção do ponto-chave propriamente dito, o algoritmo analisa os seus píxeis vizinhos residentes no mesmo nível da pirâmide, assim como os dos níveis adjacentes. Se um píxel tem um valor mais alto na diferença gaussiana do que todos os restantes vizinhos, então ele é considerado um candidato a ponto-chave.

- Localização de pontos-chave - Uma vez que um conjunto de recursos é encontrado, o algoritmo testa cada recurso para determinar a sua qualidade como um recurso e para refinar a estimativa da sua localização. Esse processo baseia-se numa interpolação usando a expansão quadrática de Taylor onde se calcula um deslocamento a ser adicionado à localização do ponto-chave. Finalizado este processo, os pontos candidatos a pontos-chave são invariantes à escala e localização.

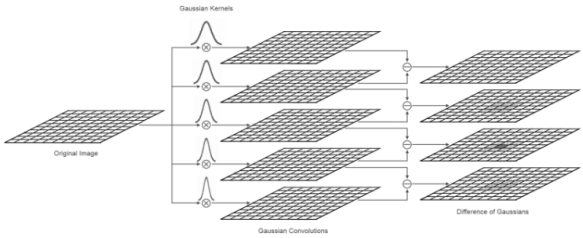


Figura 2.11 – Deteção espaço-escala com base no cálculo das diferenças gaussianas [6].

- Atribuição de orientação - Uma vez localizados os pontos-chave, o objetivo passa por torná-los invariantes à rotação logo, para cada candidato, é lhe atribuída uma magnitude e uma orientação baseada essencialmente na comparação das derivadas direcionais sobre os pontos em redor dos mesmos. Seleciona-se toda a vizinhança em redor e constrói-se, assim, um histograma de orientação para todo os píxeis vizinhos (Figura 2.12). Os picos resultantes do histograma são as direções dominantes na imagem. Dessa forma, os recursos SIFT não são apenas invariantes à escala e localização, mas também invariantes à orientação.

- Descritor de ponto-chave - Uma vez obtidas as informações pretendidas, é necessário organizá-las de forma a que se possa obter correspondências. Para fazer isso, a vizinhança de 16x16 regiões em redor de cada ponto-chave é agrupada em blocos e para cada um desses blocos calcula-se o histograma de orientação, sendo cada um destes agrupado num padrão 4x4 de sub-regiões em torno do ponto-chave. Tendo em conta que cada histograma tem 8 entradas por região e que no total existem 16 regiões, então o vetor descritor formado tem 128 elementos para cada ponto-chave. Todo o processo encontra-se ilustrado na figura 2.12.

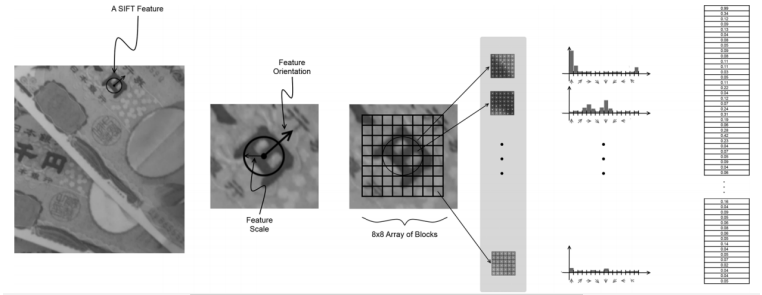


Figura 2.12 – Processo de formação de vetor descritor [6].

Contudo, a construção do vetor de recursos SIFT é complexa e apresenta uma grande exigência a nível computacional afetando a eficácia em termos temporais, o que pode comprometer aplicações com exigências de tempo restritas.

2.7.2 SURF

O método *Speeded-Up Robust Features (SURF)* [21] surge com o principal objetivo de desenvolver um algoritmo capaz de detetar e descrever *features* suprimindo a maior limitação do

método SIFT que reside na seu desempenho temporal. De forma a ultrapassar essa limitação sem comprometer a precisão, o detector é baseado no determinante da matriz Hessiana, explorando as imagens integrais e reduzindo, assim, os tempos de computação [21]. Cada entrada de uma imagem integral $I_{\sum}(\chi)$ numa dada localização (x,y) é dada por uma soma de todos os píxeis na respetiva imagem, equação 2.16, numa região retangular formada entre as coordenadas (x,y) e a origem como representado na figura 2.13a.

$$I_{\sum}(\chi) = \sum_{i=0}^{x}\sum_{j=0}^{y} I(i,j) \tag{2.16}$$

(a) **(b)**

Figura 2.13 – Representação da imagem integral à esquerda e calculo da área A usando uma imagem integral à direita: A = L_4 + L_1 - L_2 - L_3.

Como já foi referido, o descritor é baseado no determinante da matriz Hessiana. Dado um píxel de coordenadas (x,y) a matriz Hessiana, H, é dada pelas derivadas parciais da imagem I nesse píxel:

$$H(I(x,y)) = \begin{bmatrix} \frac{d^2I}{dx^2} & \frac{d^2I}{dxdy} \\ \frac{d^2I}{dxdy} & \frac{d^2I}{dy^2} \end{bmatrix} \tag{2.17}$$

Para adaptar a qualquer escala, filtra-se a imagem usando um *kernel* gaussiano, logo a matriz Hessiana $H(x,y,\sigma)$ pode ser calculada agora, em função do espaço (x,y) e da escala σ:

$$H(x,y,\sigma) = \begin{bmatrix} L_{xx}(x,y,\sigma) & L_{xy}(x,y,\sigma) \\ L_{xy}(x,y,\sigma) & L_{yy}(x,y,\sigma) \end{bmatrix} \tag{2.18}$$

onde $L_{xx}(x,y,\sigma)$ é a convolução da derivada de segunda ordem gaussiana com a imagem original I no ponto (x,y), seguindo o mesmo raciocínio para $L_{xy}(x,y,\sigma)$, $L_{yx}(x,y,\sigma)$ e $L_{yy}(x,y,\sigma)$.

De modo a calcular o determinante da matriz H, primeiro é necessário aplicar a convolução com um *kernel* gaussiano e depois calcular a derivada de segunda ordem. Este processo complexo é contornado usando o método *box filter* que, apesar de apresentar resultados similares ao filtro gaussiano, é um filtro significativamente mais rápido [9]. De uma forma simples, o *box filter* pode ser descrito como o cálculo de um novo valor de píxel com base nos valores dos píxeis que estão em seu redor. Ou seja, o filtro é usado para calcular a derivada gaussiana de segunda ordem, melhorando o tempo de computação, usando a técnica integral de imagem [21].

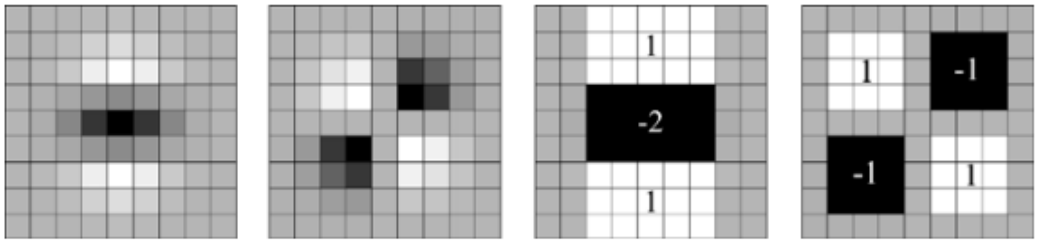


Figura 2.14 – Da esquerda para a direita: as derivadas parciais gaussianas de segunda ordem nas direções y, (L_{yy}), e xy, (L_{xy}), respetivamente, seguidas da aproximação do *box filter* nas direções x e xy, respetivamente [21]. As regiões cinzas são iguais a zero.

Os filtros de caixa 9 × 9, representados na figura 2.14, são aproximações para derivadas gaussianas de segunda ordem com $\sigma = 1.2$. Denotando essas aproximações por D_{xx}, D_{yy} e D_{xy}, pode-se representar o determinante do Hessiano aproximado como:

$$det(H_{\text{aproximado}}) = D_{xx}D_{yy} - (wD_{xy})^2 \tag{2.19}$$

onde w é um peso relativo da resposta do filtro.

O objetivo do descritor SURF é fornecer uma descrição única e robusta de um recurso. A criação do descritor SURF ocorre em duas etapas. A primeira etapa consiste em fixar uma orientação reproduzível para todos os pontos-chave, de forma a ser invariante à rotação, com base nas informações de uma região circular em torno do ponto-chave. O descritor é baseado nas respostas de Wavelet Haar na direção x e y [22] e pode ser calculado de forma eficiente com imagens integrais. A orientação é dada pelo maior valor da soma de todas as respostas dentro de uma janela de orientação deslizante. Em seguida, constrói-se uma região quadrada centrada e orientada com o ponto-chave e extraí-se o descritor SURF.

O método apresenta características que são invariáveis à rotação e escala, mas têm pouca invariância afim. Por fim, o descritor apresenta duas versões de operação, uma com 64 bits e outra expandida para 128 bits. Na versão com 128 bits, os descritores resultantes são maiores, o que significa que a correspondência deles será mais lenta, mas o maior poder descritivo dos recursos estendidos melhora o desempenho [6].

2.7.3 FAST

O *Features from Accelerated Segments Test* (FAST) [23] é um algoritmo apenas dedicado à identificação dos pontos-chave de uma imagem. Em cada píxel candidato a ponto característico, chamado píxel p, o algoritmo deteta os píxeis que o rodeiam sob a forma de um anel com um determinado raio r (Figura 2.15). Com base na intensidade do píxel p, com o valor I_p, é selecionado um valor limite de intensidade (T). Assim, os N píxeis que constituem o anel são analisados ao nível da intensidade e caso existam, por exemplo, pelo menos N/2+1 píxeis com um valor maior que $I_p + T$ ou menor que I_p - T, o píxel central p é considerado um ponto-chave.

Uma das desvantagens deste método é a deteção de pontos-chave adjacentes, uma vez que os píxeis contíguos ao píxel central podem também estabelecer as condições dos pontos-chave. Logo será necessário avaliar cada um desses pontos e verificar qual é o mais relevante, sendo que, neste caso, o ponto-chave é o que apresenta uma maior diferença de intensidade entre os

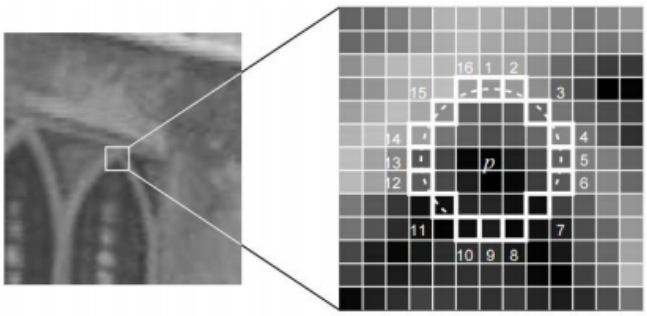


Figura 2.15 – Assumindo um determinado raio, o ponto central, p, candidato a canto é rodeado por um anel de 16 píxeis que são analisados e comparados com o ponto central quando à sua intensidade [23].

---

restantes pontos do anel. A principal vantagem deste método é sua velocidade de execução, pois consegue identificar um ponto-chave com bastante simplicidade algorítmica, o que se reflete no seu tempo de computação.

2.7.4 BRIEF

O *Binary Robust Independent Elementary Features* (BRIEF) [24] [25] é responsável não por detetar pontos característicos, mas apenas por formar descritores a partir de pontos-chave previamente identificados, recorrendo a um outro algoritmo de deteção de *features* .

O BRIEF começa por filtrar a imagem sob a forma de uma convolução, aplicando um filtro gaussiano. Depois, sobre cada ponto-chave, centra-se um padrão de amostragem e são selecionados, aleatoriamente, pares de pontos x e y contidos nesse padrão. A seleção destes pares de pontos é depois comparada quanto à intensidade dos pontos, e se a intensidade do ponto x, $p(x)$, for maior ou igual que a do ponto y, $p(y)$, então ao descritor é adicionado um '0' caso contrário é adicionando um '1', como representado na seguinte equação:

$$\tau(p; x, y) := \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases} \tag{2.20}$$

Este procedimento é realizado para cada ponto-chave e são criados desta forma os descritores binários. Assim, cada ponto-chave é descrito por um vetor de recursos que é composto por várias palavras binárias de 128 a 512 *bits*. O descritor pode ser dado pela equação 2.21, onde n é o número de *bits* das palavras binárias.

$$f_n(p) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(p; x_i; y_i) \tag{2.21}$$

Com o objetivo de comparar descritores de diferentes imagens, esse processo torna-se simples e rápido quando se trata de descritores binários, uma vez que basta verificar o número de *bits* diferentes entre eles, usando a distância de *hamming*. A vantagem deste método, quando comparado com os anteriormente referidos, reside nos baixos requisitos computacionais exigidos. No entanto, esta abordagem sofre em termos de confiabilidade e robustez, pois tem baixa tolerância a distorções e transformações como rotações e variações de escala [26].

2.7.5 BRISK

O *Binary Robust Invariant Scalable Keypoints* (BRISK) [26] é um método de extração e descrição de pontos-chave que foi desenvolvido com o objetivo de apresentar um melhor desempenho temporal que os métodos anteriormente referidos e, simultaneamente, sem a necessidade de altos requisitos computacionais [6].

Os pontos-chave são identificados usando a metodologia de deteção de canto *Adaptive and Generic Corner Detection Based on the Accelerated Segment Test* (AGAST) [18], encontrando todos os recursos em todas as escalas com o objetivo de lhes atribuir uma pontuação. Para alcançar a invariância de escala, o BRISK procura os máximos ao nível da imagem e do espaço-escala usando o critério de saliência.

Recorre-se a um padrão de amostragem distribuído, como ilustrado na figura 2.16, aplicados na vizinhança de cada ponto-chave. De seguida, são gerados, aleatoriamente, pares de pontos e classificados como de curtos e longos, segundo um limite mínimo e máximo das distâncias entre eles. Os pares longos são usados para determinar a orientação, usando o gradiente de intensidade, e os pares curtos para as comparações de intensidade que constroem o descritor, recorrendo à distância de *hamming*, tal como no BRIEF.

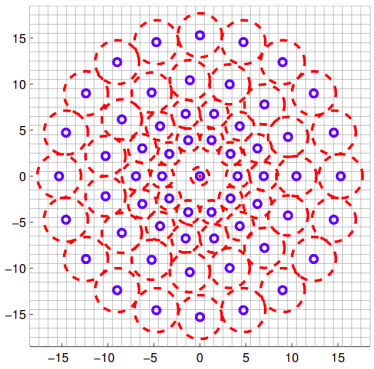


Figura 2.16 – Padrão de amostragem do BRISK [26].

Esta estratégia demonstra-se bastante eficiente computacionalmente, sendo que é ainda possível identificar a direção característica da cada ponto-chave de forma a permitir obter descritores normalizados de orientação, conseguindo, assim, atingir a invariância de rotação e permitindo uma maior robustez do método.

2.7.6 ORB

O algoritmo *Oriented FAST and Rotated* (ORB) [27] surge como uma agregação de variantes de dois métodos, nomeadamente o FAST, dedicado à identificação dos pontos-chave, e o BRIEF, responsável pela descrição dos mesmos.

Esta abordagem é essencialmente dedicada a sistemas com requisitos de elevada cadência e essa é a sua maior vantagem. Além da eficiência temporal, o ORB apresenta uma imunidade ao ruído gaussiano superior há imunidade apresentada por outros descritores como, por exemplo, o SIFT [27].

De forma a contornar a limitação imposta pelo FAST ao nível da orientação e multi-escala, o procedimento do ORB é usar uma pirâmide com a mesma imagem em diversas escalas, ou seja, com diferentes resoluções. Depois de redimensionadas as imagens para as

diferentes escalas, são então sujeitas ao algoritmo de deteção dos recursos. Assim, pode-se dizer que o ORB é invariante à escala [27]. Além da necessidade de ser sensível à multi-escala, a introdução de uma orientação é também conseguida pelo ORB. Logo, depois de localizados todos os pontos-chave, é atribuída uma orientação com base nas variações dos níveis de intensidade que rodeiam o próprio ponto-chave. Essa orientação é obtida usando o centróide de intensidade [11], onde os momentos das intensidades I da imagem nos pontos de coordenadas x, y são definidos na equação 2.14, enquanto que o centróide é calculado com base na equação 2.15.

Assume-se que a intensidade de cada recurso é, na prática, um deslocamento relativamente ao centro e esse vetor criado é usado como indicação de uma orientação [27]. A orientação é calculada da seguinte forma:

$$\theta = tan^{-1}\left(\frac{m_{01}}{m_{10}}\right) \tag{2.22}$$

Como anteriormente referido o BRIEF apresenta algumas limitações quando lhe são exigidas condições de rotação. Assim, o ORB apresenta uma variante do BRIEF, que permite ao método ser invariante a rotações. Na prática, resulta em direcionar o BRIEF de acordo com a orientação dos pontos-chave. Para qualquer conjunto de recursos de n testes binários no local (x_i,y_i), é necessário uma matriz 2 x n [28]:

$$S = \begin{pmatrix} x_1, ...x_n \\ y_1, ...y_n \end{pmatrix} \tag{2.23}$$

A matriz baseia-se na orientação dos recursos obtida em (2.22) e em conjunto com a matriz rotação correspondente, R_θ, é possível construir a matriz S_θ:

$$S_\theta = R_\theta S \tag{2.24}$$

Assim, o descritor BRIEF apresentado na equação 2.21 , admitindo invariância à rotação pode ser dado por:

$$g_n(p, \theta) := f_n(p) | (x_i, y_i) \in S_\theta \tag{2.25}$$

Depois de compensada a limitação da orientação, o ORB ao contrário do BRIEF que apresenta um processo aleatório de seleção dos pares de pontos descritores, apresenta um processo de aprendizagem dos pares de ponto [27], de forma a garantir que os recursos selecionados apresentem pouca correlação, de modo que cada novo par contribua com novas informações, tornando o descritor mais robusto.

2.7.7 KAZE

KAZE [29] é um detector e descritor de pontos-chave que explora o espaço de escala não-linear por meio de filtragem de difusão não linear (Equação 2.26). Em vez de usar o desfoque gaussiano como o SIFT, que faz com que não preserve totalmente os limites, o KAZE combina a filtragem de difusão não-linear com uma função de condutividade proposta por Perona e Malik [30].

A extração dos pontos-chave é baseada no determinante da matriz Hessiana, que é calculado em vários níveis de escala, sendo que a orientação é atribuída tal como no SIFT, com a principal diferença de usar vetores para representar os gradientes em vez de histogramas.

No que diz respeito à descrição, o KAZE usa uma variante do descritor SURF, logo introduz a propriedade de invariância de rotação ao encontrar a orientação dominante em uma vizinhança circular em torno de cada recurso detectado. Finalmente, o vetor descritor é normalizado num vetor unitário para torná-lo invariante ao contraste. Pode-se afirmar que os recursos KAZE são, assim, invariáveis à rotação, escala e transformações afins limitadas.

A equação 2.26 representa a fórmula geral da difusão não linear:

$$\frac{\partial L}{\partial t} = div(c(x, y, t).\nabla L) \tag{2.26}$$

onde c é a função condutividade, *div* é a divergência, ∇ é o operador gradiente e L é a luminância da imagem.

2.7.8 AKAZE

O KAZE acelerado (AKAZE) é um algoritmo também baseado em filtragem de difusão não-linear, como o KAZE, mas os espaços de escala não-linear são construídos usando um *framework* computacionalmente mais eficiente chamada FED [31]. O descritor de AKAZE é baseado no algoritmo Modified Local Difference Binary (MLDB), que também é altamente eficiente [32]. Os recursos do AKAZE são invariantes à escala, rotação e transformações afim limitadas. Na prática, o AKAZE é baseado no método KAZE apenas sofrendo algumas alterações em algumas metodologias de forma a tornar o processo mais eficiente a nível computacional.

2.7.9 Comparação dos extratores e descritores

A partir deste estudo, é possível concluir que existe uma grande variedade de métodos baseados em características locais com características bastante distintas. Devido a esta diversidade, deve-se ter em consideração os requisitos estabelecidos e escolher de uma forma criteriosa o algoritmo de extração e descrição de *features*. A tabela 2.1 dá uma visão global dos algoritmos apresentados, onde o tempo de computação é divido em 3 níveis, sendo o nível 1 (+) o mais lento e o nível 3 (+ + +) o mais rápido.

Algoritmo	Extrator Ponto-Chave	Descritor Ponto-Chave	Invariância à Escala	Invariância à Rotação	Tempo de computação
SIFT	✓	✓	✓	✓	+
SURF	✓	✓	✓	✓	++
FAST	✓	×	✓	×	++
BRIEF	×	✓	×	×	+++
BRISK	✓	✓	✓	✓	++
ORB	✓	✓	✓	✓	+++
KAZE	✓	✓	✓	✓	+
AKAZE	✓	✓	✓	✓	+++

Tabela 2.1 – Visão global dos extratores e descritores de *features*.

Com vista a ter uma melhor perceção das potencialidades e vulnerabilidades de cada um dos algoritmos, estudaram-se alguns artigos que se dedicaram à comparação dos mesmos. Com base no artigo designado *"A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK"* [18] é possível retirar as seguintes conclusões:

- O ORB é algoritmo que deteta o maior número de *features*, seguido do BRISK e do SURF.
- O ORB é o algoritmo mais eficiente, seguido do BRISK e do SURF. Ou seja, são os que apresentam uma melhor relação entre o número de *features* extraídas e o tempo necessário.
- Relativamente à eficiência de correspondência de *features*, o ORB é o mais eficiente, seguido do BRISK e do AKAZE.
- SIFT, SURF, BRISK são os algoritmos mais invariantes à escala;
- ORB, BRISK, KAZE são os algoritmos mais invariantes à rotação;
- ORB e BRISK são os algoritmos mais invariantes a transformações afins;
- SIFT e KAZE são os algoritmos mais precisos na deteção de *features*;

Além deste artigo, estudou-se outro designado *"Object recognition using local invariant features for robotic applications: A survey"* [17] onde se realiza o estudo dos diferentes algoritmos com a perspetiva de usar em aplicações robóticas, onde se concluiu que:

- Na perspetiva de aplicações robóticas, os detetores de pontos-chave mais adequados são ORB, BRISK e SIFT e os descritores mais adequados são ORB, BRISK, SIFT e SURF;
- Uma abordagem força bruta torna-se inviável com o aumento da dimensão dos descritores;
- O desempenho dos sistemas de reconhecimento de objetos usando recursos invariantes locais depende fortemente dos métodos de verificação de transformação geométrica usados;

- Os sistemas de reconhecimento de objetos baseados em ORB são mais rápidos, enquanto os SIFT são mais precisos.

Os métodos de correspondência de objetos anteriormente referidos são de seguida apresentados e detalhados.

2.8 Correspondência de features

No processo de correspondência entre duas imagens, usando *features*, cada descritor de um ponto-chave, de uma imagem, é comparado com os descritores dos pontos-chave de outra imagem. Esta comparação baseia-se no cálculo da distância entre os descritores de ambas as imagens onde interessa determinar as correspondências com a menor distância. Esta distância pode ser encarada como uma métrica de similaridade entre os descritores onde, quanto menor a distância, maior é a similaridade. O cálculo da distância é, normalmente, efetuado segundo a distância euclidiana ou a de *hamming*, dependendo das características do descritor usado.

No processo de correspondência de descritores, são avaliadas as melhores correspondências, ou seja, as com menor distância. Segundo Lowe [19], estas correspondências podem ser filtradas com base no rácio das distâncias. Assim, para cada correspondência, com uma dada distância, é analisada a seguinte melhor correspondência e caso o rácio entre as distâncias seja superior a 0.8, a 2ª melhor correspondência é descartada [19]. Segundo o autor, este critério elimina cerca de 90% de falsos positivos e elimina apenas 5% de verdadeiros positivos. Na prática, este é um critério que permite determinar as correspondências que são mais distintivas, dado que impossibilita a existência de correspondências demasiado semelhantes que acabam por descrever, desnecessariamente, as mesmas *features*, sendo que apenas uma delas pode estar correta.

De seguida são apresentados alguns dos métodos usados no processo de correspondência de *features*, nomeadamente, o *Brute Force* e o FLANN, assim como um método iterativo que permite eliminar ainda mais falsas correspondências, designadas pelo método como de *outliers*.

2.8.1 Força bruta

O método Força Bruta (*Brute Force*), como o nome indica, faz uma comparação grosseira e direta de todos os recursos previamente recolhidos, ou seja, compara na totalidade os dois conjuntos de descritores e gera uma lista de correspondências. Assim, para cada ponto-chave de um conjunto, é calculada, usando uma determinada métrica, a distância a cada um dos pontos-chave do outro conjunto e a melhor correspondência pode ser escolhida com base na menor distância [33].

Este método, disponibilizado pelo OpenCV, pode recorrer a várias métricas para o cálculo da distância entre dois vetores descritores de igual dimensão. O fator determinante é o tipo de descritores extraídos, uma vez que caso sejam do tipo *float* como, por exemplo, o SIFT e o SURF, poderá ser usada norma L1, equação 2.27, ou a norma L2, equação 2.28. As seguintes equações representam as suas formulas respetivamente [34]:

$$d_{normaL1} = \sum_{i=1}^{n} |a_i - b_i| \tag{2.27}$$

$$d_{normaL2} = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{2.28}$$

No entanto, caso se tratem de descritores binários como, por exemplo, o ORB, BRIEF e o BRISK, então deve-se usar a distância de *hamming* [35]:

$$d_{hamming} = \sum_{i=1}^{n}(a_i \oplus b_i) \tag{2.29}$$

Além desta métrica, existem ainda de um método adicional chamado verificação cruzada onde, o objetivo é aumentar a robustez do algoritmo [35][6]. Na prática, este algoritmo de confirmação verifica se num dado conjunto de pontos-chave as distâncias ao seus vizinhos mais próximo são iguais independentemente da lista de descritores onde se inicia a pesquisa. Caso esta condição se estabeleça, o algoritmo considera uma correspondência válida.

A eficiência deste método degrada-se com o aumento significativo de recursos extraídos pois é um processo pouco criterioso. Em situações com poucos recursos pode-se apresentar como uma solução bastante válida, no entanto, em aplicações de elevada cadência e com grandes conjuntos de dados, esta não será a melhor abordagem. Assim uma alternativa ao *Brute Force* é usar o FLANN, já que para grandes conjuntos de dados apresenta-se como sendo mais rápido [35].

2.8.2 FLANN

A nível computacional, uma das tarefas mais complexas nos algoritmos de visão por computador consiste em procurar pelas correspondências mais próximas de descritores previamente recolhidos. O problema agrava-se quando o objetivo passa por aplicar o algoritmo de pesquisa num conjunto de dados numa ordem de grandeza dos milhares de recursos [36]. Numa perspetiva de reconhecimento de objetos, este processo pode demonstrar-se importante e, caso se pretenda integrar num sistema de elevada cadência, esta problemática ganha ainda mais relevo.

O *Fast Library for Approximate Nearest Neighbor (FLANN)*, como o nome assim o indica, é uma biblioteca que permite realizar pesquisas aproximadas e rápidas dos seus vizinhos mais próximos em extensos conjuntos de dados, onde, por vezes, os vizinhos não ótimos são retornados. Esta biblioteca do OpenCV contém uma vasta seleção de algoritmos para a busca do vizinho mais próximo, de forma a retirar o melhor proveito para os diferentes conjuntos de dados existentes. De seguida são apresentados, de forma breve, dois dos métodos disponibilizados pela biblioteca:

- *randomized kd-tree* - Uma árvore k-dimensional é uma estrutura de dados de partição de espaço para organizar pontos em um espaço k-dimensional e este é um algoritmo de pesquisa aproximada pelo vizinho mais próximo [37]. As árvores kd aleatórias são construídas através da divisão dos dados, a partir de um processo aleatório que escolhe as dimensões, de cada uma delas, baseando-se num conjunto de dados que apresentam uma maior variância. Ao pesquisar as árvores paralelamente, uma única fila de prioridade é mantida em todas as árvores aleatórias. A fila de prioridade é ordenada aumentando a distância para o limite de decisão de cada ramo na fila, de modo que a pesquisa explore primeiro as ramificações mais próximas de todas as árvores. Depois de um ponto dos

dados ser examinado dentro de uma árvore, ele é marcado para não ser reexaminado em outra árvore. A função da biblioteca FLANN permite especificar o número de árvores a serem construídas, no entanto, é preciso ter em conta a sobrecarga da memória no uso de várias árvores aleatórias, pois esta aumenta linearmente com o número de árvores, portanto, em algum ponto, o recurso a mais árvores com vista à aceleração pode não justificar a memória adicional requerida [38].

- *Locality Sensitive Hashing* (LSH) - Este algoritmo baseia-se em construir um conjunto de funções hash que, na prática, são algoritmos que mapeiam dados de grande dimensão em tabelas de dispersão (*hash*), em diferentes grupos, permitindo a consulta rápida de dados [27]. O método baseia-se em fazer a dispersão dos pontos de dados em intervalos, de modo que os pontos de dados próximos uns dos outros estejam localizados nos mesmos intervalos com uma alta probabilidade, enquanto os pontos de dados distantes uns dos outros provavelmente estejam em intervalos diferentes [39]. Mostrou-se que o método LSH apresenta bons resultados em conjuntos de dados de grande dimensão, quer ao nível da exatidão do vizinho mais próximo, quer a nível temporal [39][40].

  A função que implementa este algoritmo apenas é aplicável em métodos que retornam conjunto de dados binários como, por exemplo, o caso dos descritores ORB ou BRIEF. Neste caso, a função *hash* é, simplesmente, um subconjunto de bits que caracteriza os descritores. Dada a natureza dos descritores, o cálculo da distância é realizado com a distância de *Hamming*. Outra variação deste método é o *multi-probe*, que permite melhorar o LSH tradicional através da análise dos grupos vizinhos. Assim, apesar de resultar em mais correspondências para avaliar, na realidade, faz com que o número de tabelas *hash* diminua, o que resulta num menor recurso computacional [27].

2.8.3 RANSAC

O *Random Sample Consensus* (RANSAC) [41] é um método iterativo, proposto por Fischler e Booles, que permite estimar os parâmetros de um modelo matemático a partir de um conjunto de dados extraídos, admitindo a existência de uma grande quantidade de *outliers*. Os *outliers* são dados que se diferenciam drasticamente de todos os outros, ou seja, que não seguem a normalidade do comportamento dos dados, o que causará anomalias nos resultados obtidos caso não sejam devidamente identificados .

Ao contrário das muitas técnicas normalmente usadas na visão computacional para fazer estimativas robustas, como é o caso do método dos mínimos quadrados, o RANSAC foi desenvolvido dentro da comunidade da visão por computador. Este método apresenta algumas vantagens em relação aos métodos convencionais uma vez que, por um lado, não é necessário uma boa estimativa inicial para garantir a convergência e, por outro lado, a convergência é atingida mesmo num conjunto de dados onde existe uma grande percentagem de *outliers*.

No âmbito da visão computacional, o RANSAC surge pela necessidade de colmatar algumas falhas que os métodos de extração e descrição de *features*, anteriormente referidos possam apresentar. Assim, este método é capaz de filtrar correspondências de *features*, eliminando as possíveis falsas correspondências. Os erros produzidos pelos detetores de recursos podem classificar-se com dois tipos [42]. Por um lado, têm-se os erros de classificação que ocorrem quando um detetor identifica incorrectamente uma parte da imagem como uma ocorrência de um recurso, quando na verdade não se verifica e, por outro lado, têm-se erros de localização que se constata quando o detetor identifica corretamente a existência de um recurso,

no entanto, falha no cálculo da sua localização.

Na figura 2.17 pode-se verificar visualmente um caso concreto de determinar uma reta que se aproxima o máximo de *inliers* possível.

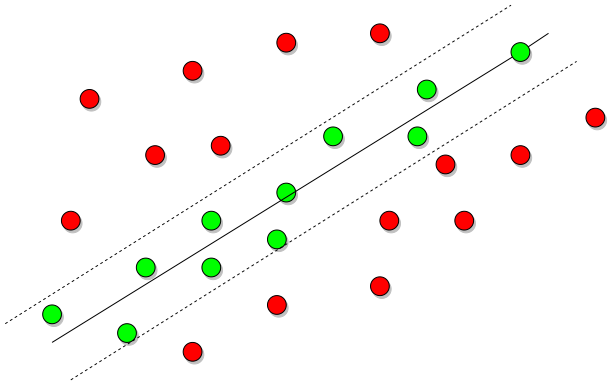


Figura 2.17 – Ilustração do método RANSAC onde os *outliers* estão representados pelos pontos a vermelho e os pontos verdes são os *inliers*.

Segundo Konstantinos [43] pode-se sintetizar o método RANSAC como apresentado no algoritmo 1 do seguinte modo:

Algoritmo 1 RANSAC

1. Selecionar, aleatoriamente, o número mínimo de pontos necessários para determinar os parâmetros do modelo;
2. Calcular os parâmetros do modelo;
3. Determinar quantos pontos do conjunto de todos os pontos se ajustam a uma predefinida tolerância, ou seja, quantos *inliers* existem. A margem de tolerância está ilustrada pela linha a tracejado;
4. É armazenado para o modelo estimado o número de *inliers* estimados. E em cada ciclo compara-os com os modelos já armazenados, numa prespetiva de determinar o modelo com o maior número de *inliers*;
5. Voltar a repetir todos os passos de 1 a 4 até um número máximo de N iterações predefinidas;
6. No fim das N iterações, o algoritmo devolve os parâmetros do modelo com um maior número de *inliers*.

Assim este algoritmo permite configurar não só o número máximo de iterações, sendo que quanto mais iterações, maior é a probabilidade de uma correta aferição dos parâmetros do modelo, bem como definir o valor limite para a distinção entre *inliers* e *outliers*.

O algoritmo RANSAC pode, por exemplo, ser aplicado no cálculo das transformações entre dois conjuntos de descritores. Seguindo o mesmo raciocínio do algoritmo 1, seleciona-se aleatoriamente uma correspondência e verifica-se as restantes correspondências. De seguida, avalia-se quantas dessas correspondências são consideradas *inliers* ou *outliers*, relativamente à correspondência selecionada. O processo repete-se até um número máximo de ciclos pré-definidos e no fim analisa-se e seleciona-se qual a que obteve uma maior número de *inliers*. Com base nessa seleção estima-se as transformações, por exemplo, ao nível da translação, escala e rotação.

2.8.4 Transformações geométricas

A transformação geométrica é uma aplicação objetiva entre duas figuras bidimensionais de forma que, a partir de uma figura geométrica original, transforma-se noutra figura geometricamente igual ou equivalente, com base numa matriz transformação. As transformações geométricas mais usuais podem ser de 3 tipos:

- Similaridade: baseadas numa matriz, constituída por 4 graus de liberdade, nomeadamente, a rotação, a escala uniforme e a translação no eixo x e y;
- Afim: baseadas numa matriz, composta por 6 graus de liberdade, particularmente, a rotação, translação no eixo x e y, a escala não uniforme e o enviesamento. Entende-se o enviesamento como uma transformação que pode ocorrer em cada um dos eixos x e y. Por exemplo, o enviesamento no eixo x mantêm as coordenadas no eixo y e transforma as coordenadas do eixo x.
- Homografia: baseada numa matriz transformação, com 8 graus de liberdade, nomeadamente, a rotação, translação no eixo x e y, a escala não uniforme, o enviesamento e a perspetiva;

Transformações similaridade

As transformações similaridade são constituídas por 4 graus de liberdade, nomeadamente, a rotação, a escala uniforme e a translação no eixo x e y. Devido ao facto de todas estas transformações serem lineares, pode-se caracterizar a transformação entre os dois conjuntos de pontos com uma matriz T de dimensões 3x3. Assim, as matrizes M que constituem a matriz transformação T assumem três formas de representação, conforme as transformações que se verifiquem:

- A matriz translação referente aos eixos x e y é dada por:

$$M_{translacão} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.30}$$

- No que diz respeito à escala, a matriz apresenta um fator de escala uniforme dado por s. A matriz pode representar-se da seguinte forma:

$$M_{escala} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

- Quanto à rotação, pode ser representada pela seguinte matriz:

$$M_{rotação} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.32}$$

Assim, a matriz geral da transformação 2D de similaridade, T, com quatro graus de liberdade pode ser dada representada da seguinte forma:

$$T = \begin{bmatrix} s \cdot cos(\theta) & -s \cdot sin(\theta) & t_x \\ s \cdot sin(\theta) & s \cdot cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.33}$$

Depois de obtida a matriz transformação 2.33 e assumindo a seguinte nomenclatura:

$$T = \begin{bmatrix} s \cdot cos(\theta) & -s \cdot sin(\theta) & t_x \\ s \cdot sin(\theta) & s \cdot cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.34}$$

pretende-se fazer a extração de cada um dos graus de liberdade. As componentes da translação são retiradas diretamente em t_x e t_y, sem que seja necessário qualquer cálculo adicional. Quanto ao ângulo, obtém-se da seguinte forma:

$$tan(\theta) = \frac{c}{d} = \frac{s \cdot sin(\theta)}{s \cdot cos(\theta)} = \frac{sin(\theta)}{cos(\theta)} \tag{2.35}$$

$$\theta = tan^{-1}\left(\frac{sin(\theta)}{cos(\theta)}\right) \tag{2.36}$$

Relativamente ao fator de escala, pode ser obtido através da seguinte expressão:

$$s = |\sqrt{a^2 + c^2}| = |\sqrt{b^2 + d^2}| \tag{2.37}$$

As transformações de similaridade podem ser destinadas a diversas aplicações na área da visão por computador e o reconhecimento de objetos é um dos diversos exemplos onde podem ser aplicadas. No caso do reconhecimento de objetos, sabe-se que imagens relativas ao mesmo objeto serão diferentes e, por isso, interessa calcular as transformações que as relacionam. Neste caso de estudo, as transformações de similaridade são frequentemente usadas, em vez das transformações afim e de perspectiva, não só pelo facto de terem menos parâmetros e, portanto, são mais fáceis e rápidas de determinar, mas também pelo facto de a distância entre a câmara e o objeto ser fixa, então não existe variações nem de escala nem enviesamento, logo não é necessário calcular esses parâmetros [44].

Assim, usando umas das funções disponibilizadas pelo OpenCV, nomeadamente, o *estimateAffinePartial2D*, é possível calcular a matriz transformação recorrendo aos pontos representativos de correspondências. Desta forma, são necessários no mínimo 2 conjuntos de pontos 2D, permitindo construir 4 equações e determinar os 4 graus de liberdade. Esta função calcula a matriz transformação e os respetivos *inliers* com base no algoritmo RANSAC e no método de Levenberg-Marquardt.

O método RANSAC foi apresentado na subsecção 2.8.3 e o algoritmo de Levenberg-Marquardt é um método iterativo que localiza os mínimos de uma função que pode ser expressa como a soma de quadrados de funções não-lineares. Tornou-se uma técnica padrão para problemas não lineares dos mínimos quadrados e pode ser entendido como uma combinação do método de otimização Gauss-Newton e a técnica do gradiente descendente [45]. Este método apresenta um bom compromisso entre robustez e eficiência, por um lado, devido à sua capacidade de encontrar uma solução independentemente do ponto de partida e, por outro lado, uma vez que apresenta uma satisfatória rapidez de convergência [45].

2.9 Conclusão

Neste capítulo estudou-se alguns algoritmos de processamento de baixo nível com vista à resolução da situação problema. No que diz respeito às correspondências de objetos, abordaram-se algumas possibilidades e os métodos de eleição são os que usam extração, descrição e comparação de *features*. No entanto, existem diversos extratores e descritores de recursos com características bastante distintas e, por isso, a seleção de um deles deve ser tomada de uma forma ponderada e rigorosa, consoante a aplicação que se pretenda. Neste sentido serão realizado testes de forma a aferir qual o que melhor se adequa aos objetivos definidos. Além disso, será necessário avaliar qual a melhor abordagem entre o método *Brute Force* e os algoritmos de aproximação disponibilizados pela biblioteca FLANN. Posteriormente, com base no algoritmo RANSAC e na teoria das transformações geométricas, será estimada a matriz transformação similaridade que dará a indicação da posição e orientação dos objetos.

Capítulo 3

Manipuladores robóticos

Os sistemas robóticos, apesar de já estarem bastante presentes na indústria moderna, estão a sofrer constantes mudanças, provocando impactos sócio-económicos cada vez mais significativos [46]. Na indústria robótica, os robôs manipuladores têm promovido fortemente os processos de produção convencionais devido à sua elevada cadência, alta robustez e extrema precisão [47]. Com o avanço da tecnologia, diversos tipos de manipuladores robóticos que emulam um braço humano, têm surgido de forma a dar respostas às exigências dos processos industriais.

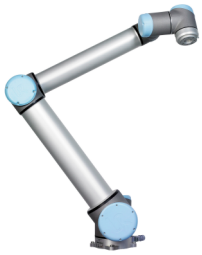
Os elementos que constituem os manipuladores são nomeados nesta dissertação como juntas (*joints*) e segmentos (*links*). Um dos fatores mais determinantes na caracterização de um manipulador é o número de graus de liberdade, uma vez que determina os tipos de movimentos a que o robô está limitado. O grau de liberdade é, essencialmente, dado pelo número de juntas que o constituem [48], sendo que quanto maior esse grau, mais complexa será a cinemática associada. Dado que as juntas têm uma papel importante na caracterização dos manipuladores é relevante mencionar os diferentes tipos de juntas mais comuns. Estas são as cilíndricas, prismáticas, rotativas, planares, esféricas e parafuso, das quais as rotativas e prismáticas são as mais usadas nos manipuladores industriais [49].

3.1 Tipos de Manipuladores

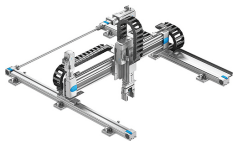
No que diz respeito aos elementos físicos que compõe um manipulador robótico, destacam-se os atuadores e o sensores. Por um lado, os atuadores são os responsáveis por converter um tipo de energia em movimento mecânico, enquanto que os sensores transformam grandezas físicas em sinais elétricos que, depois de processados, podem dar informação sobre a velocidade, posição do manipulador, ou detalhes sobre o objeto a manipular como, por exemplo, a respetiva massa [50].

Quanto aos atuadores, podem-se ainda classificar em hidráulicos, pneumáticos (Figura 3.1c) e electromagnéticos (Figura 3.1a) [48]. Cada tipo apresenta as suas vantagens e desvantagens logo, o importante é ter em consideração o propósito do manipulador para que se possa optar pela melhor solução. Os objetivos podem ser diversos como, por exemplo, manipular cargas de peso elevado, ter uma alta cadência, ter uma extrema precisão, ou ser barato e eficiente do ponto de vista energético, entre outros. Numa vasta gama de tarefas possíveis para os manipuladores, podem-se enquadrar tarefas que destacam a quantidade de carga que deve suportar, a velocidade de trabalho exigida, a precisão requerida, os custos associados,

entre outros fatores.



(a) Manipulador elétrico UR10 da *Universal Robots* [51].



(b) Manipulador robótico cartesiano da *Festo* [52].

(c) Manipulador pneumático colaborativo da *Festo* [52].

(d) Manipulador SCARA, SR-6iA, da *FANUC* [53].

Figura 3.1 – Exemplos de tipos de manipuladores.

Os tipos de manipuladores industriais podem-se distinguir em 5 categorias diferentes [54]:

- Cartesianos - Constituídos apenas por juntas prismáticas, estes robôs são caracterizados principalmente pela sua pequena área de trabalho, e por possuírem um alto grau de rigidez mecânica (Figura 3.1b).
- Cilíndricos - Constituídos por juntas rotativas e prismáticas, que permite a formação de movimentos rotacionais e lineares com uma área de trabalho um pouco maior que os robôs de coordenadas cartesianas.
- Esféricos - Constituídas por juntas rotativas e esféricas, de forma a que a garra se movimente num espaço esférico. Possuem uma área de trabalho maior que os robôs cilíndricos.
- SCARA - Constituídos por duas juntas rotativas e uma prismáticas, são também conhecidos por robôs com articulações horizontais (Figura 3.1d).
- Antropomórfico - Constituídos apenas por juntas rotativas, são robôs que possuem uma grande flexibilidade e podem ocupar uma grande área de trabalho (Figura 3.2d).

Existe ainda uma categoria de manipuladores que se nomeiam como manipuladores colaborativos (*Cobots*). Estes manipuladores robóticos, quer sejam fixos ou móveis, têm ganho cada vez mais relevância, devido à grande potencialidade e versatilidade de aplicações a que podem ser destinados [55]. Na prática, estes robôs destacam-se pela sua capacidade de colaborar em tarefas particularmente difíceis para um ser humano, sendo que nunca comprometem a segurança do colaborador e não havendo, por isso, a necessidade das típicas vedações de segurança.

3.2 Tipos de Garras

O objetivo dos manipuladores robóticos passa por manipular objetos de uma forma fácil e rápida e, por isso, é necessário que seja acoplado ao manipulador uma garra (*gripper*) para o devido efeito. Existem diversos mecanismos que se adaptam às necessidades de cada aplicação. Os principais fatores de decisão na escolha de uma garra podem incidir em questões como a massa, a textura, a dimensão, a resistência, a forma, o atrito e a temperatura dos objetos que se pretendem manipular. Relativamente aos mecanismos, estes podem ser mecânicos (Figura 3.2d), magnéticos, pneumáticos (Figura 3.1c), a vácuo (Figura 3.2c), hidráulicos, entre outros [56].

As garras de precisão (Figura 3.2a), pneumáticas e antropomórficas limitam-se a pressionar os objetos e utilizam o atrito entre as superfícies para os agarrar. No caso da garra antropomórfica, esta apresenta vários graus de liberdade, uma vez que simula as articulações da mão humana. A garra da figura 3.2c agarra os objetos utilizando sistemas de vácuo com base na pressão do ar.

(a) Garra de precisão - Caliper NSR-PG-10-20-URE da *New Scale Robotics*.

(b) Garra pneumática da *SCHUNK Universal Robots*.

(c) Garra Picobot a vácuo da *Piab AB*

(d) Garra antropomórfica com 5 dedos da *qbrotics*.

Figura 3.2 – Exemplos de tipos de garras [56].

3.3 Calibração Hand-Eye

A calibração *Hand-eye* é uma calibração extrínseca entre um sensor e um sistema robótico. Existem duas variantes possíveis dentro deste âmbito. Por um lado, existe a possibilidade de a câmara estar acoplada à mão do manipulador (*eye-on-hand*). Por outro lado, é também possível que a câmara esteja fixa (*eye-to-base*), como ilustrado na figura 3.3 [57]. Tendo em conta os objetivos apresentados, o caso abordado será o *eye-to-base* pois a câmara estará fixa sobre o tapete rolante e, para a calibração, será necessário um padrão de calibração. A precisão da estimativa da posição e orientação do padrão é crítica para a resolução do

problema, por isso, a maioria dos algoritmos de calibração recorre a padrões de calibração como, por exemplo, um *ArUco* ou um *ChArUco*.

A estimativa da posição do padrão é um processo baseado em encontrar correspondências entre pontos no ambiente real e sua projeção numa imagem bidimensional. Concretamente, o marcador *ArUco* é um padrão quadrado sintético composto por uma ampla borda preta e uma matriz binária interna que determina o seu identificador (ID), como ilustrado na Figura 3.4. A vantagem destes marcadores é que um único marcador fornece correspondências suficientes para obter a posição e orientação do marcador relativamente à câmara. Além disso, a sua borda preta facilita a deteção rápida, e a codificação binária interna torna-os especialmente robustos, permitindo a possibilidade de aplicar técnicas de deteção e correção de erros [58].

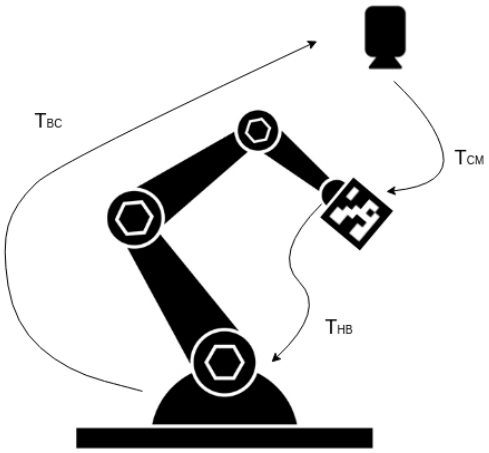


Figura 3.3 – Ilustração das transformações geométricas usadas na calibração *Hand-eye*.

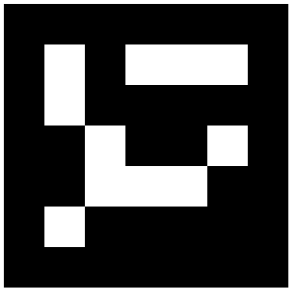


Figura 3.4 – Exemplo de um ArUco gerado com identificador igual a 300 [59].

Um dos métodos mais aceites pela comunidade da robótica é a solução apresentada por Tsai e Lens [57]. Estes autores demonstraram que a eficiência do seu algoritmo era independente do número de amostras adquiridas. Este é o método adotado pela biblioteca OpenCV. O procedimento é o seguinte:

- O marcador é acoplado à garra do robô e esta é movida a fim de adquirir amostras em várias posições e orientações. São necessárias, pelo menos duas, no entanto é recomendado usar mais. O robô manipulador deve possuir graus de liberdade suficientes de

modo a ser capaz de girar o marcador em torno dos seus eixos, mantendo-o sempre ao alcance da câmara;

- Para cada configuração, usando a estimativa de posição do padrão anteriormente referida, calcula-se a transformação entre a câmara e o marcador (T_{CM});
- Para cada configuração, a transformação homogénea entre a mão e a base do robô (T_{HB}) é registada usando a cinemática do robô;
- Com as respetivas amostras armazenadas, para cada configuração, a solução pode ser formulada como:

$$AX = ZB \tag{3.1}$$

onde A e B são as transformações T_{HB} e T_{CM}, respetivamente, X é a transformação desconhecida que se pretende determinar, nomeadamente, a transformação entre o base do robô e a câmara (T_{BC}) e Z é a transformação entre mão e o marcador (T_{HM}), como representado na Figura 3.5.

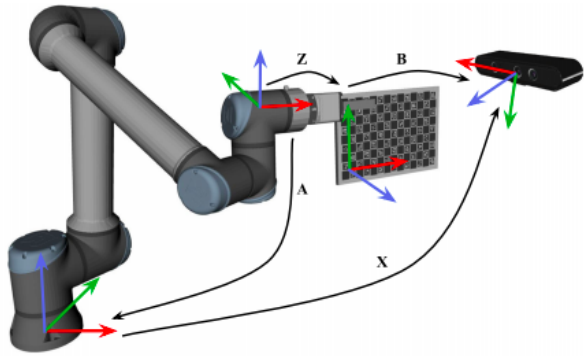


Figura 3.5 – Representação da formulação do problema da calibração *Hand-eye* no caso *eye-to-base* [60].

Segundo os autores da calibração [57], existem alguns cuidados a ter em atenção de forma a tornar a calibração mais rigorosa:

- Maximizar as rotações e minimizar as translações entre cada posição;
- Minimizar a distância entre a câmara e o *ArUco*;
- Usar posições redundantes;
- Calibrar os parâmetros intrínsecos da câmara;
- Ficheiro descritivo do robô deve ser preciso e rigoroso de forma a minimizar o erro associado;

A estimativa desta transformação é essencial em diversas aplicações robóticas que fazem uso de informações baseadas em visão para planear e controlar ações de movimento. A manipulação de objetos é um exemplo claro dessa necessidade, uma vez que esta tarefa depende

das informações visuais fornecidas pela câmara de modo a estimar a posição e orientação do objeto relativamente à base do manipulador. Neste contexto, uma imprecisa estimativa da transformação incapacita o rigor na realização da tarefa de manipulação.

3.4 Cinemática Inversa e Direta

A cinemática de um robô manipulador é o estudo da posição, orientação e da velocidade da sua garra, assim com das suas juntas. Relativamente à velocidade, tem-se em consideração a velocidade tanto linear como angular [61]. No ramo da cinemática existem duas distinções possíveis: a cinemática direta e a cinemática inversa. A cinemática direta (*Forward kinematics* (FK)) é o método que permite calcular a posição e a orientação da garra do manipulador sabendo as posições das juntas, onde existe uma só solução possível. A cinemática inversa (*Inverse kinematics* (IK)), num raciocínio invertido, calcula para uma determinada posição e orientação da garra a configuração de juntas a que corresponde, sendo que neste caso podem existir múltiplas soluções possíveis.

Existem, essencialmente, dois tipos de algoritmos de cinemática inversa [62]. Por um lado, os algoritmos que calculam a IK de uma forma iterativa e recursiva. Estes, apesar de não serem os mais rápidos são muitas vezes os mais utilizados pois são bastante versáteis e independentes do manipulador, o que faz com que a sua integração seja simples e rápida. Por outro lado, existem métodos de cálculo baseados em processos analíticos. Estes, apesar de apresentarem, normalmente, tempos de cálculo inferiores, têm a desvantagem de funcionar apenas para os manipuladores que foram definidos à partida, pois as expressões matemáticas resultantes são diretamente relacionadas com a estrutura do manipulador.

3.5 Planeadores de trajetórias

Assumindo que se sabe a configuração geométrica do robô e do mundo, assim como a sua posição inicial e a posição objetivo, é então necessário recorrer a um planeador de trajetórias, de forma a que o manipulador se movimente de forma suave e o mais direta possível desde o ponto em que se encontra até o seu objetivo. O planeador de trajetórias é responsável por encontrar a melhor trajetória possível tendo em conta, não só as suas próprias colisões como as colisões do mundo que o rodeia. Além disso, é possível impôr restrições relativamente à posição e orientação da garra durante todo o movimento. A trajetória obtida resume-se a uma sequência de posições de juntas, velocidades e acelerações que levam o robô da configuração inicial à final, respeitando as devidas restrições físicas.

Atualmente, existem muitos algoritmos de planeamento de trajetórias disponíveis. Porém, os tipos de planeadores mais comuns na área dos manipuladores robóticos, quando se usa o sistema operativo ROS são: o *Open Motion Planning Library* (OMPL) [63], *Covariant Hamiltonian Optimization for Motion Planning* (CHOMP) [64], *Stochastic Trajectory Optimization for Motion Planning* (STOMP) [65] e o *Search-Based Planning Library* (SBPL) [66]. Os algoritmos são na sua maioria probabilísticos, baseados em amostragem, ou seja, para uma determinada entrada, podem devolver saídas diferentes, portanto, não têm repetibilidade.

3.5.1 OMPL

A OMPL é uma biblioteca de código aberto formada por um conjunto de algoritmos de planeamento de movimento baseados em amostragem [63]. Estes algoritmos, além de serem estocásticos, são totalmente abstratos à aplicação a que são propostos, ou seja, eles não têm o conceito de manipulador em consideração, sendo que, posteriormente, o MoveIt! será responsável por fazer esta tarefa, integrando a biblioteca OMPL. Esta biblioteca é constituída por diversos algoritmos como, por exemplo, *Rapidly-exploring Random Trees* (RRT), RRTStar (RRT*), RRTConnect, *Single-query Bi-directional Lazy collision checking planner* (SBL), *Probabilistic Roadmap Method* (PRM), PRMStar (PRM*), *Bi-directional Transition-based Rapidly-exploring Random* (BiTRRT), *SPArse Roadmap Spanner algorithm* (SPARS), entre muitos outros. Estes algoritmos podem ser caracterizados pelas seguintes propriedades [67]:

- Otimização de solução - O algoritmo procura obter a solução ótima do ponto de vista da distância percorrida, ou seja, a trajetória mais curta. Exemplos: RRT*, PRM*, BiTRRT e SPARS;
- Multiconsulta - O algoritmo procura analisar todo o espaço disponível, mesmo não necessitando de todo esse conhecimento para solucionar o problema. Em ambientes estáticos, esta característica é valorizada, pois o planeador conhece todo o espaço. No entanto, caso o espaço seja dinâmico, os tempos de computação podem tornar-se desajustados. Exemplos: PRM, PRM* e SPARS;
- Objetivo invariante no tempo - O algoritmo é caracterizado por tentar encontrar a solução o mais rapidamente possível, mesmo que contenha movimentos desnecessários. Ou seja, não se preocupa por otimizar a solução encontrada. Exemplos: RRT, RRTConnect e BiTRRT;

Além disso, os planeadores com prefixo "Bi" e o RRTConnect, são chamados de versões bidireccionais das versões originais dos algoritmos primeiramente desenvolvidos. Estas versões são mais rápidas, estando sempre dependentes da capacidade do computador em executar vários processos em paralelo. Na prática, sucede que estas versões fazem a exploração das árvores em paralelo, onde uma começa no ponto inicial e outra no ponto objetivo, e a solução é encontrada quando as duas se conectam [68]. Concluindo, todos os planeadores funcionam de forma diferente e com distintos desempenhos, sendo que depende sempre do manipulador e do ambiente em que os manipuladores operam [67].

3.5.2 CHOMP

O CHOMP é um planeador de otimização de trajetórias que cria uma trajectória inicial ingénua que serve de ponto de partida e que depois é otimizada. O processo de otimização recorre a um método semelhante ao gradiente covariante descendente dedicado ao planeamento de trajetórias para manipuladores, o que torna a problemática de otimização do planeamento simples e rápida [64]. Quando submetido a uma trajetória com obstáculos, o CHOMP reage ao ambiente circundante de forma a calcular rapidamente a trajetória de colisão, enquanto, simultaneamente, procura otimizar a função de custo definida com parâmetros ao nível das velocidades, acelerações, energia despendida, torque, entre outros. Assim, o método converge rapidamente para uma trajetória ideal local, não só livre de colisões, como também com movimentos relativamente suaves.

Este planeador, no entanto, apresenta uma limitação, a saber, estar sujeito a frequentemente convergir para o mínimo local ideal mais próximo, o que pode levar a que não determine o mínimo global ideal [64]. O STOMP, devido à sua natureza estocástica, ultrapassa essa limitação.

3.5.3 STOMP

O STOMP [65] baseia-se na otimização da trajetórias estocásticas usando a geração de trajetórias ruidosas para explorar o espaço em torno de uma trajetória inicial ingénua. Assim, em cada iteração é produzida uma trajetória atualizada com base numa função de custo. Esta função é avaliada ao nível das colisões, suavidade, energia despendida, torque, entre outros. A taxa de convergência para mínimos locais depende, essencialmente, do desvio-padrão com o qual as trajetórias vizinhas são amostradas.

Alguns dos planeadores de movimento tendem a produzir trajetórias com movimentos repentinos e desnecessários. No entanto, o STOMP, com base neste método, tende a produzir planos de movimento suaves e num curto espaço de tempo [65].

O STOMP, comparativamente ao CHOMP, não necessita do gradiente da função de custo e requer um menor ajuste de parâmetros, o que torna mais fácil encontrar os parâmetros ideias para os requisitos impostos. Além disso, o STOMP, devido à sua natureza estocástica, tem uma menor probabilidade de convergir para um mínimo local que poderá não ser simultaneamente um mínimo global [69].

3.5.4 Comparação planeadores de trajetórias

A partir deste estudo, é possível concluir que existe uma grande variedade de planeadores com características muito distintas, e não é o objetivo desta dissertação analisá-los a todos. Por esse motivo, realizou-se uma pesquisa sobre casos de estudo relacionados com esta temática, de forma a verificar quais são os que apresentam, por norma, os melhores resultados.

Numa das dissertações desenvolvidas no laboratório IRIS desenvolveu-se um teste intensivo a uma grande parte dos planeadores disponíveis da biblioteca OMPL para um robô manipulador construído de raiz [70]. Conclui-se nesse trabalho que o RRTConnect apresenta os tempos de planeamento menores, apresentando uma diminuta percentagem de falhas, mesmo em testes de maior dificuldade, apesar de nem sempre apresentar a trajetória mais direta.

Num outro trabalho desenvolvido no IRIS testou-se também um conjunto de planeadores OMPL com vista a integrar num robô móvel KUKA [71]. Os testes realizados mostram que o RRTConnnect nos testes de menor dificuldade apresentam os melhores resultados em termos temporais e de variações das juntas, no entanto, em tarefas de maior dificuldade o PRM*, com um tempo de planeamento de 0.4 segundos, forneceu os melhores resultados em termos de taxa de sucesso e tempos de planeamento.

Por fim, analisou-se outro artigo em que se realizou testes a vários planeadores também eles da família OMPL, num contexto de um manipulador dedicado a tarefas de agronomia [72]. Conclui-se, nesse trabalho, tendo em consideração a taxa de sucesso, o tempo de planeamento, a distância percorrida e suavidade, o melhor algoritmo foi o BiTRRT.

3.6 Manipulador UR10e

Tendo em conta a disponibilidade de recursos e os requisitos definidos, o manipulador usado nesta dissertação é o UR10e da *Universal Robots* [73]. O UR10e encontra-se no grupo de manipuladores considerados colaborativos, apresentando, por isso, um grande nível de segurança para o operador. Este tipo de manipuladores, dado à presença de sensores estrategicamente alocados, quando detetam uma colisão, entram no modo segurança e param, e por isso não têm a necessidade de se encontrarem fechados numa área reservada.

Tendo em conta a mecânica do robô, este é caracterizado por ser um manipulador de 6 graus de liberdade, onde todas as juntas são rotativas com uma gama de operação ±360 ° e é capaz de fazer forças até 100 N. A sua carga máxima é de 10 kg e a velocidade máxima nas juntas *elbow*, *wrist1*, *wrist2* e *wrist3* é de ±180 °/s enquanto que a da *base* e do *shoulder* é ± 120°/s (Figura 3.6). Fundamentalmente, é constituído por materiais como alumínio, ferro e plástico. Suporta uma alimentação de 24 V a 2 A .

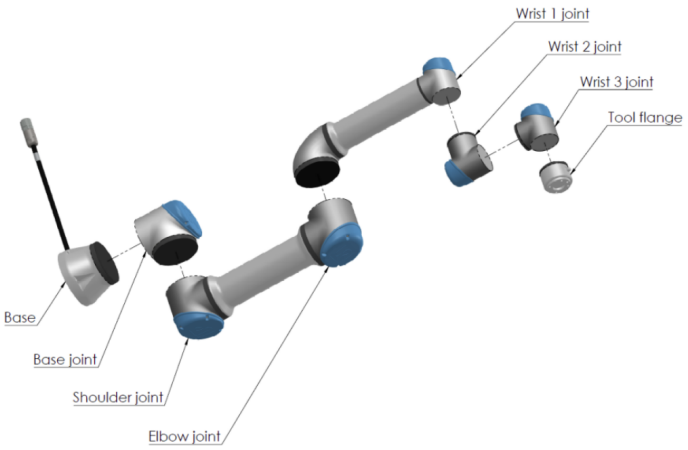


Figura 3.6 – Juntas, base e segmentos do manipulador UR10e da *Universal Robots* [51].

Este tipo de robôs são muito usados na indústria devido à sua versatilidade e programação fácil que pode ser feita através da consola tátil por pessoas com um mínimo de formação do sistema. Além disso, foram projetados para pequenas e médias empresas que necessitem de automatizar de forma flexível e que pretendam um rápido retorno do investimento. Estes robôs são fáceis de deslocar e não necessitam de fixações muito robustas e pesadas devido à sua estrutura de peso reduzido de cerca de 33.5 kg.

3.7 Conclusão

Este capítulo apresenta os vários tipos de manipuladores e garras existentes, o que permite ter uma noção de qual a melhor solução para a situação problema desta dissertação. No que diz respeito ao manipulador robótico, será usado o robô UR10e da *Universal Robots*, por um lado, porque é o robô disponível no laboratório de trabalho e, por outro lado, porque cumpre com as exigências impostas. Além disso, estudaram-se pontos fundamentais da manipulação robótica, nomeadamente, a calibração, a cinemática e os planeadores de trajetórias. Quanto aos planeadores, é difícil concluir com base na literatura qual o melhor método e, por essa

razão, serão realizados testes com vista a aferir qual o melhor planeador no âmbito desta dissertação.

Capítulo 4

ROS

4.1 Conceitos gerais

O *Robot Operating System (ROS)* surge como resposta aos vários desafios de elevada complexidade que os engenheiros na área da robótica têm vindo a enfrentar ao longo dos anos, do ponto de vista da criação de *software*. Assim, o ROS apresenta-se como um *framework* que visa potencializar o desenvolvimento de *software* para o mundo da robótica [74], promovendo uma camada de comunicação de alto nível independente do *hardware*. A comunicação baseia-se essencialmente num modelo cliente-servidor e publicador-subscritor, permitindo a comunicação entre os diferentes programas com fins distintos bem definidos.

Atualmente, o ROS já conta com várias distribuições, que são um conjunto de pacotes ROS com várias versões. O objetivo das distribuições ROS é permitir que os programadores de *software* trabalhem numa base de código relativamente estável, promovendo a fácil implementação nas mais diversas aplicações. As duas distribuições mais recentes são o *Melodic Morenia*, lançada no ano 2018 e voltada principalmente para o *Ubuntu 18.04*, apesar de ter suporte para outros sistemas operativos como o *Windows*, *Mac Os* e *Android*, e o *Noetic Ninjemys*, lançado em 2020, desenvolvido para operar no mais recente *Ubuntu 20.04*.

O ROS assume uma estrutura baseada em alguns conceitos básicos como [75]:

- *node* - Um nó ROS é um processo com uma determinada função, que comunica com outros nós através de mensagens, como tópicos e/ou serviços, como ilustrado na figura 4.1. Cada nó pode subscrever/publicar múltiplos tópicos e fornecer/requerer múltiplos serviços.

- *master* - O ROS *master* é o nó responsável por fornecer serviços de nomenclatura e registo para novos nós individuais. Assim, a função deste nó é permitir que os nós se localizem, dado que eles não poderão comunicar até que o *master* lhes indique a existência de um novo nó. A partir daí, os nós estabelecem uma comunicação ponto a ponto sem a intervenção do nó *master* (Figura 4.1). O *master* também disponibiliza um servidor de parâmetros que é partilhado entre os nós.

- *message* - As mensagens são descritores simplificados dos dados que os nós publicam através dos tópicos do mesmo tipo. A vantagem desta descrição é facilitar a comunicação entre nós de diferentes linguagens, uma vez que os tipos de dados estão bem definidos. A descrição das mensagens é armazenada em ficheiros com a terminação `.msg` e estão divididos em campos e constantes. Os campos são os dados enviados dentro da

mensagem e podem ser, por exemplo, inteiros de 32 bits (*int32*), *string* ou *time*, e as constantes definem valores que podem ser usados para interpretar esses mesmos campos.

- *topic* - Os tópicos são barramentos que permitem o fluxo unidireccional de mensagens entre os vários nós. Tanto o nó destinado à publicação de mensagens como o nó dedicado à subscrição de mensagens necessitam se registar no ROS *master* e, a partir daí, é instanciada uma comunicação ponto a ponto entre os dois nós. Além disso, cada tópico está dedicado a enviar um único tipo mensagem de forma a manter a coerência de comunicação entre o publicador e subscritor.

- *service* - Num sistema distribuído que é o ROS é necessário, por vezes, uma comunicação bidirecional, permitindo assim a comunicação do tipo servidor-cliente. O nó cliente envia uma mensagem de pedido ao nó servidor e aguarda uma resposta. O nó servidor recebe o pedido e, depois da conclusão da tarefa pedida, envia uma mensagem de resposta ao nó cliente. Os tipos de mensagem de pedido e resposta de um serviço são definidos através de um ficheiro com a terminação `.srv`, utilizando os mesmo tipos de mensagens que os tópicos.

- *actions* - Tal como os *services*, assenta numa comunicação cliente-servidor, as ações permitem enviar uma solicitação a um nó para realizar uma determinada tarefa. A vantagem assenta no facto de poder obter um *feedback* periódico, possibilitando o cancelamento da solicitação. Este é um método de comunicação não nativo, uma vez que pertence à biblioteca *actionlib*.

- *parameter server* - Um servidor de parâmetros é, na pratica, um servidor habilitado a armazenar e devolver parâmetros durante a execução dos vários nós ROS, sendo que o servidor é executado dentro do ROS *master*. Este pode ser designado como um dicionário compartilhado que pode ser acedido por meio de APIs de rede.

- *bags* - São ficheiros que permitem guardar, de forma compacta, as mensagens publicadas pelos tópicos durante a execução de um sistema ROS. É permitido que os ficheiros sejam editados e reproduzidos através da ferramenta *rosbag*.

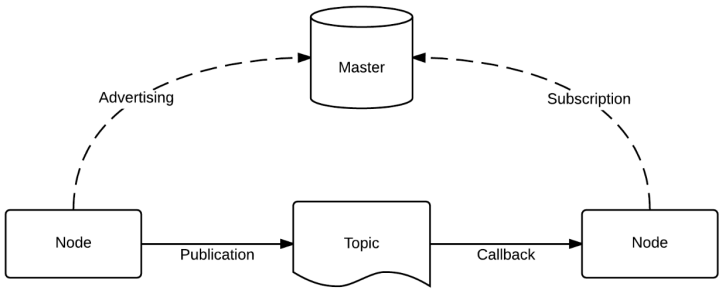


Figura 4.1 – Esquemático geral do ROS [76].

O ROS é também conhecido pela sua grande versatilidade no que diz respeito às linguagens de programação que suporta. Esta característica dá margem aos utilizadores em relação à escolha da linguagem que melhor se adequa aos seus propósitos, quer seja por questões de sintaxe, eficiência temporal, complexidade ou bibliotecas já existentes.

É importante ressalvar que o *software* no ROS é organizado sobre a forma de pacotes (*packages*). Estes pacotes podem incluir conjuntos de dados, ficheiros de configuração, nós ROS ou qualquer outro módulo que seja útil. Com este tipo de organização, pacotes são facilmente desenvolvidos e reutilizáveis, permitindo conjugar diferentes pacotes se assim for necessário. A criação dos mesmos pode passar por um processo manual ou usar a ferramenta `catkin_create_pkg`.

Atualmente, o ROS suporta algumas linguagens muito diferentes: C++, Python e LISP. Esta facilidade de implementação multi-linguagem ocorre graças ao descritor de interfaces *Interface Definition Language* (IDL), usado para descrever as mensagens enviadas entre módulos, através de ficheiros com campos bem definidos.

O código fonte do ROS está disponível publicamente e sem qualquer tipo de restrições, uma vez que é distribuído nos termos da Licença BSD, que permite o desenvolvimento de projetos não comerciais e comerciais [77].

4.2 Ferramentas e pacotes de desenvolvimento

4.2.1 Rviz

O RViz (ROS visualization) é um programa dedicado à visualização de dados provenientes de tópicos e parâmetros ROS, quer sejam 2D ou 3D. Assim, o Rviz pode ter dados de diversas fontes como modelos de robôs e respetivos estados como, por exemplo, a posição das juntas, transformações 3D do robô, ou ainda dados de diferentes sensores como, por exemplo, uma nuvem de pontos com origem num laser ou imagens RGB de uma câmara RGB, entre outros. Do ponto de vista dos planeadores para os manipuladores, é ainda possível visualizar a trajetória planeada pelo planeador antes de executar, entre outras funcionalidades. Além da visualização, o RViz usa algumas bibliotecas do ROS que permitem a manipulação do ambiente.

Como é possível verificar na figura 4.2, o RViz pode recorrer a outras ferramentas de desenvolvimento, neste caso, o MoveIt!. Na figura é ilustrado o estado atual do robô Panda, assim como uma representação da posição objetivo da manipulação. A manipulação poderá ser efetuada diretamente através do RViz.

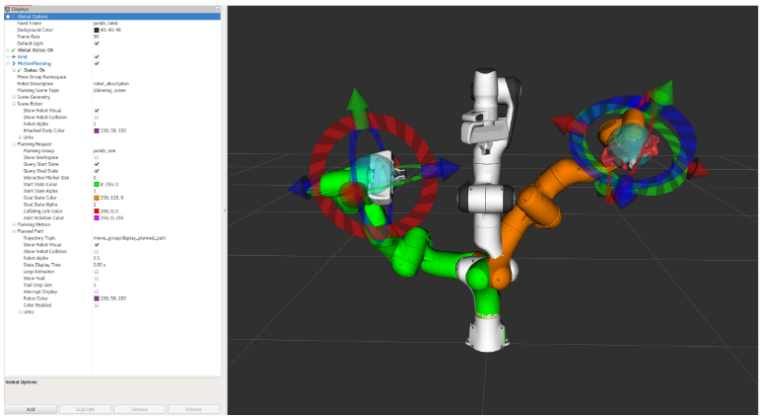


Figura 4.2 – Visualização do planeamento de um movimento do robô Panda usando a GUI do MoveIt! [78].

4.2.2 rqt

O rqt é um *framework* do ROS multiplataforma para desenvolvimento de interfaces gráficas em C++, baseada em Qt. O rqt implementa várias ferramentas de GUI na forma de *plugins*, permitindo aos próprios utilizadores criarem os seus próprios *plugins* consoante as suas necessidades. Na prática, o rqt organiza múltiplas ferramentas de GUI numa só janela (`rqt_gui`), facilitando a usabilidade, do ponto de vista do utilizador, em sistemas de maior complexidade, como ilustrado na figura 4.3.

Neste momento já existem diversos *plugins* rqt disponíveis como, por exemplo, as interfaces gráficas para a visualização de tópicos (`rqt_topic`), ações (`rqt_action`) e serviços (`rqt_services`), interfaces para a gestão de ficheiros *bags* (`rqt_bag`), interfaces de visualização de tópicos e/ou serviços resultado de captura de imagens (`rqt_image_view`), ou ainda uma interface dedicada à visualização de uma diagrama geral de todos os nós ROS, assim como os respetivos tópicos e/ou serviços.

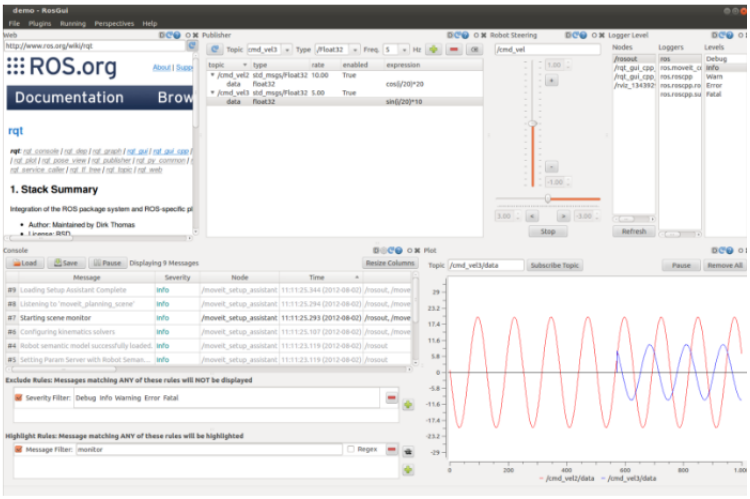


Figura 4.3 – Janela do rqt_gui incorporando diferentes *plugins* [79].

4.2.3 URDF

O ROS recorre ao pacote *Robot Description Format* (URDF) para descrever a geometria do robô [80], mais concretamente ao formato *eXtensible Markup Language* (XML), de forma a especificar os modelos relacionados com o robô propriamente dito, sensores e cenários.

A descrição do robô é composta essencialmente por segmentos (*links*) e juntas (*joints*) [81] detalhadamente descritas tanto do ponto do vista geométrico como na hierarquia que as relaciona. As juntas têm a função de interligar os segmentos e apresentam um modelo hierárquico rigoroso assente numa relação pai-filho, onde cada *link* tem um único pai (Figura 4.4).

Neste formato, um segmento pode ser encarado como um elemento isolado e, como tal, apresenta o seu próprio sistema de eixos (*frame*) que pode ser facilmente referenciado e relacionado com outros sistemas de eixos, através de uma simples translação e/ou rotação. Uma peça pode ser representada por 3 componentes: visual, colisão e inércia e, para cada uma, está associado um ponto de origem (*origin*) que representa tanto a posição como a orientação, sendo que cada uma das componentes pode não coincidir necessariamente (Figura 4.4).

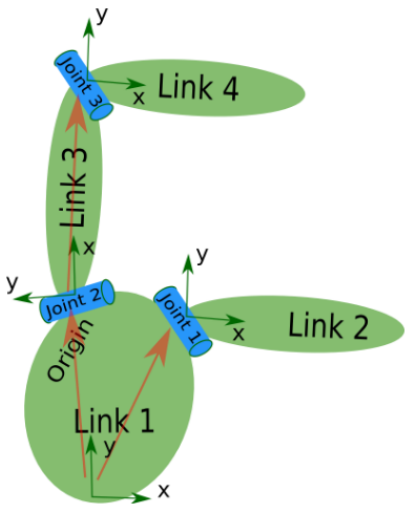


Figura 4.4 – Ilustração exemplo da relação existente entre um conjuntos de *links* conectadas por *joints* [82].

Relativamente à componente visual, é apenas necessária para efeitos de visualização, logo é onde é associado o modelo tridimensional minucioso, não só com as respetivas dimensões, como com o material usado e respetiva textura. No que diz respeito à componente da colisão, são determinados os coeficientes de fricção, rigidez e amortecimento do segmento, sendo nulos por defeito. Em relação ao movimento das juntas, estas definem 6 tipos, que se distinguem pelos seus graus de liberdade de movimento relativamente aos elemento hierárquico mais elevado. Os diferentes tipos são: *revolute*, *continous*, *prismatic*, *floating*, *planar* e *fixed* [83].

Do ponto de vista computacional, a simulação da colisão pode ser bastante exigente para modelos altamente complexos e, por essa razão, na maioria das situações, os modelos de colisões do robô são aproximados por um conjunto de figuras geométricas primitivas, como paralelepípedos e esferas. No entanto, caso se pretenda obter um modelo mais detalhado é também possível usar *meshes*, onde se encontra a descrição do modelo 3D do objeto criado.

Por fim, na componente referente à inércia, é descrita a resistência da peça a variações de velocidade de translação e rotação, ou seja, a massa e a matriz de inércia rotacional do segmento. A transmissão é uma extensão do modelo descritivo do robô que permite descrever a relação entre um atuador e uma junta.

Do ponto de vista da reutilização e ajustes de código, o URDF apresenta grandes limitações, não sendo sequer possível definir variáveis nem criar elementos modulares facilmente reutilizáveis, o que pode ser bastante penoso em projetos de elevada complexidade. Assim sendo, surgiu a linguagem xacro (XML *Macros*) que visa colmatar estas limitações.

4.2.4 xacro

A xacro (XML *Macros*) é um pacote que permite descrições de modelos URDF de uma forma mais modular e reutilizável. Como o próprio nome indica, viabiliza a construção de arquivos XML mais curtos e legíveis, usando *Macros* e variáveis. Este pacote é tanto mais útil quanto maiores e mais complexos forem os documentos XML como, por exemplo, em casos de manipuladores bastante complexos e com um número significativo de graus de liberdade.

Assim, o ROS inclui este pacote onde reside um programa em Python responsável por

traduzir ficheiros XML, que pode ser interpretado num ficheiro URDF se as *tags* aplicadas assim o permitirem, não necessitando de qualquer edição.

4.2.5 Transformações ROS

A tf é um biblioteca que permite ter controlo sobre os vários sistemas de coordenadas dos *frames* ao longo do tempo, e foi projetada com o objetivo de encontrar um procedimento padrão que permita o controlo de todos os *frames* sem exigir o conhecimento de todo o sistema, uma vez que existe um relação bem explícita entre eles [84].

Num sistema robótico podem-se encontrar diversos *frames* de coordenadas 3D que se relacionam e que variam ou não ao longo do tempo (Figura 4.5) como, por exemplo, a *frame* do mundo em que está o sistema (*world*), a *frame* associada à base do robô (*base_link*) ou a *frame* localizada na garra do manipulador (*gripper_link*), entre outras. Desta forma, é possível ter um controlo total de todos os *frames*, permitindo transformar coordenadas entre os diferentes referenciais.

Esta biblioteca surge devido à necessidade fundamental do robô, que necessita de saber onde está, bem como onde o resto dos seus constituintes estão em relação a si mesmo. No caso prático desta dissertação, um dos desafios passa por saber a localização de um objeto relativamente à garra, de forma a ser possível manipulá-la. Com o uso de uma câmara e assumindo que o robô sabe as coordenadas do *frame* do sensor pode-se, a partir daí, saber a relação das coordenadas do objeto relativamente à câmara e, consequentemente, do objeto relativamente ao robô, permitindo a precisa manipulação do objeto.

Assim, à medida que os sistemas robóticos vão ficando cada vez mais complexos, dado que conjugam dados de sensores externos, os sistemas de coordenadas de *frames* tornaram-se críticos e imprescindíveis. Logo, a biblioteca tf foi desenvolvida como um pacote ROS para fornecer essa relevante capacidade.

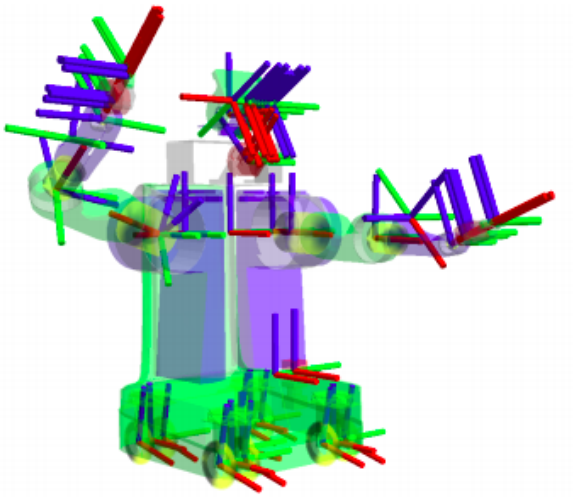


Figura 4.5 – Visualização de todos os *frames* tf do robô PR2 [84]. Os cilindros vermelhos, verdes e azuis representam os eixos X, Y e Z respetivamente.

4.2.6 Conversão de mensagens ROS para OpenCV

O *vision_opencv* é um pacote ROS que têm disponível um conjunto bastante diversificado de bibliotecas do OpenCV dedicadas à visão computacional [85]. Dentro deste pacote, estão disponíveis diversos pacotes:

- *cv_bridge* - responsável por fazer a conversão de mensagens ROS em imagens formatadas para o OpenCV, como ilustrado na figura 4.6;

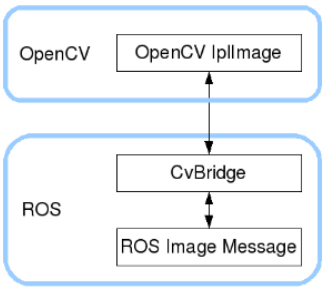


Figura 4.6 – Esquema geral da comunicação entre o ROS e o OpenCV [85].

- *image_geometry* - contém bibliotecas Python e C++ que simplificam a interpretação de imagens do ponto de vista geométrico, usando os parâmetros disponibilizados pelos sensores usados como, por exemplo, o *CameraInfo*, no caso de se usar uma câmara.

4.3 Controlo de manipuladores

O controlador de trajetórias recebe o pedido de uma determinada trajetória e verifica a validade da mesma, tendo em conta as limitações físicas, e se o ponto inicial corresponde à configuração atual do manipulador, assumindo sempre uma tolerância devido aos ruídos existentes.

O ROS apresenta um conjunto de bibliotecas incorporadas no pacote *ros_control* que foi desenvolvido com o intuito de facilitar a implementação e reaproveitamento de controladores para robôs. Este pacote já inclui controladores que dão garantias para a maioria das aplicações, no entanto, é possível parametrizar o controlador ou desenvolver o próprio controlador.

Na prática, o controlador toma conhecimento da trajetória solicitada, assim como o estado das juntas fornecido pelos *encoders* do atuador, e faz um controlo individual das juntas de forma a deslocar-se, com um movimento o mais suave possível, para os pontos definidos pela trajetória. Este processo recorre a um mecanismo de *feedback* de controlo genérico, normalmente usando um controlador *Proportional Integral Derivative* (PID) para o controlo dos atuadores [86].

4.3.1 Trajetórias ROS

As trajetórias no ROS são definidas como um conjunto de pontos representados por mensagens do tipo *JointTrajectory*, que definem a posição, velocidade, aceleração e torque de uma ou mais juntas de um manipulador. Estas mensagens têm a seguinte estrutura:

- `std_msgs/Header` header;
- `string[ ]` joint_names;
- `trajectory_msgs/JointTrajectoryPoint[ ]` points.

Informações gerais sobre o sistema, por exemplo, data da criação da mensagem, estão alocadas no campo *header*, o nome das juntas que fazem parte da trajetória são armazenados no campo *joint_names* e os pontos resultantes da mesma são guardados no campo *points*, sendo que estes constituem um tipo de mensagem chamado *JointTrajectoryPoint* e têm a seguinte estrutura:

- `float64 [ ]` positions;
- `float64 [ ]` velocities;
- `float64 [ ]` accelerations;
- `float64 [ ]` effort;
- `duration` time_from_start;

Os quatro primeiros campos são responsáveis pelas posições, velocidades, acelerações e torques das próprias juntas. O campo *time_from_start* representa o tempo, a que os valores desse ponto são atingidos, relativamente ao início da trajetória.

4.4 Simuladores

No mundo da investigação, e mais concretamente na área da robótica, os simuladores têm ganho cada vez mais relevo, uma vez que se tornaram numa ferramenta rápida, eficiente e barata, permitindo exaustivos testes de validação de algoritmos, métodos e arquiteturas.

A grande vantagem dos simuladores reside na estimativa do desempenho de um robô real, sem qualquer tipo de riscos materiais ou pessoais associados, além que os testes são realizados mais rapidamente sem a exigência de um *hardware* específico. Apenas é necessário ter em conta o compromisso entre o poder computacional exigido e a precisão dos fenómenos físicos simulados. Todos os benefícios resultam numa inigualável contribuição para o desenvolvimento progressivo acelerado do mundo da robótica [87]. De uma forma simplificada e ligeira, podemos caracterizar um simulador, por um lado, pela capacidade de simular visualmente o ambiente simulado tendo em conta as dimensões, formas, cores, sombras e textura, ou seja, a renderização gráfica, e por outro lado, a capacidade de representar as leis da física inerentes ao mundo real. No que diz respeito ao motor de física, normalmente tem em conta as seguintes grandezas: velocidade, inércia, atrito, posição, orientação, etc.

Outros aspetos a ter em conta são a compatibilidade multi-plataforma e a flexibilidade de adicionar dispositivos sensoriais incorporados com algoritmos de mapeamento e navegação 3D ou reconhecimento de objectos 2D e 3D. A questão sensorial é um fator importante para o desenvolvimento de algoritmos de percepção robótica, uma vez que muitas das vezes é necessário adquirir dados visuais através de sensores também eles virtuais, como é o caso de câmaras virtuais, que usam o mecanismo de renderização do simulador para obter as imagens virtuais do ambiente simulado.

4.4.1 Gazebo

O Gazebo é um simulador 3D, de acesso livre, que acomoda, com uma grande viabilidade, todos os aspetos anteriormente mencionados. Como esperado, apresenta a possibilidade de integração multiplataforma e, em especial, tem uma forte agregação com o ROS. Por esse motivo, apesar de não ser o único ou melhor simulador da atualidade, acaba por ser o simulador de eleição quando se trata de sistemas baseados em ROS [88].

O Gazebo usa mecanismos externos de renderização *Open Source 3D Graphics Engine* (OGRE), que, por um lado, produzem uma boa fidelidade gráfica nas simulações mais exigentes, apresentando vários mecanismos de física de alto desempenho como *Open Dynamics Engine* (ODE), *Dynamic Animation and Robots Toolkit* (DART), Bullet e Simbody e, por outro lado, usam o *Open Graphics Library* (OpenGL) no desenvolvimento gráfico da aplicação [89]. Além disso, admite incluir nas suas bibliotecas robôs bastante usuais como o PR2, UR10, Pioneer2 Dx, TurtleBot ou então simular robôs criados de raiz [90] através do formato *Simulation Description Format* (SDF) ou URDF. Além disso, o Gazebo tem a capacidade de carregar *plugins* no servidor do Gazebo de forma a configurar a simulação consoante as necessidades. Estes permitem integrar novas interfaces de comunicação entre os modelos e o exterior do servidor, incluindo mais sensores e interfaces de controlo para as juntas.

A interface gráfica do Gazebo pode ser visualizada na figura 4.7, onde de encontram no mundo (*world*) do simulador vários modelos (*models*) como o do robô UR10e, uma mesa e diversificados objetos. As componentes dos modelos podem agrupar-se em 3 distintas categorias como corpos, juntas e sensores [89].

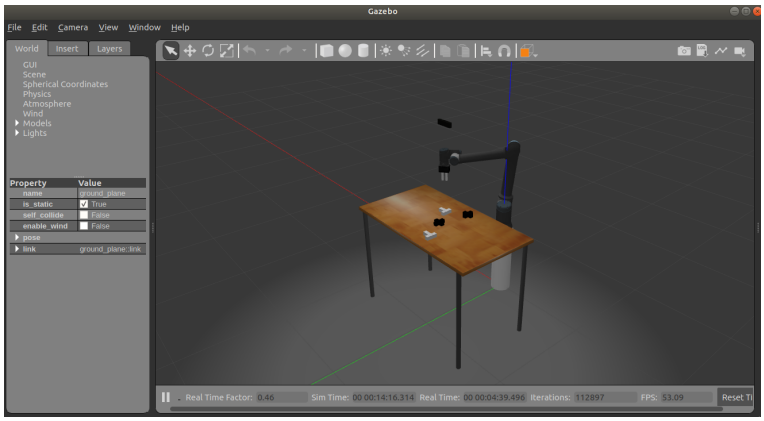


Figura 4.7 – Interface gráfica do simulador Gazebo, com uma mesa, uma mesa e diversificados objetos.

SDF

Como referido anteriormente, o URDF é o descritor padrão do ROS, no entanto, este apresenta algumas limitações, uma vez que não foi sofrendo as devidas atualizações conforme a evolução do mundo da robótica. Assim, o URDF só pode especificar as propriedades cinemáticas e dinâmicas de um único robô de forma isolada, além de que, não pode especificar a posição do próprio robô dentro do mundo. Para lidar com esse problema, o Gazebo optou por usar como descritor padrão o formato *Simulation Description Format* (SDF) de forma a ultrapassar as limitações do URDF.

SDF é um formato que permite a descrição dos robôs e dos próprios ambientes simulados em que se enquadram, sendo que essa descrição pode estar num só ficheiro. Além disso, este formato usa também a linguagem XML, numa descrição baseada em *tags*, com todas as vantagens inerentes anteriormente apresentadas.

Relativamente ao mundo propriamente dito, existem descritores que permitem controlar e simular fenómenos naturais como condições atmosféricas, vento, gravidade e campo magnético. Assim, o próprio mundo Gazebo pode ser guardado, editado e carregado através de um ficheiro SDF. No que diz respeito ao segmentos (*links*) do robô, existem *tags* referentes à descrição das 3 principais componentes: visual, colisão e inércia. Além disso, ainda existem *tags* dedicadas à descrição da interface gráfica do Gazebo, o que se torna relevante quando é pretendido especificar a interface e a posição de um sensor como, por exemplo, uma câmara. No que se refere à aquisição de dados dos sensores, a simulação é realizada através de *plugins* e os dados propriamente ditos são extraídos dos tópicos.

Gazebo em ambiente ROS

No ROS, quando o gazebo se inicia, implementa dois executáveis, o *gzserver* e o *gzclient*. Enquanto que o *gzserver* executa continuamente o motor de física e a gestão dos dados dos sensores, o *gzclient* executa uma interface de utilizador baseada em Qt [91]. Depois de iniciar o servidor, é possível adicionar objetos e robôs à simulação através do tópico `/gazebo/set_model_state` ou do nó *spawn_model* do pacote *gazebo_ros*. Este nó pode ser utilizado para enviar ao servidor do Gazebo uma mensagem, para o tópico `/gazebo/set_model_state`, com um objeto existente na base de dados do Gazebo ou num ficheiro de descrição SDF ou URDF. O Gazebo permite o controlo das juntas dos robôs através de comandos diretos sobre a posição, velocidade ou torque, contudo, este não disponibiliza controladores. Por essa razão, é comum encontrar na literatura o recurso ao pacote *ros_control* para esse efeito [92].

O *plugin libgazebo_roscontrol.so* permite utilizar os controladores do *ros_control*, sendo que a comunicação com a simulação faz-se através de memória partilhada e não através de tópicos. Este *plugin* instancia o *Controller Manager* e fornece as interfaces, com o *hardware* de posição (*PositionJointInterface*), velocidade (*VelocityJointInterface*) e torque (*EffortJointInterface*), necessárias para os controladores interagirem com o robô simulado. Este *plugin* tem acesso à descrição URDF do robô, que se encontra, por defeito, no parâmetro ROS `/robot_description`. Esta descrição recorre a *tags* `<transmission>` de modo a informar que tipo de interfaces com o hardware é que cada junta utiliza.

4.5 MoveIt!

O MoveIt! é um conjunto de pacotes integrados com o ROS que apresentam funcionalidades muitos específicas como, por exemplo, obter cinemática inversa, planeamento de movimentos, verificação de colisões com objetos dinâmicos e estáticos, planeamento operações de agarrar e largar objetos, entre outras [78].

O MoveIt! fornece uma GUI chamada *Setup Assistant*, para gerar todos esses elementos. De um modo geral, essa ferramenta gera o *Semantic Robot Description Format* (SRDF) e os arquivos de configuração e inicialização que são necessários para configurar o nó *move_group*. O arquivo SRDF contém detalhes sobre as juntas do braço, juntas da garra, juntas virtuais e os pares de *links* de colisão. O arquivo de configuração contém detalhes sobre os algoritmos cinemáticos, limites das juntas, controladores, entre outros. Usando o pacote de configuração

gerado pelo robô, pode-se, não só, gerar planeamentos de movimento, como controlar diretamente o manipulador através do RViz, sem a presença de um robô real ou interface de simulação (Figura 4.2).

O servidor de parâmetros é responsável por armazenar os dados cinemáticos do robô, o URDF, o SRDF e os ficheiros de configuração (Figura 4.8). Este servidor dá as informações necessárias ao MoveIt! sobre o manipulador e as respetivas configurações, de forma a que seja possível iniciar a componente de controlo. O controlo do robô é realizado através das várias interfaces disponíveis. Pode-se usar as interfaces gráficas existentes em C++ (*move_group_interface*) ou Python (*moveit_commander*) para enviar comandos para o nó (Figura 4.8), ou então poder-se-á optar por usar o *plugin* de planeamento do Rviz, que tira partido da própria interface gráfica do Rviz (Figura 4.2) [78]. Após o planeamento do movimento, o nó é ainda responsável por enviar, através de ações, a trajetória a executar pelo o robô, que será recebida pelos controladores do mesmo.

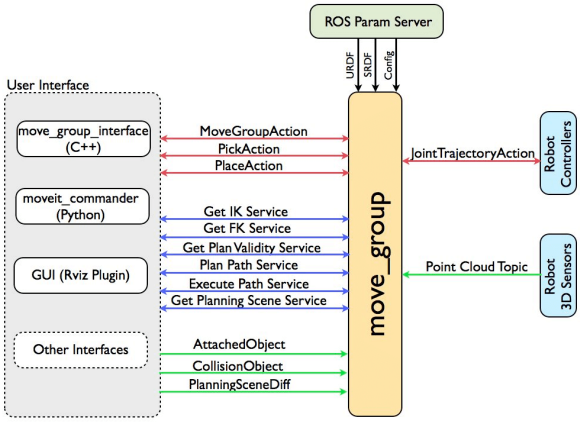


Figura 4.8 – Esquemático geral do nó ROS do MoveIt! [78].

Na prática, o *move_group* é responsável por integrar as várias componentes do robô que sejam necessárias, e concede uma interface de comunicação com o utilizador baseada em tópicos, serviços e ações (Figura 4.8). Este, não é responsável por nenhum tipo de algoritmo de planeamento de movimentos ou de cinemática, pois apenas se encarrega de interligar todas as funcionalidades disponíveis sobre a forma de *plugins*. Desta forma, podemos usar qualquer planeador de movimento simplesmente alterando o *plugin* e impondo as devidas restrições cinemáticas, tais como: restrições de posição e orientação dos *links*, restrições de juntas, entre outras. Uma vez definidas as restrições, é solicitado o planeamento e geradas as trajetórias requisitadas no nó *move_group* que, por sua vez, serão enviadas para os controladores de trajetória. Além disso, é necessário especificar os grupos de juntas que se pretende controlar de forma conjunta e os grupos de segmentos que formam as extremidades dos manipuladores.

O MoveIt!, tal como nos planeadores, usa *plugins* para a componente da cinemática. A cinemática direta está integrada na própria classe *RobotState* e o *plugin* padrão de cinemática inversa pertence à biblioteca Orocos *Kinematics and Dynamics Library* (KDL). Este *plugin* é configurado automaticamente pelo *MoveIt Setup Assistant*.

A verificação de colisão é uma das tarefas mais exigentes a nível computacional, portanto, para simplificar esse cálculo, o MoveIt! fornece uma matriz chamada ACM (Allowed Collision Matrix), onde são definidos os segmentos que estão sempre em colisão e os que nunca podem colidir, de forma a reduzir a quantidade de cálculos na verificação de colisões [78].

4.5.1 Cinemática Inversa e planeadores de trajetórias do MoveIt!

O MoveIt! promove a utilização de vários algoritmos de cinemática inversa e de planeadores de trajetórias de uma forma fácil e intuitiva, integrando-os nos mais diferentes manipuladores.

Relativamente aos algoritmos de cinemática inversa, a biblioteca KDL é a mais requisitada no ambiente ROS [93] e é este o algoritmo predefinido. Outro algoritmo disponível é o *Sequential Quadratic Programming* (SQP) que, por norma, tende a apresentar tempos de convergência superiores [93]. Existe ainda um algoritmo chamado IKFast que resolve as equações cinemáticas de um manipulador, analisando a sua descrição URDF, que apresenta, por norma, tempos de computação mais rápidos que o método KDL [94].

Quanto aos planeadores de trajetórias, o MoveIt! foi projetado de modo a integrar diferentes tipos de algoritmos, o que permite compará-los tendo em conta diferentes propósitos. Existem inúmeros planeadores disponíveis e os mais usuais são: o OMPL [63], CHOMP [64], STOMP [65] e o SBPL [66].

4.6 Conclusão

Neste capítulo explorou-se, de uma forma ampla, todos os tópicos fundamentais do ROS, nomeadamente, o modo de funcionamento de uma forma generalizada, as ferramentas de desenvolvimento disponíveis, a importância e os pontos fundamentais que caracterizam um simulador, assim como a parte de controlo, e os planeadores de trajetórias usando o MoveIt!. Dadas as motivações referidas neste capítulo, usar-se-á o Gazebo para as simulações de teste iniciais, de forma a, posteriormente, aplicar-se as metodologias propostas em ambiente real. Além disso, o estudo dos planeadores anteriormente mencionados será realizado usando os planeadores disponíveis no MoveIt!.

Capítulo 5

Metodologia e resultados no reconhecimento de objetos

Neste capítulo é apresentado o trabalho prático direcionado para o subsistema da visão, responsável pelo reconhecimento das peças, começando por introduzir em que âmbito este capítulo se enquadra e o material a ser utilizado. Ao longo de todo o capítulo são apresentadas e comparadas diferentes soluções para os diferentes objetivos e problemas que foram enfrentados no decorrer de todo o trabalho, assim como as discussões dos resultados obtidos.

Ao longo do capítulo referir-se-á imagem original como a imagem que é capturada a cada instante pela câmara e imagem modelo (*template*) como a imagem que já está previamente armazenada na base de dados. Além disso, quando se menciona objetos e peças, está-se a referir a mesma entidade, ou seja, aos itens que se pretendem identificar e manipular.

5.1 Introdução

O presente capítulo foca-se no desenvolvimento de soluções para o subsistema responsável pela identificação das diferentes peças. Segundo os requisitos apresentados na secção 1.2, pretende-se que a estação de identificação seja capaz de identificar as peças no tapete transportador e de fornecer ao subsistema de manipulação robótica as localizações e as orientações, assim como as dimensões das mesmas e as respetivas coordenadas para a gravação a laser. De forma a cumprir os requisitos impostos, o subsistema deve ser constituído por, essencialmente, uma câmara RGB, um tapete transportador, um dispositivo de processamento e as respetivas peças (Figura 5.1).

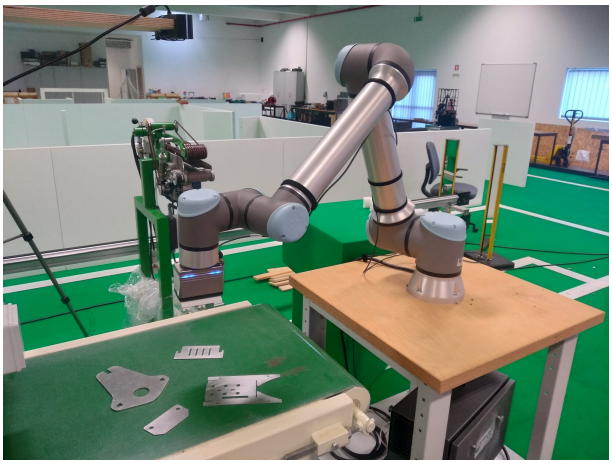


Figura 5.1 – Configuração geral do sistema, constituído por uma câmara, um tapete rolante, um manipulador robótico e os respetivos objetos.

Relativamente ao sensor, teve-se também em consideração o material disponível no *Intelligent Robotics and Systems* (IRIS) *Laboratory*, optando-se pela câmara Orbbec Astra RGB-D (Figura 5.2). Em termos de especificações técnicas, têm uma resolução de 640x480 píxeis e têm a capacidade de captar 30 *fps* que se acreditam ser suficientes. Esta câmara apresenta-se como um opção válida dadas as suas características, pois apenas é necessário a componente RGB. Usou-se o *driver* presente no pacote ROS chamado *astra_camera* [95].

Figura 5.2 – Câmara Orbbec Astra RGB-D.

Relativamente ao processamento dos dados, desenvolvido em Python3, testou-se toda a dissertação num computador pessoal com as seguintes características:

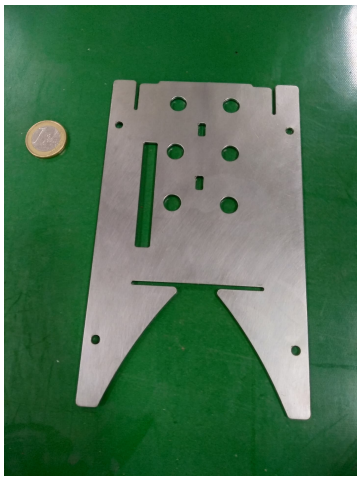
- CPU: Intel Core i7-7500U, 2.7 GHz, 2 núcleos físicos e 2 núcleos virtuais;
- Memória RAM: 8 GB;
- GPU: NVIDIA GeForce 930MX 2 GB.

Quanto às peças que se pretendem manipular, estas apresentam-se com formatos e dimensões bastante diversificados, como é possível verificar na figura 5.3. Atribuiu-se a cada peça um identificador numérico de 1 a 7 e, ao longo deste capítulo e do capítulo 6, as referências às peças seguirão esta nomenclatura.

(a) Peça Nº1

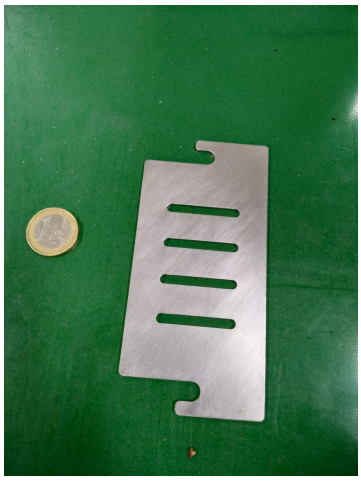
(b) Peça Nº2

(c) Peça Nº3

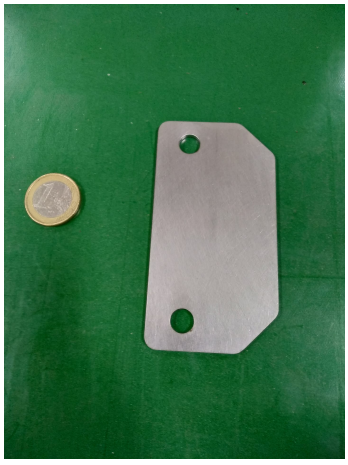


(d) Peça Nº4

(e) Peça Nº5



(f) Peça Nº6



(g) Peça Nº7

Figura 5.3 – Peças usadas no desenvolvimento da dissertação.

5.2 Arquitetura proposta

A figura 5.4 apresenta o esquema geral proposto na preparação da base de dados local usada onde, para cada é capturada uma imagem que depois de processada e segmentada, são armazenadas as *features*, a área, a imagem processada, as coordenadas da marcação a laser, fornecidas pelo utilizador, e uma variável booleana que indica a simetria da peça.

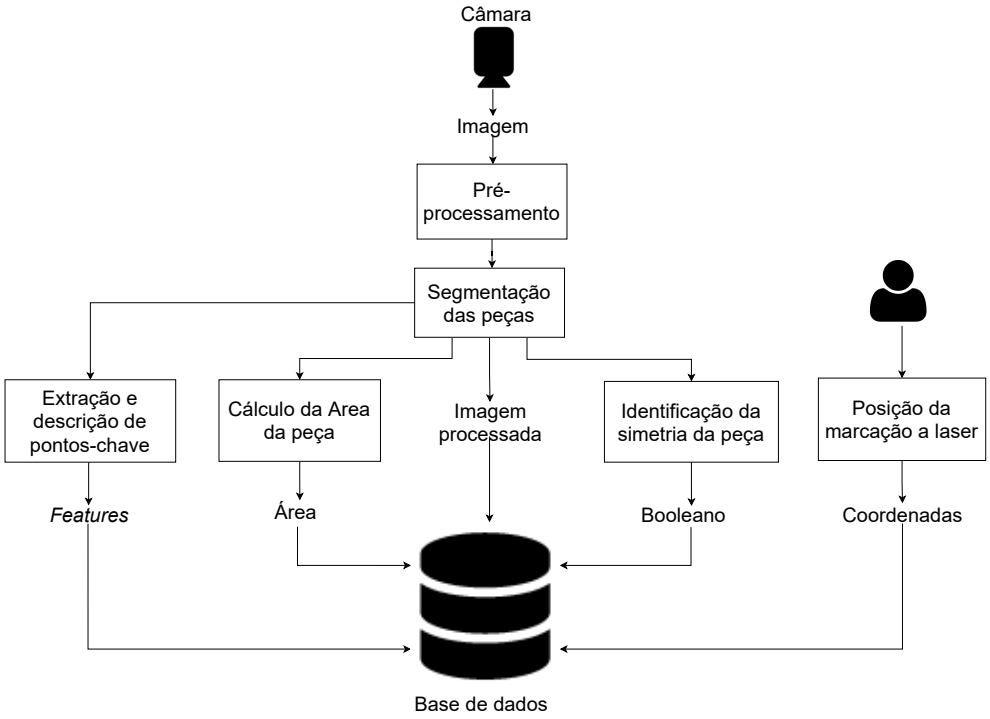


Figura 5.4 – Esquema geral da preparação da base de dados.

Tendo em consideração o sistema geral apresentado na figura 1.1, a arquitetura geral proposta para o subsistema de reconhecimento de objetos é apresentada na figura 5.5. Neste subsistema, uma imagem da peça é capturada, por uma câmara, e de seguida, passa por um conjunto de algoritmos de pré-processamento e segmentação, para que depois sejam extraídas as suas *features*, assim como, calculada a respetiva área, em píxeis. De seguida, é comparada com as peças que estão previamente armazenadas na base de dados e, caso não se encontre uma correspondência, repete-se o procedimento, de modo a que a câmara capture uma nova imagem. No entanto, caso se encontre a peça correspondente, é estimada a posição, a orientação e a dimensão desta. Sabendo estas informações juntamente com as coordenadas da marcação a laser, é possível ou manipular a peça de modo a disponibilizá-la ao marcador a laser corretamente, ou fazer a própria marcação com o manipulador.

A figura 5.6, ilustra a comunicação que existe entre os dois subsistemas, através de 2 nós ROS, representados pelos círculos, e os respetivos tópicos de comunicação representados pelas setas. Os tópicos enviam a informação através de mensagens do pacote *geometry_msgs* e informam o subsistema de manipulação sobre a localização, a dimensão e a orientação da peça, assim como as respetivas coordenadas para a marcação a laser. A arquitetura proposta relativamente ao subsistema de manipulação será apresentada no capítulo 6.

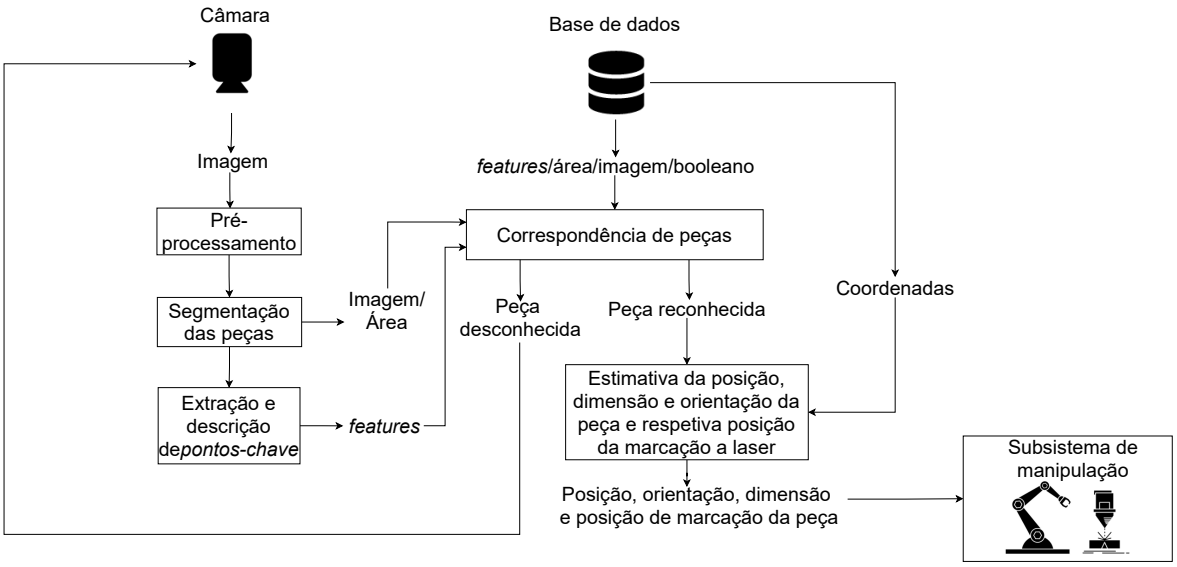


Figura 5.5 – Esquema geral do subsistema de visão.

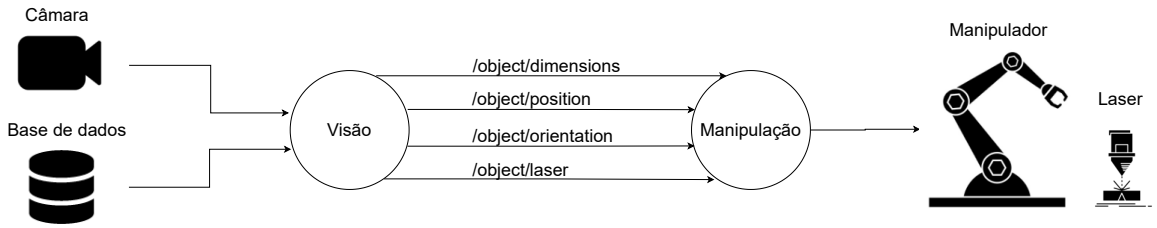


Figura 5.6 – Ilustração da comunicação entre os dois subsistemas.

5.3 Calibração dos parâmetros intrínsecos

Como se verificou na secção 2.3 o processo de calibração da câmara tem como objetivo permitir a um sensor ótico projetar pontos tridimensionais em pontos bidimensionais, numa imagem, com o menor erro possível.

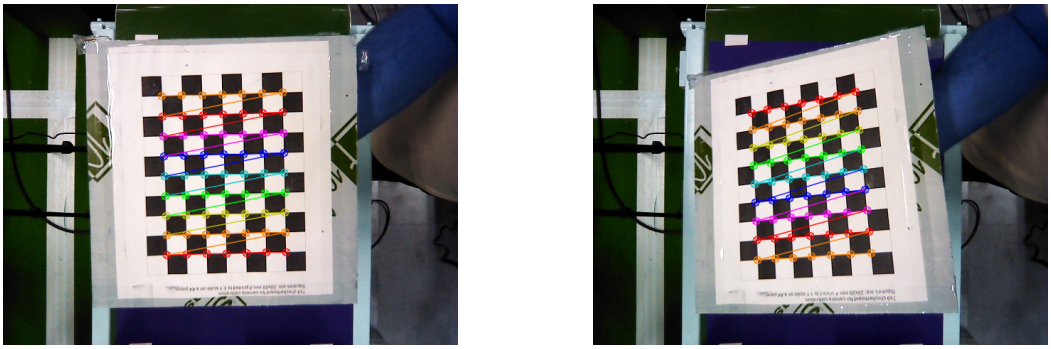
A câmara disponibiliza um conjunto diversificado de tópicos incluindo, por exemplo, as informações gerais do sensor, assim como as imagens com e sem retificação. A retificação é realizada com base nos parâmetros intrínsecos disponíveis no tópico `/camera/rgb/camera_info`. No entanto, é importante salientar que esses valores são definidos pelo fabricante para este modelo, mas não para a câmara usada, especificamente. Por essa razão, procedeu-se à extração dos parâmetros intrínsecos do sensor usado. De forma a validar a calibração realizada, procedeu-se também ao cálculo do erro de reprojeção.

Experiências

Com vista a calibrar a câmara, capturou-se um conjunto de 15 imagens com o padrão do tabuleiro de xadrez 8x10 em várias configurações de orientação e, usando a função *findChessboardCorners*, do OpenCV, identificaram-se os cantos (Figura 5.7) e as respetivas coordenadas. Como os pontos 3D, relativos aos cantos dos quadrados, estão bem definidos e igualmente

espaçados, as coordenadas de cada ponto são facilmente estabelecidas tomando um ponto como referência e definidos os restantes em relação a esse mesmo ponto. Depois de obtidas as coordenadas dos pontos 3D e 2D projetados, usou-se a função *calibrateCamera* e obteve-se os parâmetros intrínsecos da câmara. Após realizar a calibração com base neste conjunto de imagens, um arquivo de configuração **.xml** é gerado para que os parâmetros sejam utilizadas na correção das imagens.

A implementação da função *calibrateCamera*, do OpenCV, é baseada num artigo de Zhengyou Zhang [96]. Neste artigo, é proposto um método de calibração dos parâmetros intrínsecos de câmaras de uma forma fácil e flexível. O algoritmo requer apenas que a câmara observe um padrão plano amostrado em diferentes orientações de modo a identificar os pontos 3D em várias perspetivas.



(a) **(b)**

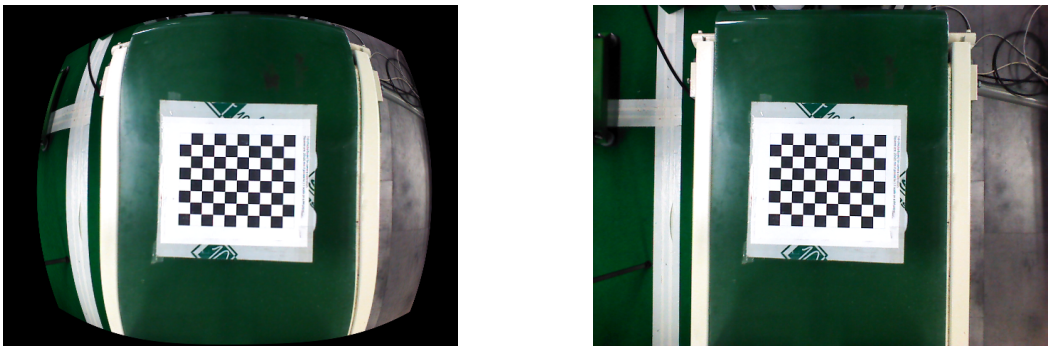
Figura 5.7 – *Frames* usados no processo de calibração.

Resultados e discussão

Com base no processo anteriormente apresentado, obteve-se a matriz transformação e de coeficientes de distorção no formato apresentado na secção 2.3:

- Matriz intrínseca: $\begin{bmatrix} 511.2 & 0 & 327.2 & 0 \\ 0 & 511.1 & 232.5 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
- Coeficientes de distorção: $\begin{bmatrix} 0.0145 & 0.0206 & -0.000483 & -0.00196 & -1.08 \end{bmatrix}$

A calibração realizada obteve um erro de reprojeção de 0.31 píxeis o que se traduz numa calibração relativamente precisa, dado que se acredita ser aceitável um erro inferior a 0.5 píxeis. Depois de aplicada a retificação obteve-se a imagem da figura 5.8b como resultado da retificação da imagem da figura 5.8a.



(a) (b)

Figura 5.8 – Ilustração do efeito da correção aplicada numa imagem sem qualquer retificação.

5.4 Pré-processamento de baixo nível

O objetivo desta secção é realizar a filtragem da imagem recebida pela câmara com o objetivo de realizar a segmentação das peças, obtendo-as de uma forma isolada. Esta secção consiste no desenvolvimento do bloco denominado "Pré-processamento" da figura 5.5.

A câmara é responsável por fazer a aquisição do cenário onde está presente e disponibilizar essa informação através de tópicos. Neste trabalho, adquiriram-se as imagens extraídas através da subscrição do tópico `/camera/rgb/image_raw`, que fornece a imagem já rectificada com uma taxa de atualização de 30 *fps*. De seguida, converteu-se a mensagem ROS, fornecida pelo tópico, em imagens formatadas para o OpenCV, usando o pacote *cv_bridge*, e fez-se o processamento da imagem.

Os algoritmos de deteção de limites caracterizam os contornos dos objetos representados e, por isso, assumem uma elevada importância no processamento de imagens. A deteção de limites, na perspetiva de reconhecimento de objetos, apresenta significativas vantagens, dado que reduz significativamente a quantidade de dados, filtrando informações desnecessárias e não comprometendo a deteção das propriedades estruturais importantes numa imagem [10]. Com base na análise realizada na secção 2.5.2, optou-se pelo algoritmo *Canny Edge*, usando a função *Canny*. Além da robustez esperada quando está presente ruído, o método é também robusto na presença de variações de luminosidade e, por isso, os limites de histerese do método devem ser cuidadosamente escolhidos.

Antes da aplicação do *Canny Edge*, com a imagem em escala de cinza, procedeu-se à filtragem do ruído com recurso à função *filter2D* usando um *kernel* de dimensões 3x3, construído com '1', para suavizar a imagem. Além disso, aplicaram-se as operações morfológicas já mencionadas na secção 2.5.3, nomeadamente a erosão e a dilatação, usando as funções *erode* e *dilate*. Ambas foram aplicadas de forma a garantir que se eliminassem pequenos segmentos isolados provenientes de, por exemplo, ruído, e que não existissem pequenas falhas nos limites dos objetos.

A deteção dos limites é também efetuada de forma a ser possível aplicar o método dos contornos, usando a função *findContours*, como se pode verificar nas figuras 5.9b e 5.9e. A segmentação das peças, viabilizada por esta função, permite individualizar as peças devido ao vetor de contornos que é criado, onde a cada índice do vetor está associado uma peça. Assim, isola-se a peça alocada no 1º índice do vetor. Esta segmentação é realizada com o objetivo

de aplicar os métodos de extração e descrição de *features* e, posteriormente, os métodos de correspondência. O tempo de processamento em todo este processo será registado a analisado.

Pretende-se ainda validar um dos requisitos definidos, nomeadamente, a distância mínima de 1 cm a que as peças se podem encontrar.

No subsistema de manipulação de objetos, desenvolvido no capítulo 6, é importante conhecer as dimensões da peça a manipular. Assim, o nó ROS responsável pelo reconhecimento das peças publica um tópico chamado `object/dimensions`, como ilustrado na figura 5.6. As dimensões são calculadas com base no menor retângulo em que a peça fica circunscrita, usando a função *minAreaRect*, do OpenCV e, deste modo, extrai-se a orientação e as dimensões do rectângulo, publicando-as no respetivo tópico.

Experiências

Definiram-se os valores limite de histerese do Canny de uma forma empírica, verificando quais os que apresentavam uma deteção dos limites mais autêntica. Deste modo, determinou-se os valores 170 e 250 como os limites mínimo e máximo, respetivamente.

Assumiu-se uma distância entre a câmara e as peças de cerca de 77 cm. Além disso, reduziu-se a dimensão da imagem para 290x390 píxeis, limitando a área de interesse aos limites às extremidades do tapete transportador. De seguida, aplicaram-se os métodos de deteção de limites e segmentação anteriormente referidos e, por fim, realizaram-se os testes para vários cenários como, por exemplo, o cenário da figura 5.9a e 5.9d.

Pretendendo-se validar um dos requisitos estabelecidos, nomeadamente, a deteção de objetos com uma distância mínima de 1 cm, realizaram-se testes com as peças distanciadas 0.5 cm e 1 cm.

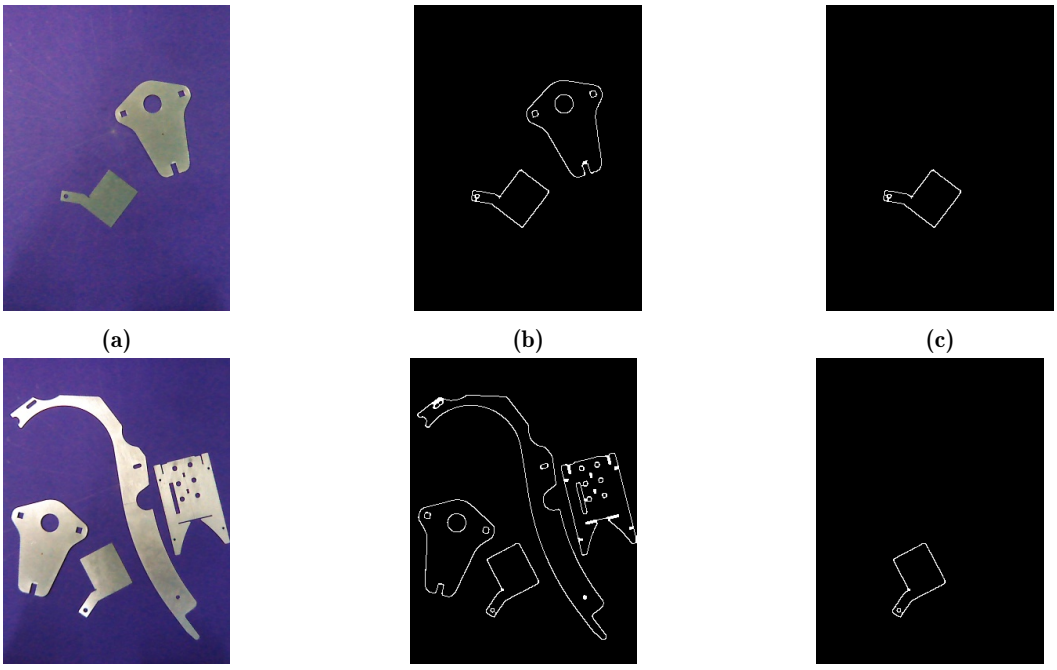
Por fim, registou-se o tempo computacional exigido em todo o processo, não só quando as peças estão isoladas como quando o cenário é constituído por várias peças. Para cada cenário realizou-se um conjunto de 100 testes.

Resultados e discussão

Como mencionado anteriormente, a figura 5.9 ilustra 2 possíveis cenários, validando o método de deteção de limites das peças (Figuras 5.9b e 5.9e) e a segmentação das mesmas (Figuras 5.9c e 5.9f) para obter a correspondência de cada peça. As figuras 5.10 e 5.11 são ilustrações dos resultados obtidos quando as peças se encontram distanciadas 0.5 cm e 1 cm, respetivamente.

Contudo, como se pode verificar na figura 5.10c a segmentação não é possível se as peças se encontrarem a menos de 1cm, pois estas são consideradas como uma só. Apesar de se encontrarem distanciadas 0.5 cm, não é suficiente tendo em conta a distância da câmara às peças e a resolução do sensor. A dilatação, neste caso não contribui para a segmentação das peças levando a que contornos quase coincidentes se agrupem. No cenário da figura 5.11a as peças encontram-se distanciadas exatamente 1 cm e, neste caso, já é possível validar a segmentação das peças como ilustrado na figura 5.11c, cumprindo o requisito imposto relativamente à distância mínima entre peças.

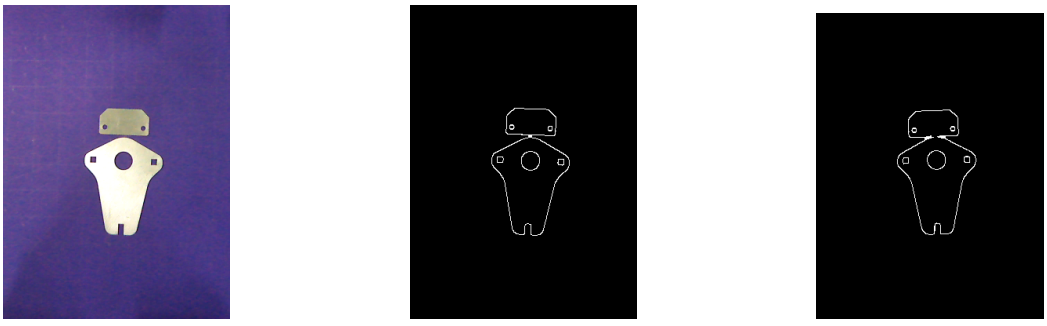
A tabela 5.1 ilustra a média, num conjunto de 100 testes, dos tempos obtidos nesta etapa do subsistema, onde são realizados todos métodos referidos nesta secção. Os tempos obtidos em todos os cenários são bastante reduzidos, na ordem dos 2 a 3 milissegundos, dando boas indicações da sua usabilidade numa aplicação robótica de elevada cadência.



(a) (b) (c)

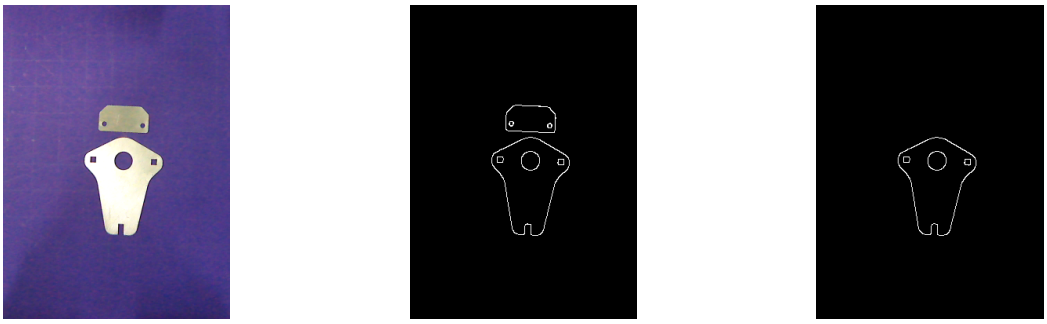
(d) (e) (f)

Figura 5.9 – Ilustração da segmentação das peças usando o *Canny Edge*.



(a) (b) (c)

Figura 5.10 – Ilustração do processo de segmentação com as peças distanciadas 0.5 cm.



(a) (b) (c)

Figura 5.11 – Ilustração do processo de segmentação com as peças distanciadas 1 cm.

Nº da peça	**Tempo (ms)**	
	Média	**Desvio-padrão**
1	2.68	0.85
2	2.69	0.59
3	2.68	0.73
4	2.68	0.72
5	2.72	0.67
6	2.62	0.62
7	2.71	0.66
Várias peças	2.75	0.99

Tabela 5.1 – Tempos computacionais obtidos no pré-processamento das imagens.

5.5 Estimativa da posição

No que diz respeito ao reconhecimento de objetos, interessa não só identificar qual o objeto que se encontra sobre o tapete transportador, mas também, a sua disposição sobre o mesmo, de forma a possibilitar uma correta manipulação.

O objetivo nesta secção é determinar a posição exata da peça independentemente da sua configuração no tapete. A posição da peça é dada por uma das propriedades dos contornos designada de centróide que, na prática, corresponde ao centro geométrico dos contornos da própria peça. Assim, caso se saiba o posicionamento exato da câmara, é possível determinar com rigor a posição do objeto relativamente à câmara e, consequentemente, relativamente ao manipulador. Depois de estimada a posição da peça, relativamente à câmara, essa informação é publicada no tópico `/object/position`.

Esta localização apenas é possível usando o modelo *Pinhole* da câmara apresentado na secção 2.3. Com base na trigonometria do modelo descrito na figura 2.1 e sabendo o distância focal (f), a distância da câmara ao objeto (Z) e a distância do objeto, relativamente ao centro da câmara, projetado em píxeis (x), pode-se determinar a distância do objeto ao centro da câmara em metros (X).

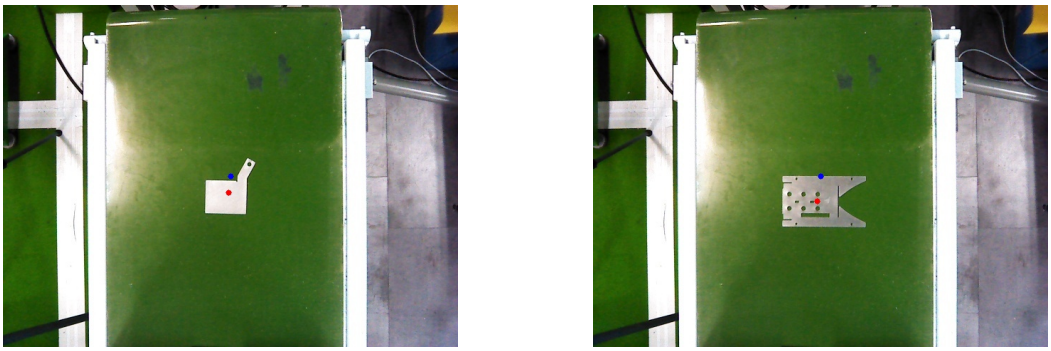
Experiências

Acedendo aos parâmetros intrínsecos disponíveis no tópico `/camera/rgb/camera_info`, recorreu-se à classe *PinholeCameraModel*, mais concretamente às funções *getDeltaX* e *getDeltaY*. Desta forma, é possível obter as coordenadas do objeto relativamente à tf da câmara (*camera_link*). A estimativa da tf da câmara, proveniente da calibração extrínseca, irá ser explicada e realizada posteriormente no capítulo 6.

Assumiu-se uma distância entre a câmara e as peças de cerca de 77 cm. Assim, com o objetivo de validar a exatidão da estimativa da posição, baseada no centróide da mesma, estas foram verificadas, como ilustrado na figura 5.12, para um conjunto de posições com distanciamento nulo e distanciadas do centro da câmara em 5 cm, 10 cm, 15 cm e 20 cm. Assim, para cada amostra, calculou-se o erro da estimativa da posição.

Resultados e discussão

A figura 5.12 ilustra os testes de estimativa de posição realizados para as peças Nº3 e Nº4.



(a) **(b)**

Figura 5.12 – Estimativa da posição baseado no centróide da peça. O ponto a azul representa o centro da câmara e o ponto a vermelho o centróide.

Identificou-se facilmente o centróide em todo o conjunto de peças disponíveis, como ilustrado, por exemplo, nas figuras 5.12a e 5.12b, para as peças Nº3 e Nº4, respetivamente. Relativamente o erro de estimativa da posição, obtiveram-se os erros absolutos apresentados na tabela 5.2. De notar que existe uma certa tendência crescente do erro em função da distância, proveniente da não exata calibração intrínseca. Os valores obtidos são próximos dos valores desejados nos requisitos estabelecidos, no entanto, existe ainda margem para melhorias. Estes valores foram obtidos nas condições anteriormente mencionadas, no entanto acredita-se que existem dois aspetos que poderiam contribuir positivamente para a obtenção de resultados mais satisfatórios, por um lado, o uso de uma câmara com uma maior resolução e, por outro lado, a redução da distância entre as peças e a câmara.

5.6 Estimativa da orientação baseada na correlação

Seguindo o mesmo raciocínio da necessidade de saber a posição da peça, é igualmente importante saber a orientação da mesma. Assim, desenvolveu-se um algoritmo que se baseia

Distância (cm)	Erro (mm)
0	1.01
5	2.10
10	2.89
15	2.57
20	3.32

Tabela 5.2 – Erro da calibração em função da distância.

na correlação entre duas imagens, ao nível do píxel, usando as funções *filter2D* e *matchTemplate*, disponibilizadas pelo OpenCV. Este método poderia também ser usado para detetar correspondências de objetos, no entanto, é de extrema dificuldade determinar um valor limite de correlação, que permita diferenciar uma correspondência de uma não correspondência.

Assumindo, à partida, que se trata de uma correspondência do objeto, na perspetiva de calcular a transformação entre a imagem original e a imagem modelo, este método apresenta algumas limitações como a sua falta de invariância à rotação, escala e reflexão. A reflexão é também relevante, dado que, dependendo da geometria da peça, esta pode apresentar diferentes formatos quando refletida, o que ao nível do píxel, pode resultar em grandes variações. Como forma de mitigar essas limitações, desenvolveu-se uma algoritmo que, iterativamente, faz o reajuste das imagens ao nível da sua escala, rotação e reflexão e, a cada iteração, é avaliada a correlação entre a imagem original e a imagem modelo.

De seguida, são apresentadas duas metodologias baseadas no mesmo raciocínio onde, em cada uma delas, se testou ainda as duas funções de cálculo de correlação. Ambas as abordagens são avaliadas em dois parâmetros: o desempenho temporal e o erro, em graus, da estimativa da orientação.

5.6.1 Estimativa exclusivamente baseada na correlação

Após o processamento da imagem apresentado na secção 5.4 aplicou-se um método iterativo onde em cada iteração é calculada a correlação entre a imagem orignal e a imagem modelo. Assim, fez-se uma rotação de 10° por cada iteração até completar uma rotação completa. Posteriormente, fez-se uma verificação mais pormenorizada nas zonas onde a variação relativa da correlação se apresentou mais elevada. Ou seja, numa gama de $\pm 10°$ e com variações de ângulos de apenas 1°, realizou-se uma análise mais rigorosa do ângulo de rotação, o que permitiu concluir qual a posição estimada do objeto relativamente ao modelo. Sabendo que a escala é sempre a mesma, com a exceção de pequenas variações que possam existir na aquisição do frame, uma vez que a câmara está fixa e posicionada perpendicularmente sobre tapete transportador. No entanto, de modo a tornar o algoritmo mais robusto, é considerada uma variação de escala de $\pm 2\%$, com um passo de 1%. Além disso, o processo iterativo de reajuste, tanto ao nível da escala como da rotação, é também realizado para a imagem reflectida de forma a detetar peças invertidas. A deteção de uma peça invertida baseia-se em verificar se existe uma variação relativa superior com a peça invertida.

Experiências

Realizaram-se testes para todas as peças, com variações de rotação entre 0° e 360°, com um incremento de 10° e mudanças de reflexão. Além disso, como referido anteriormente,

testou-se para todo o conjunto de peças quais as diferenças de desempenho ao nível do tempo computacional e do erro da estimativa da rotação, usando as funções *filter2D* e *matchTemplate*.

Resultados e discussão

A título de exemplo, apenas se apresentam os gráficos, usando a função *matchTemplate* sem a imagem invertida, obtidos para o caso onde a peça Nº4 apresenta uma posição exata de 60° relativamente ao modelo. As figuras 5.13a e 5.13b ilustram a correlação obtida para 1ª estimativa do ângulo e as respetivas variações relativas relativamente ao valor médio das correlações referentes aos restantes ângulos. De seguida, é ilustrado nas figuras 5.13c e 5.13d, o ajuste fino realizado, onde é possível verificar as correlações calculadas assim como as respetivas variações relativas.

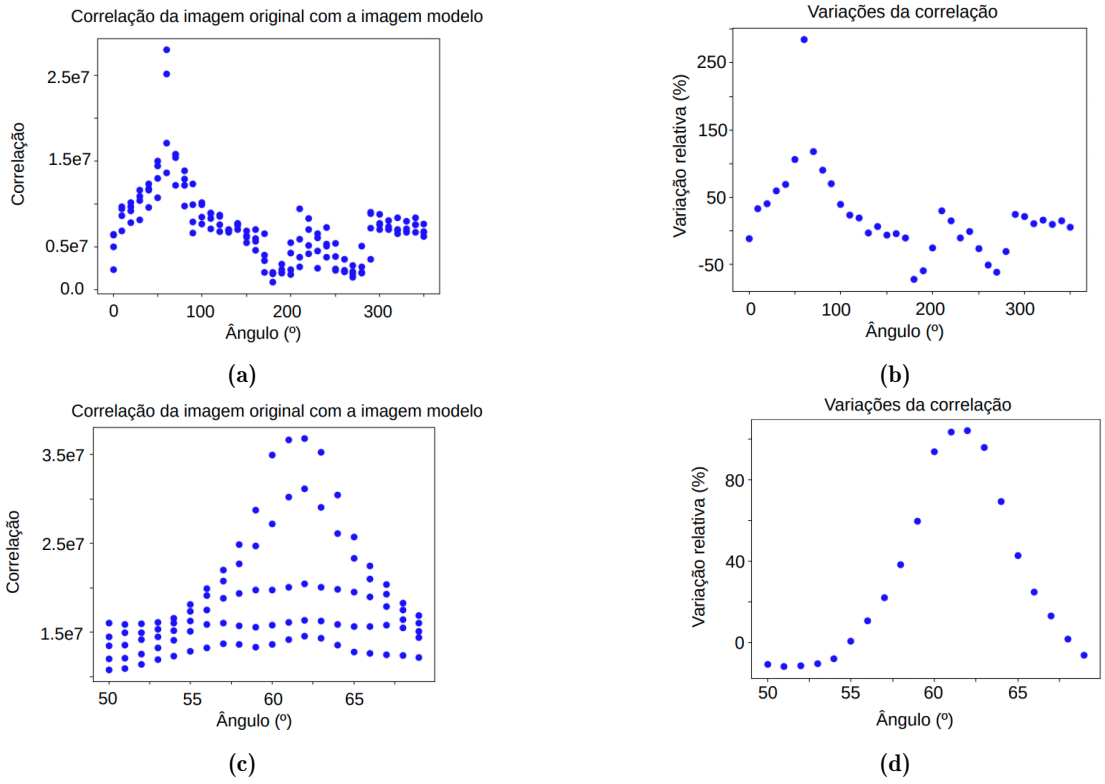


Figura 5.13 – Peça Nº4 numa posição real de 60° relativamente ao modelo.

A tabela 5.3 faz a comparação, tanto do tempo computacional requerido, como do erro do ângulo obtido pelo algoritmo desenvolvido, onde se usaram as duas funções do OpenCV.

Conforme apresentado na figura 5.13a existe um aumento significativo da correlação numa gama de valores a rondar os 60° e, na figura 5.13b, pode verificar-se que o maior aumento, corresponde a essa gama, com uma variação de cerca de 280% relativamente ao valor médio de todas as correlações calculadas. Em conformidade com a figura 5.13d, conclui-se que o ângulo estimado é 62° com um aumento de correlação de cerca de 105%. Sabendo que o ângulo real é de 60° esta solução, neste teste, apresenta uma erro absoluto de 2°.

Usando a função *filter2D*, o erro médio absoluto, no conjunto das 7 peças, é de cerca de

Algoritmo	Tempo (s)		Erro ângulo (°)	
	Média	Desvio-padrão	Média	Desvio-padrão
filter2D	14.34	0.96	0.78	0.72
matchTemplate	6.22	0.75	2.53	3.07

Tabela 5.3 – Comparação dos algoritmos *filter2D* e *matchTemplate*.

0.78±0.72°. Esta solução apresenta uma exatidão insatisfatória dado que em alguns casos pode ultrapassar o erro de 1°, em termos de eficiência temporal, deixa muito a desejar, sendo que, neste conjunto de testes, o tempo médio é de 14.34±0.96 segundos. Logo, tendo em conta os objetivos propostos é impraticável. Usando a função *matchTemplate* no mesmo conjunto de testes, o erro médio absoluto obtido é de 2.53±3.07° e o tempo médio obtido é inferior, com cerca de 6.22±0.75 segundos. Esta solução apresenta uma exatidão ainda menor, no entanto em termos de eficiência temporal apresentar melhores resultados, contudo estes são ainda elevados olhando os requisitos definidos para este sistema.

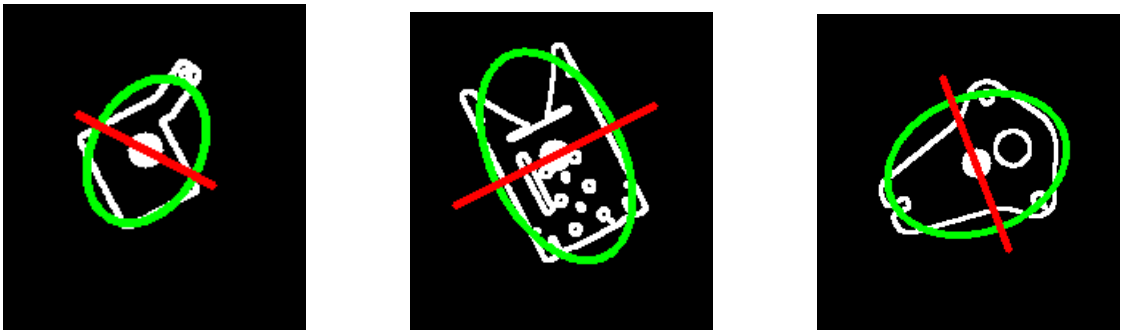
5.6.2 Estimativa baseada no fitEllipse e na correlação

Como é perceptível, a metodologia anteriormente apresenta uma exatidão e uns tempos computacionais desajustados aos requisitos definidos e, por isso, foi testada uma outra abordagem onde se usou um algoritmo adicional que está encarregue da primeira indicação do ângulo em que se encontra a peça. Ou seja, em alternativa à primeira etapa do processo iterativo em que se realizava uma rotação completa e se verificava o ângulo correspondente à correlação mais elevada, nesta abordagem esse processo é realizado pelo método *fitEllipse*. Esta função calcula a elipse que melhor se ajusta a um conjunto de pontos bidimensionais e a orientação resultante da elipse retorna a indicação do ângulo pretendida.

As experiências realizadas são as mesmas apresentadas na secção 5.6.1.

Resultados e discussão

A figura 5.14a, 5.14b e 5.14c ilustram o resultado da aplicação do método *fitEllipse* na peça Nº3, Nº4 e Nº5, respetivamente.



(a) (b) (c)

Figura 5.14 – Aplicação do método *fitEllipse*.

A tabela 5.4 faz a comparação, tanto do tempo computacional requerido, como do erro do ângulo obtido pelo algoritmo desenvolvido, onde se usou as duas funções do OpenCV, depois de se aplicar o *fitEllipse*.

Algoritmo	**Tempo (s)**		**Erro ângulo (°)**	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
filter2D	3.09	0.75	2.27	3.45
matchTemplate	0.79	0.13	3.39	1.04

Tabela 5.4 – Comparação dos algoritmos *filter2D* e *matchTemplate*.

Usando o *filter2D*, obteve-se um erro médio absoluto de cerca de 2.27±3.45° e um tempo médio de 3.09±0.75 segundos. Usando o *matchTemplate* obteve-se erro médio absoluto de 3.39±1.04° e um tempo médio de 0.79±0.13 segundos. Ambos os métodos apresentam um erro superior aos resultados apresentados na secção 5.6.1. Além disso, o *matchTemplate* apresenta, novamente, os tempos menores, mas ainda desajustados à situação problema, pois é preciso ter em conta que este método apenas é usado para calcular a sua orientação, logo a este tempo é ainda acrescentado um tempo computacional para as restantes verificações como, por exemplo, a posição e a própria correspondência. Assim, é necessário estudar outra abordagem que apresente um melhor desempenho ao nível da exatidão e do tempo computacional associado.

Com base nos testes realizados em ambas as abordagens, é também possível concluir que o *matchTemplate* apresenta sempre tempos de computação inferiores ao *filter2D*. As diferenças podem dever-se à implementação interna da própria função, que não foi alvo de estudo.

5.7 Correspondência de peças usando features

Uma vez que a correspondência de peças baseada apenas na correlação apresenta exatidões e tempos computacionais desajustados, tendo em conta os requisitos definidos, desenvolveu-se uma metodologia para a correspondência das peças baseada em *features*.

Os objetivos são reconhecer a peça, assim como determinar a sua configuração, nomeadamente, a rotação e a reflexão. Uma vez que o foco desta dissertação não é desenvolver um novo algoritmo de extração e descrição de *features*, estudaram-se alguns dos algoritmos já existentes e verificou-se qual o que apresenta resultados mais satisfatórios com base num conjunto de critérios, seguidamente apresentados. Relativamente ao método de correspondência de descritores, testou-se tanto o método *Brute Force* como os algoritmos disponíveis na biblioteca FLANN, com vista a analisar qual o que apresenta melhores resultados segundos os critérios a serem estabelecidos.

Como referido na secção 2.5.2, o formato de cada objeto é, nesta aplicação em específico, o fator mais relevante na caracterização do objeto e, por isso, qualquer tipo de atributos adicionais são dispensáveis. Logo, simplificou-se e reduziu-se a complexidade das imagens, através do pré-processamento já mencionado, antes da aplicação de qualquer método de extração e descrição de *features*.

5.7.1 Extração e descrição de features

Com base na síntese desenvolvida na secção 2.7.9, optou-se por testar apenas alguns dos extratores e descritores já existentes. Testaram-se os extratores e descritores binários ORB e BRISK, e os baseados em valores reais, nomeadamente, o SIFT e o SURF. Neste estudo, pretende-se aferir quais os métodos que apresentam um melhor compromisso entre a deteção de "boas" correspondências, ou seja, as correspondências que realmente são usadas nos métodos de correspondências das *features*, e o tempo necessário em todo processo desde a extração, descrição e cálculo das correspondências. Com o objetivo de aferir qual o método que melhor se adequa aos objetivos desta dissertação, decidiu-se testar o comportamento dos extratores e descritores tendo em conta as variações ao nível da rotação, escala e na presença de ruído gaussiano. Os parâmetros recolhidos para cada um dos algoritmos são:

1. Número de *features* extraídas;
2. Tempo de extração e descrição das *features*;
3. Número total de correspondências;
4. Número de correspondências mais distintivas;
5. Número de *inliers*, segundo o RANSAC;
6. Tempo total.

Os dois primeiros critérios ilustram, por um lado, o número de pontos-chave extraídos, assim como, o tempo de processamento necessário, não só para os identificar como a descrevê-los. O número total de correspondências resulta do método de correspondência usado sem filtragem, que depois são sujeitas a duas etapas de rejeição de correspondências, ou por serem pouco distintivas ou por serem consideradas *outliers*, segundo o método RANSAC. A primeira etapa é baseada no critério apresentado na secção 2.8, onde o algoritmo desenvolvido rejeita todas as correspondências em que o rácio da distância entre as correspondências encontradas é superior a 0.8, de forma a selecionar as correspondências mais distintivas. A 2ª etapa de rejeição baseia-se no número de *inliers* que são devolvidos pelo algoritmo RANSAC. O tempo total é o tempo de processamento necessário para a execução de todos os procedimentos anteriormente referidos. Com base nestes parâmetros extraídos, os critérios de avaliação escolhidos para os diferentes descritores são os seguintes:

1. Eficiência temporal na extração e descrição de *features*, ou seja, o rácio entre o número de *features* extraídas e descritas, e o tempo necessário nesse processo;
2. Rácio entre o número de *inliers* e o número total de correspondências;
3. Eficiência temporal no cálculo das correspondências: rácio entre número de *inliers* e o tempo computacional necessário total;
4. Tempo total.

Usaram-se diagramas de caixa para fazer a ilustração dos resultados obtidos, segundo os critérios apresentados. Os diagramas de caixa dividem-se essencialmente em 4 conjuntos de dados, em que cada um representa 25% dos dados.

Por fim, pretende-se determinar quais os limites mínimos, tanto das correspondências mais distintivas como de *inliers*, para cada um dos algoritmos. Estes limites são depois usados como critérios de seleção para quando uma peça estiver a ser comparada com as peças da base de dados logo, se num processo de correspondência os valores obtidos forem inferiores a estes limites, pode-se descartar à partida essa peça.

Experiências

Com o objetivo de testar os diferentes extratores e descritores, criou-se uma base de dados local, onde estão presentes múltiplas imagens de cada peça. Quanto às variações de rotação, estas encontram-se rodadas 5°, de 0° a 360°, formando assim um conjunto de 72 imagens diferentes para cada peça. Relativamente à escala, aplicou-se a uma das imagens de cada peça, variações de $\pm 10\%$, com um passo de 1° e por último, aplicou-se ruído branco gaussiano. Os testes realizaram-se sempre com o mesmo método de correspondência de descritores, o *Brute Force*, de forma a testar os descritores sempre nas mesmas condições. Dada a natureza dos descritores, as correspondências dos métodos SIFT e SURF são calculadas com base na distância euclidiana, enquanto que os métodos ORB e BRISK usam a distância de *hamming*.

Os extratores e descritores mencionados no capítulo 2 são explorados pelo OpenCV e, por essa razão, existem diversas funções parametrizáveis desses algoritmos. Assim definiram-se alguns dos parâmetros como, por exemplo, o *scoreType* do ORB, como *orb_fast_score*, pois é o método classificador de *features* mais rápido. Ainda, no SURF, definiu-se, o parâmetro *extended* como *True*, de forma a usar a versão do algoritmo com 128 *bits*, o que significa que a descrição dos recursos é ligeiramente mais lenta, mas com melhor desempenho. Além disso, definiu-se para todos os algoritmos, o máximo de 1000 *features* extraídas. Os restantes parâmetros não são mencionados, pois não se realizaram alterações relativamente aos valores padrão sugeridos na documentação do OpenCV. Além disso, usou-se a função *estimateAffinePartial2D* numa configuração baseada no método RANSAC, com um máximo de 2000 iterações.

Resultados e discussão

As figuras 5.15, 5.16 e 5.17 são, exemplos de resultados dos testes de correspondência de peças realizados, com variações ao nível da rotação, escala e ruído, respetivamente. Os pontos multicolor são os pontos-chave extraídos e os segmentos a verde correspondem às correspondências consideradas *inliers*.

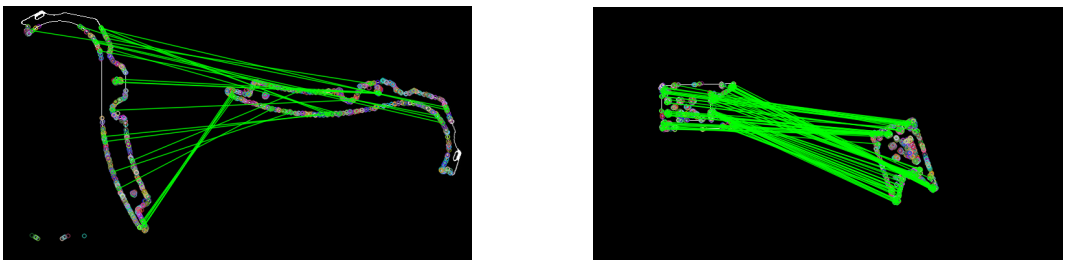


Figura 5.15 – Correspondência de objetos com variações de rotação usando o ORB.

Os diagramas de caixa das figuras 5.18, 5.19, 5.20 e 5.21 ilustram os resultados obtidos, para cada um dos algoritmos, tendo em consideração cada um dos critérios de avaliação.

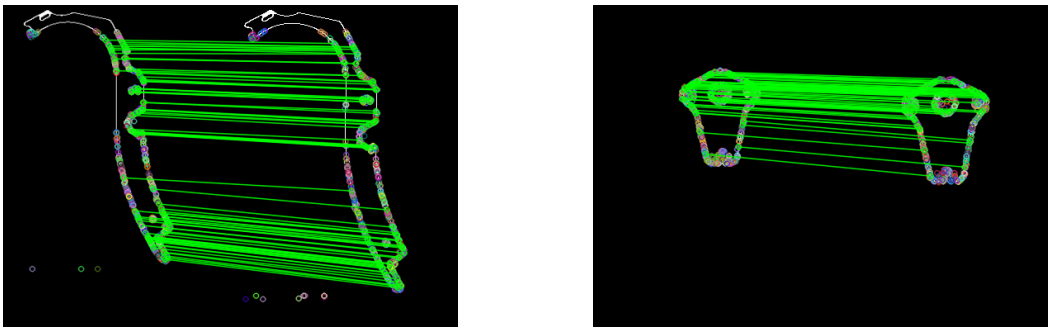


Figura 5.16 – Correspondência de objetos com variações de escala usando o ORB.

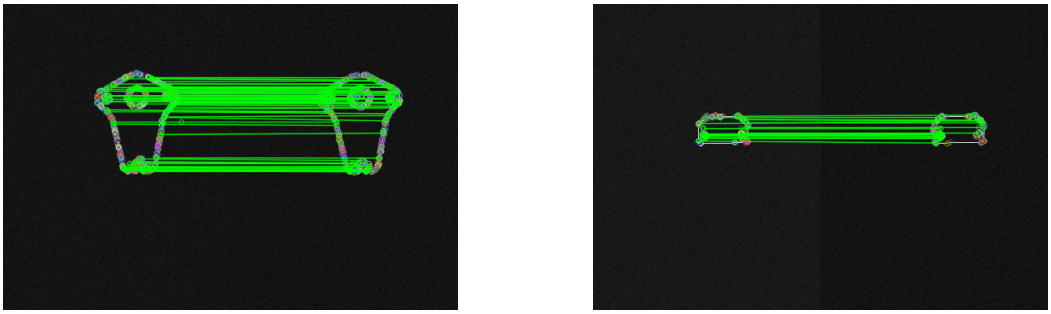
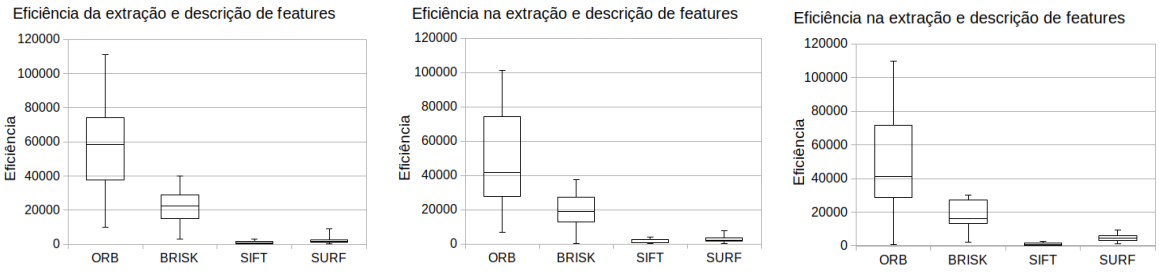
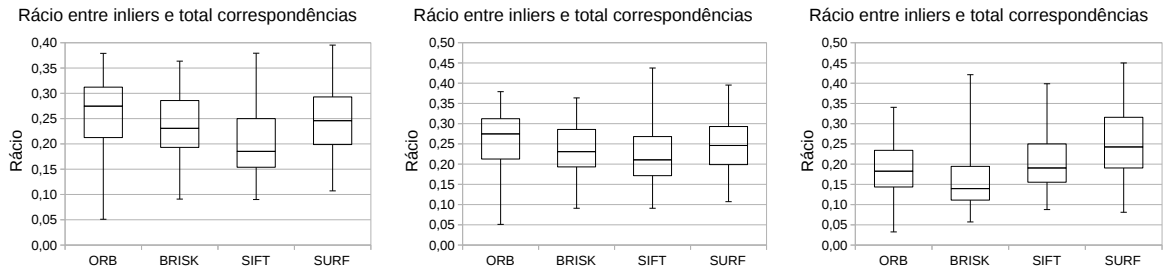


Figura 5.17 – Correspondência de objetos com variações de ruído gaussiano usando o ORB.



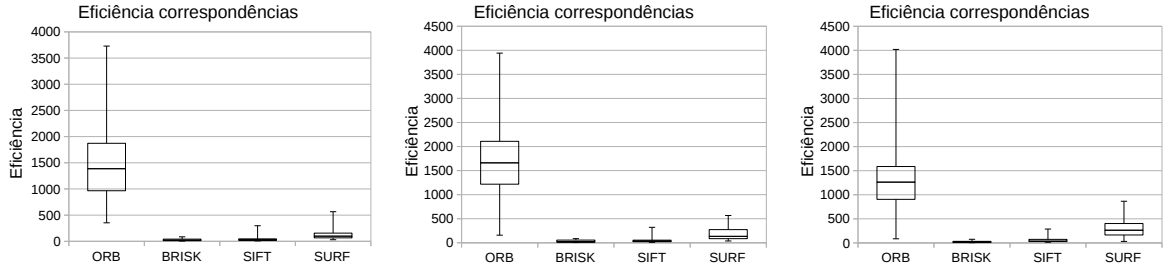
(a) Variações de rotação. **(b)** Variações de escala. **(c)** Variações de ruído.

Figura 5.18 – Comparação dos algoritmos, com base na eficiência da extração e descrição das *features*.



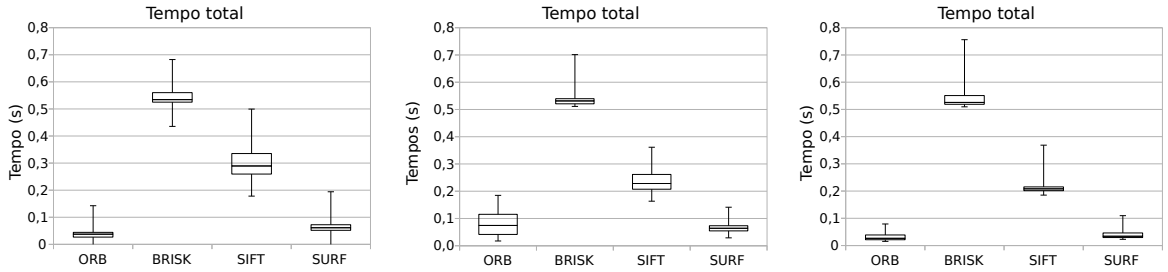
(a) Variações de rotação. **(b)** Variações de escala. **(c)** Variações de ruído.

Figura 5.19 – Comparação dos algoritmos, com base no rácio entre o número *inliers* e o número total de correspondências.



(a) Variações de rotação. **(b)** Variações de escala. **(c)** Variações de ruído.

Figura 5.20 – Comparação dos algoritmos, com base na eficiência de correspondências.



(a) Variações de rotação. **(b)** Variações de escala. **(c)** Variações de ruído.

Figura 5.21 – Comparação dos algoritmos, com base no tempo total de processamento.

Com base nos resultados obtidos consegue-se aferir as seguintes conclusões:

- Nos 3 testes da figura 5.18, o ORB é o mais eficiente na extração e descrição de *features*, logo têm uma melhor relação entre o número de *features* extraídas e descritas, e o tempo computacional exigido. A seguir ao ORB estão os algoritmos BRISK, SURF e o SIFT;
- Quanto ao rácio entre o número de *inliers* e o número total de correspondências encontradas, o ORB e o SURF têm os 75% dos dados com os maiores rácios, nos testes das figuras 5.19a e 5.19b, seguido do BRISK e do SIFT. Na figura 5.19c, o SURF têm o maior rácio seguido do SIFT que apresenta uma ligeira superioridade em relação ao ORB, com 75% dos seus rácios a serem superiores, seguido do BRISK;

- O ORB destaca-se no que à eficiência de correspondências diz respeito. Como se pode verificar nos gráficos da figura 5.20 a eficiência do ORB é significativamente superior, com 75% dos resultados a serem sempre superiores à totalidade da eficiência dos restantes algoritmos. A seguir ao ORB, tem-se o SURF com uma gama de eficiência relativamente maior que os restantes algoritmos.

- Relativamente ao tempo total, o ORB e o SURF são os mais rápidos em todos os testes, como se pode verificar nos gráficos da figura 5.21, enquanto que o SIFT é o 3º mais rápido e, por último, o BRISK.

Com base nos resultados obtidos, optou-se por testar o ORB e o SURF no âmbito das correspondências das *features*. Estes são os dois algoritmos que apresentam um melhor compromisso entre a deteção de "boas" correspondências, ou seja, as correspondências que realmente são usadas nos métodos de correspondências das *features*, e o tempo necessário em todo processo de a extração, descrição e cálculo das correspondências. Tendo em conta apenas os algoritmos selecionados, o limite mínimo de correspondências mais distintivas do ORB é 10 e 4 o número de *inliers* mínimos. Quanto ao SURF, o limite mínimo de correspondências mais distintivas e de *inliers* é igual a 3.

5.7.2 Correspondência de features

Nos testes anteriores optou-se por fixar o método de correspondência e, por isso, usou-se sempre o método *Brute Force*. Nesta fase, o objetivo é determinar qual o método de correspondência de *features* que apresenta uma maior taxa de sucesso na correspondência de peças, assim como uma maior eficiência temporal.

O OpenCV disponibiliza um conjunto de métodos de correspondência, normalmente usados no reconhecimento de objetos como, por exemplo, o *match* e o *knnMatch*. Cada um dos métodos recorre à lista de descritores da imagem captada pela câmara e compara-os aos descritores armazenadas na base de dados, também chamado de dicionário. No método *match*, cada descritor da lista de consulta será correspondido com a melhor correspondência das listas de descritores armazenados, onde a pesquisa é realizada por todos os descritores. No método *knnMatch*, para cada descritor da lista de consulta, é feita a procura por um número específico das melhores correspondências, no respetivo dicionário. Esse número é dado pelo argumento k da função.

Tendo em conta os valores limite para o número de correspondências mais distintivas e o número de *inliers*, definidos na secção 5.7.1, desenvolveu-se um algoritmo de reconhecimento de peças como representado no algoritmo 2. As imagens modelo, assim como os respetivos descritores, estão armazenados na base de dados, e a imagem do objeto que se encontra sobre o tapete é adquirida e submetida aos métodos de processamento anteriormente tratados, como ilustrado na figura 5.22. De seguida, é realizada a comparação dos descritores, usando um determinado método de correspondência e depois calculadas, como mencionado anteriormente, as correspondências mais distintivas, assim como os respetivos *inliers*, segundo o RANSAC.

Na figura 5.22, por uma questão de simplificação não está ilustrado processo realizado com as imagens processadas. Esse processo apesar de apenas ser tratado na próxima secção já se reflete nos resultados obtidos ainda nesta secção. No entanto, nesta fase o foco é apenas a comparação do método de correspondências.

O número de correspondências mais distintivas e *inliers* é um forte indício de uma correspondência de uma peça e, caso estes limites não se estabeleçam, isto traduz-se numa não

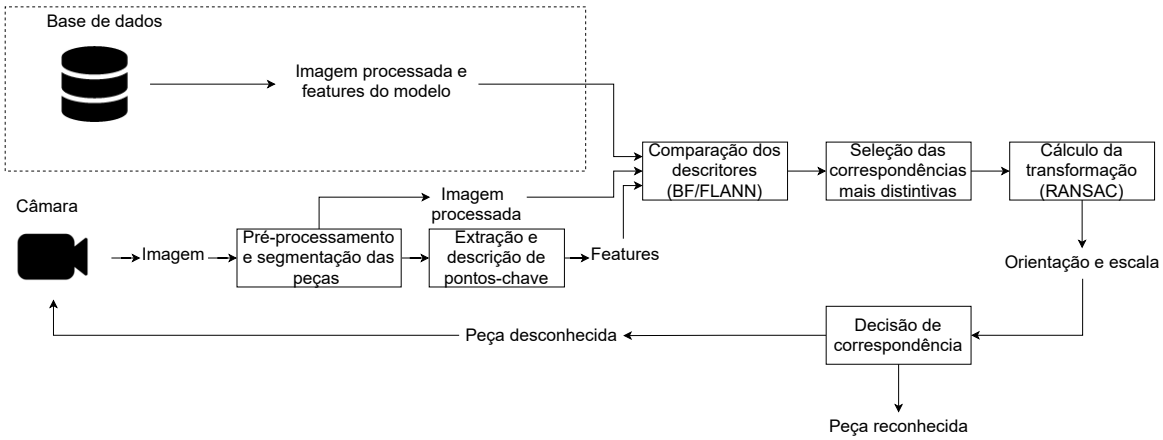


Figura 5.22 – Descrição esquemática do sistema de reconhecimento de objetos.

correspondência. No entanto, no reconhecimento de um determinado objeto, não será robusto o suficiente, dado que outras peças podem facilmente obter um número de correspondências dentro dos limites. Além disso, só através deste método não é possível determinar a orientação do objeto observado.

Assim usou-se uma função disponível no OpenCV designada *estimateAffinePartial2D* que calcula a matriz transformação usando o método RANSAC. A matriz dá a transformação entre os descritores da imagem modelo e os descritores da imagem capturada, onde é possível extrair informações sobre a rotação, escala e translação. A escala extraída é também usada como um critério de decisão, dado que, caso a escala não seja aproximadamente 1, logo traduz-se numa não correspondência da peça, pois as peças estão sempre à mesma distância da câmara. Uma vez estabelecido o critério da escala, é interrompida a pesquisa por correspondências, na restante base de dados, e extrai-se o ângulo resultante que corresponde à rotação que é necessária a imagem modelo realizar, de modo a ficar na mesma configuração da peça observada. Na secção 5.7.3 é tratado o reconhecimento de peças refletidas e os problemas inerentes enfrentados. A função, denominada "Reflexão", no algoritmo 2, trata essa problemática.

Algoritmo 2 Reconhecimento de peças

Input: *img_modelo* ▷ Imagem modelo da base de dados.
modelo_com_reflexão ▷ Variável indicativa da simetria.
img ▷ Imagem capturada pela câmara.
kp2, desc2 ▷ *Features* da peça modelo da base de dados.
Output: *msg* ▷ Indicação se a peça é reconhecida.

$k = 2, rácio = 0.8, var = 0.05$
min_correspondencias_dist ▷ Nº mínimo de correspondências mais distintivas.
min_inliers ▷ Nº mínimo de *inliers*.
$msg =$ “”
$desc = descritor$
$img\_modelo, modelo\_com\_reflexão \leftarrow$ Carregar_Imagem_Modelo_processada()
$m\_c\_r = modelo\_com\_reflexão$ ▷ Booleano indicativo de reflexão.
$img \leftarrow$ Carregar_imagem_original_processada()
$kp2, desc2 \leftarrow$ Carregar_Features(img_modelo)
for *p=0 to 1* **do** ▷ Ciclo da imagem original (p=0), imagem original invertida (p=1).
    if $p == 0$ **then**
        $kp1, desc1 \leftarrow$ Detetar_Extrair_Features(img)
    else if $p == 1$ **then**
        $img\_refletida \leftarrow$ Refletir_Imagem(img)
        $kp1, desc1 \leftarrow$ Detetar_Extrair_Features($img\_refletida$)
    end if
    correspondências $\leftarrow$ knnMatch(*desc1, desc2, k*)
    correspondencias_dist $\leftarrow$ Correspondências_Distintivas(*correspondências, rácio*)
    if dim(*correspondências_dist)* $\geq$ *min_correspondências_dist* **then**
        for i=0 to dim(*correspondências_dist*) **do**
            u $\leftarrow$Obter_Índice_Img_Modelo(*correspondências_dist(i)*)
            v $\leftarrow$Obter_Índice_Img(*correspondências_dist(i)*)
            q $\leftarrow$Adicionar_Features(*kp1(u)*)
            t $\leftarrow$ Adicionar_Features(*kp2(v)*)
        end for
    else
        msg $\leftarrow$ “Poucas correspondências distintivas encontradas, peça não reconhecida”
    end if
    for *tentativa = 1 to 2* **do** ▷ São dadas duas tentativas ao RANSAC.
        $H, mascara\_inliers \leftarrow$ EstimateAffinePartial2D(*q, t, RANSAC*)
        $inliers =$ len($mascara\_inliers$)
        $escala, ângulo \leftarrow$ Extrair_Transformações(M)
        if $escala > 1 - var \ \& \ escala < 1 + var$ **then**
            Orientação($p, tentativa, inliers, ângulo, img, img\_modelo, img\_refletidam\_c\_r$)
            msg $\leftarrow$ “Peça reconhecida.”
        else
            if $m\_c\_r ==$ False **then**
                msg $\leftarrow$ “Escala inválida, peça não reconhecida.”
            end if
        end if
    end for
end for

Com o objetivo de determinar qual o método que apresenta, não só, uma maior taxa de sucesso, como também, uma maior eficiência temporal na correspondência das peças, analisaram-se as correspondências de objetos com base nos seguintes critérios:

1. Tempo usado pelo algoritmo, no ciclo onde se encontra uma correspondência de uma peça;

2. Número de testes em que o algoritmo deteta como uma correspondência, no entanto, trata-se de uma correspondência incorreta da peça, logo um falso positivo;

3. Número de testes em que o algoritmo não deteta como uma correspondência, no entanto, trata-se de uma correspondência da peça, logo um falso negativo;

Assim, o processo de seleção do método de correspondência fez-se em 2 etapas, onde primeiro se deu prioridade ao descritor que encontra mais vezes a correta correspondência e depois verificou-se qual o método de correspondência que apresenta melhores resultados segundo os critérios apresentados.

Experiências

Com base nos resultados dos testes da secção 5.7.1 selecionou-se o ORB e o SURF, usando os mesmos parâmetros dos testes da secção anterior, para os testes de correspondências de objetos. Para ambos os métodos fez-se a pesquisa de *Brute Force*, usando o método *Match*, e a pesquisa dos k vizinhos mais próximos (*k-nearest neighbors*), usando a função *knnMatch* com um valor de k=2, o que significa que para cada imagem pesquisada são encontrados os dois vizinhos mais próximos para cada descritor na imagem modelo.

Os testes baseiam-se em selecionar uma imagem de cada uma das peças e extrair as respetivas *features*. De seguida, verifica-se as correspondências com as *features* das peças existentes na base de dados local. Esta base de dados é constituída, como já referido, por um conjunto de 72 variações da mesma peça, ou seja, de forma simula existem 504 peças. A questão das peças refletidas é detalhadamente tratada na próxima secção e nesta fase assume-se que todas as peças não contém diferenças ao nível da simetria.

Quanto ao ORB, usou-se o algoritmo LSH, disponível na biblioteca FLANN, devido à sua capacidade de lidar com descritores binários e ativou-se o parâmetro *multi-probe* que, segundo a documentação do OpenCV, apesar de resultar em mais correspondências para avaliar, na realidade, faz com que o número de tabelas *hash* diminua, o que resulta num menor peso computacional, como referido na secção 2.8.2. Relativamente ao descritor SURF usou-se o algoritmo *randomized kd-tree*, pois está preparado para lidar com descritores constituídos por valores reais, de grandes dimensões.

Para estes testes, usou-se a função *estimateAffinePartial2D* numa configuração baseada no método RANSAC, com um máximo de 2000 iterações, e devido à sua componente aleatória na seleção das *features* optou-se por conceber duas tentativas ao método com o objetivo de determinar uma transformação de escala aproximadamente unitária. Além disso, definiu-se o número mínimo das melhores correspondências e *inliers*, associados ao ORB e SURF, definidos na secção 5.7.1.

Relativamente ao critério de seleção baseado na escala, selecionou-se um intervalo de fatores de escala com valores compreendidos entre 0.95 e 1.05, ou seja, com uma variação de escala de 0.05.

Resultados e Discussão

As tabelas 5.5 e 5.6 representam os resultados obtidos para cada um dos algoritmos tendo em consideração todo o conjunto de peças disponíveis. O tempo correspondente a cada peça é o tempo médio no ciclo onde existe uma correspondência.

Algoritmo	**Nº da peça**	**Tempo (s)**		**Nº Falsos Positivos**	**Nº Falsos Negativos**
		Média	**Desvio-padrão**		
ORB FLANN	1	0.26	0.10	3	1
	2	0.27	0.10	1	1
	3	0.23	0.08	0	0
	4	0.31	0.12	0	0
	5	0.31	0.12	1	0
	6	0.26	0.12	0	0
	7	0.27	0.11	1	1
ORB BF	1	0.24	0.10	3	1
	2	0.24	0.09	1	0
	3	0.24	0.10	3	0
	4	0.27	0.11	1	0
	5	0.26	0.10	1	1
	6	0.24	0.10	2	0
	7	0.24	0.09	4	1

Tabela 5.5 – Comparação dos métodos *Brute Force* e FLANN usando o ORB num conjunto de 144 testes para cada peça.

No que diz respeito à taxa de sucesso, o algoritmo ORB, na tabela 5.5, usando o FLANN, tem, em média, uma taxa de sucesso de 98.2% e o *Brute Force* uma taxa de 96.4%. Quanto ao SURF, na tabela 5.6, em média, o FLANN tem uma taxa de sucesso de 72.6% e o *Brute Force* uma taxa de 65.1%. O número bastante significativo de falsas correspondências no SURF deve-se ao facto de este ter poucas correspondências e, devido a isso, o cálculo do RANSAC acaba por resultar frequentemente numa matriz transformação em que a escala não é aproximadamente unitária, o que faz com que se seja considerada uma não correspondência.

Quanto ao tempo computacional exigido, com base nos resultados ilustrados nas tabelas 5.5 e 5.6, pode-se concluir que a vantagem temporal teórica que se poderia fazer sentir a favor do FLANN, não tem o impacto previsto. Na maioria dos casos, os tempos foram até alguns milissegundos superiores, levando a crer que os algoritmos da biblioteca FLANN apenas demonstram o seu maior impacto quanto maior é o número de *features*, a serem tratadas, ou seja, quando atinge uma ordem de grandeza de milhares de *features* o que não acontece nesta situação, pois o número de *features* é sempre inferior a 1000.

Tendo em conta a taxa de sucesso, a escolha recai sobre o ORB. Com este algoritmo, usando o método LSH, os tempos aumentam, em média, cerca de 10.3% e a taxa de sucesso aumenta cerca de 1.8%. Assim, optou-se por usar o método FLANN juntamente com o ORB, pois dá-se, uma vez mais, mais predominância à taxa de sucesso, onde o compromisso entre a eficácia e o tempos necessários no processo não é comprometido.

Assim, relativamente aos tempos computacionais, estes são na ordem dos 0.2 segundos a

0.3 segundos, para cada peça, enquanto que no método baseado somente na correlação, os tempos são cerca de 10 vezes superiores com tempos a rondar os 2 segundos a 3 segundos. Assim, uma abordagem baseada em *features* apresenta-se como uma melhor solução.

Algoritmo	**Nº da peça**	**Tempo (s)**		**Nº Falsos Positivos**	**Nº Falsos Negativos**
		Média	**Desvio-padrão**		
SURF FLANN	1	0.19	0.07	2	25
	2	0.26	0.07	1	16
	3	0.20	0.07	4	22
	4	0.35	0.12	2	4
	5	0.27	0.10	4	28
	6	0.26	0.10	0	17
	7	0.25	0.09	1	12
SURF BF	1	0.21	0.06	3	46
	2	0.23	0.09	1	7
	3	0.16	0.04	0	55
	4	0.31	0.12	0	3
	5	0.25	0.08	1	10
	6	0.25	0.08	0	23
	7	0.25	0.08	1	26

Tabela 5.6 – Comparação dos métodos *Brute Force* e FLANN usando o SURF num conjunto de 144 testes para cada peça.

5.7.3 Estimativa da orientação

No que à estimativa da orientação diz respeito, tiveram-se em consideração dois aspetos, nomeadamente, a rotação e a reflexão. A indicação da orientação das peças é publicada através dum tópico ROS, designado `/object/orientation`, de forma a informar o subsistema da manipulação qual a configuração da peça, como ilustrado na figura 5.6.

Uma vez determinada tanto a posição da peça, como a respetiva orientação, é também possível determinar a posição da marcação a laser. A base de dados contém as coordenadas do ponto de marcação segundo o referencial da peça modelo previamente armazenada. Este referencial está centrado com o centróide da própria peça. Assim, sabendo as coordenadas de marcação estas são publicadas no tópico designado `/object/laser`.

Quanto à reflexão, desenvolveu-se o algoritmo 3, onde se calcula a correlação, recorrendo à função *macthTemplate*. Assim, calcula-se o ângulo não só entre as *features* da imagem adquirida e a imagem modelo, presente na base de dados, como também, entre as *features* da imagem invertida e o mesmo modelo. De seguida, aplica-se a rotação estimada às imagem correspondentes e calcula-se a correlação máxima entre os dois conjuntos de imagens, num intervalo de -2° a +2°, com um incremento de 1 °. Neste processo a que obtiver a correlação máxima mais elevada corresponderá à orientação estimada.

Com o desenvolvimento do algoritmo 3, teve-se em consideração os 3 casos possíveis. A correspondência pode ocorrer com a peça não invertida, invertida, ou com ambas. O caso dito normal ocorre quando apenas existe correspondência com uma delas e essa é a correspondência

encontrada. No entanto, quando existe correspondência com ambas as imagens, compara-se a correlação máxima obtida em ambos os casos e o caso onde existir uma maior correlação corresponderá à orientação estimada. Este é o algoritmo responsável pela necessidade de armazenar uma imagem processada na base de dados.

Este processo de deteção de reflexões é totalmente desnecessário para peças em que a sua simetria não apresenta diferenças e, por essa razão, elaborou-se um outro algoritmo 4, responsável pela deteção de diferenças entre 2 imagens. De modo a garantir a viabilidade deste algoritmo, assume-se que o operador, quando captura a imagem modelo, a peça se encontra orientada sobre o eixo de simetria horizontal ou vertical da câmara.

Este algoritmo calcula a diferença entre duas imagens mas, tendo em conta o ruído existente e a dificuldade em fazer a exata sobreposição dos contornos das peças, o algoritmo percorre a imagem original, píxel a píxel, e apenas é realizada a subtração dos píxeis se na imagem espelhada existir um contorno da peça até uma distância máxima de 5 píxeis, como representado no algoritmo 4. Se a imagem resultante da subtração, ilustrada na figura 5.23, não tiver qualquer tipo de contornos, então o algoritmo dá a indicação de que a imagem não apresenta diferenças quando espelhada. Essa informação é armazenada na base dados (Figura 5.4), junto das restantes especificações de cada peça. Caso não exista diferença na sua simetria, não é necessário avaliar as correspondências para a imagem refletida. A variável booleana responsável pela indicação da simetria é denominada no algoritmo 2 como *modelo_com_reflexão*.

De forma a avaliar a estimativa da orientação, testaram-se os algoritmos desenvolvidos, por um lado, tendo em consideração o erro absoluto entre o ângulo real e o ângulo estimado e, por outro lado, avaliou-se, para o conjunto de peças com diferenças ao nível da simetria, quantas vezes o algoritmo falha nessa deteção.

Experiências

Os testes, de validação da estimativa do ângulo, realizaram-se para as 7 peças onde, para cada uma delas existe um conjunto de 72 imagens, como já foi mencionado, com um ângulo de rotação de 5°, até completar uma rotação completa. Quanto aos testes de deteção de reflexões, apenas se realizaram testes para as peça Nº2, Nº3 e Nº4, com o mesmo conjunto de imagens. Os testes baseiam-se em selecionar uma imagem de cada peça invertida e verificar o número de vezes que a deteção da peça é dada como não invertida, quando na realidade a peça está invertida. Para estes testes usou-se a função *estimateAffinePartial2D* numa configuração baseada no método RANSAC, com um máximo de 2000 iterações, e definiu-se o número mínimo das melhores correspondências e *inliers*, associados ao ORB, definidos na secção 5.7.1.

Resultados e discussão

A figura 5.23 apresenta 3 exemplos da aplicação do algoritmo 4, onde as figuras 5.23c, 5.23f, 5.23i são os resultados dessa aplicação. Como se pode verificar nas peças Nº3 e Nº4, as subtrações entre as imagens 5.23a e 5.23b, e entre 5.23d e 5.23e não são exatas, relevando-se as suas diferenças ao nível da simetria. No entanto, em relação à peça Nº5 o resultado é uma imagem com os píxeis todos pretos pois não existe distinções ao nível da simetria independentemente da sua orientação.

A tabela 5.7, apresenta o erro absoluto do ângulo estimado, para cada peça, no conjunto

Algoritmo 3 Reconhecimento de orientação das peças.

Input: p ▷ Variável indicativa se o ciclo é com a imagem original ou imagem invertida.
tentativa ▷ Variável indicativa da tentativa do RANSAC.
inliers ▷ Nº de *inliers* estimados.
ângulo ▷ Ângulo estimado entre a imagem original e o modelo da peça.
img, img_refletida ▷ Imagem original e refletida.
img_modelo ▷ Imagem modelo.
m_c_r ▷ Variável indicativa da simetria da peça.
Output: *ang_original* ▷ Indicação do ângulo da peça original reconhecida.
ang_refletida ▷ Indicação do ângulo da peça refletida reconhecida.

1: **procedure** ORIENTAÇÃO($p, tentativa, inliers, ângulo, img, img\_modelo, img\_refletida, m\_c\_r$)
2: $gama = 2$ ▷ Calcula-se a correlação no intervalo de ângulos: ângulo ± gama.
3: $msg =$ “”
4: $m =$ False ▷ Booleano indicativo de correspondência da peça.
5: $corr\_original = -1$ ▷ Correlação da imagem original.
6: $corr\_refletida = -1$ ▷ Correlação da imagem original refletida.
7: **if** $p == 0$ **then** ▷ Ciclo da imagem original.
8: **if** $inliers < min\_inliers$ **then**
9: $msg \leftarrow$ “Peça não reconhecida”
10: **end if**
11: $corr\_original, ang \leftarrow$ Correlação_Max($img, img\_modelo, ângulo$)
12: **if** $m\_c\_r ==$ False **then**
13: $msg \leftarrow$ “Peça reconhecida. Ângulo:” $+ ang$
14: **else**
15: $m =$ True
16: $ang\_original = ang$
17: **end if**
18: **end if**
19: **if** $p == 1$ **then** ▷ Ciclo da imagem refletida.
20: **if** $inliers < min\_inliers$ **then**
21: **if** $m ==$ True & $tentativa == 2$ **then**
22: $msg \leftarrow$ “Peça reconhecida. Ângulo:” $+ ang\_original$
23: **else**
24: $msg \leftarrow$ “Peça não reconhecida”
25: **end if**
26: **else**
27: $corr\_refletida, ang\_refletida \leftarrow$ Correlação_Max($img\_refletida, img\_modelo, ângulo$)
28: **if** $corr\_refletida > corr\_original$ **then**
29: $msg \leftarrow$ “Peça refletida reconhecida. Ângulo:” $+ ang\_refletida.$
30: **else**
31: $msg \leftarrow$ “Peça reconhecida. Ângulo:” $+ ang\_original.$
32: **end if**
33: **end if**
34: **end if**
35: **end procedure**

Algoritmo 4 Deteção simetria.

Input: *dist_min* ▷ Distância mínima entre píxeis.
img_original ▷ Imagem original.
img_refletida ▷ Imagem original refletida.
Output: *img_final* ▷ Imagem resultante da diferença das imagens.

```
dist_min = 5, dist_min_aux = 1000                    ▷ Distâncias entre píxeis.
píxel_preto = (0, 0, 0)
img_original ← Carregar_imagem()
img_refletida ← Refletir(img_original)
img_final = img_original
píxeis_brancos ← Coordenadas_píxeis_brancos(img_original)
píxeis_brancos_refletida ← Coordenadas_píxeis_brancos(img_refletida)
for i = 1, 2, ..., dim(píxeis_brancos) do
    for k = 1, 2, ..., dim(píxeis_brancos_refletida) do
        distância ← distância_píxeis(píxeis_brancos[i],píxeis_brancos_refletida[k])
        if distância < dist_min_aux then
            dist_min_aux = distância
        end if
    end for
    if dist_min_aux < dist_min then
        img_final[píxeis_brancos[i]] = píxel_preto
    end if
end for
```

de testes mencionados. Tendo em consideração o erro máximo definido, obteve-se, em algumas peças, valores que se encontram ligeiramente acima do idealmente pretendido. Como se pode verificar na figura 5.3 as dimensões da peça Nº1 e Nº2 são as maiores, enquanto que as peças Nº3, Nº6 e Nº7 são as mais pequenas. Este fator reflete-se no erro obtido, uma vez que quanto menor forem as dimensões da peça maior será a dificuldade em estimar a posição dela, dadas as limitações impostas pela resolução da própria câmara. Assim, existem dois aspetos que poderiam contribuir positivamente para a obtenção de resultados mais satisfatórios, por um lado, o uso de uma câmara com uma maior resolução, por outro lado, a redução da distância entre as peças e a câmara.

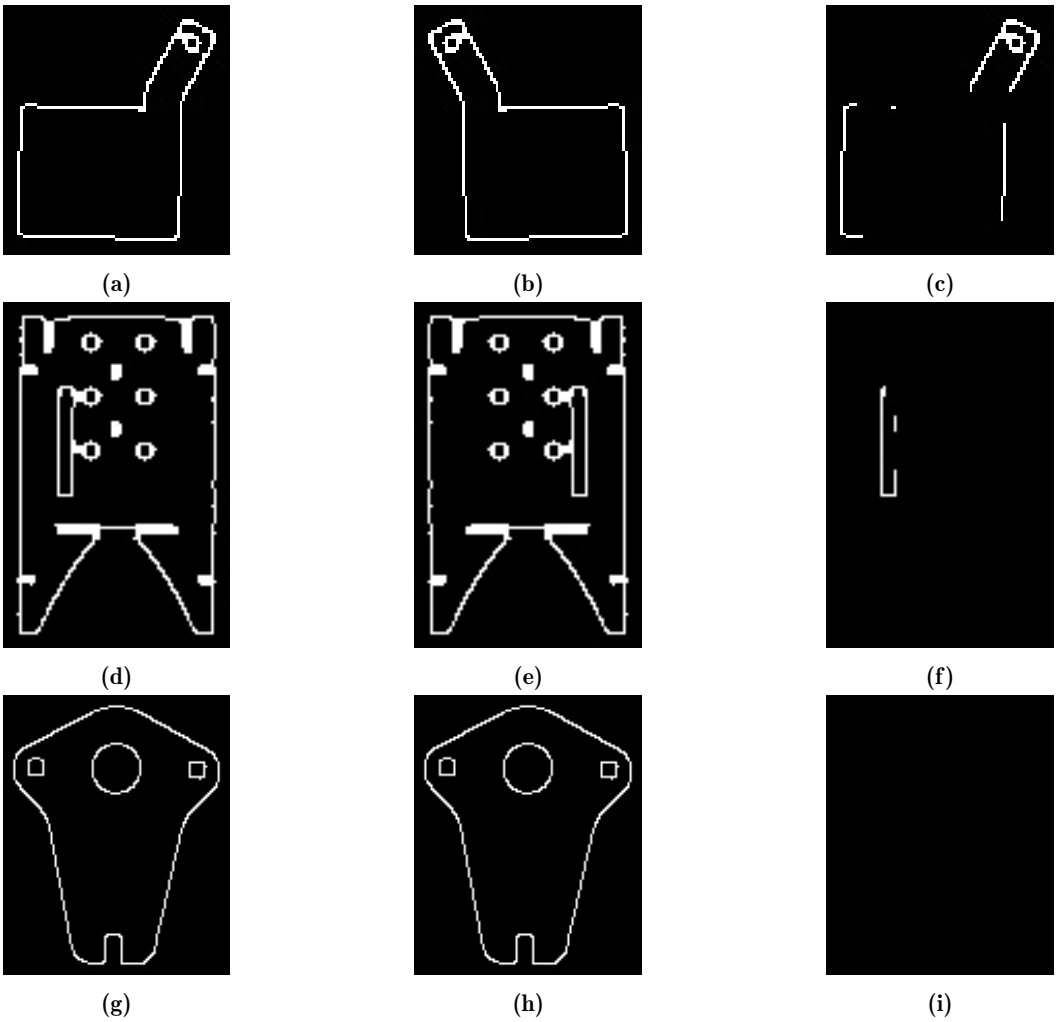


Figura 5.23 – Resultados da aplicação do algoritmo 4.

Nº da peça	Erro ângulo (°)	
	Média	Desvio-padrão
1	0.01	0.05
2	0.03	0.14
3	0.28	0.46
4	0.11	0.30
5	0.16	0.34
6	0.44	0.54
7	0.56	0.56

Tabela 5.7 – Erro absoluto do ângulo estimado num conjunto de 72 testes para cada peça.

A tabela 5.8 apresenta o número de falhas para as peças Nº2, Nº3 e Nº4, onde, para cada uma das peças, obteve-se uma taxa de sucesso na deteção de 100%, 98.6% e 94.4%, respetivamente. Dada as taxas apresentadas, acredita-se que se obtém a orientação da peça com um grau de certeza que é suficiente. O maior número de falhas na peça Nº4 deve-se ao facto de peça apresentar uma grande similaridade na sua reflexão e, por isso, faz com que a correlação máxima nos dois casos sejam muito próximas, logo aumenta a probabilidade de o algoritmo falhar.

Nº da peça	Nº de falhas
2	0
3	1
4	4

Tabela 5.8 – Número de falhas num conjunto de 72 testes para cada peça.

Uma vez validado este método de deteção da orientação da peça, importa comparar com o método inicialmente abordado na secção 5.6, onde os resultados obtidos são apresentados na tabela 5.4. Assim, o método que recorre às *features*, apresenta um erro médio significativamente inferior, com um erro médio máximo a rondar os 0.56°, na peça Nº7, enquanto que, no outro método ronda os 2° a 3°.

5.8 Correspondência de peças numa base de dados extensa

Os tempos de computação associados a cada peça, obtidos na tabela 5.5, são aceitáveis para uma pequena quantidade de peças, no entanto, num ambiente industrial, pode-se ter uma base dados com mais de 300 peças. Por esse motivo, realizaram-se testes com o objetivo de aferir o impacto dos tempos de processamento com uma base de dados com dimensões mais realistas, no ponto de vista de uma aplicação industrial.

5.8.1 Multiprocessamento

Iniciou-se o estudo comparando o comportamento da metodologia apresentada usando um sistema *single-process* e um sistema *multi-process*.

Desenvolveu-se o código em Python3 e, tendo consciência das limitações associadas a esta linguagem de programação, nomeadamente, em relação ao *Global Interpreter Lock* (GIL),

optou-se por usar processos em vez de *threads*. De uma forma resumida, o GIL é um bloqueio feito pelo interpretador Python, que apenas permite que uma *thread* mantenha o controlo do interpretador [97]. Com o uso dos processos, cada um obtém o seu próprio interpretador Python e espaço de memória para que o GIL não seja um problema, mesmo numa arquitetura *multithread* com mais de um núcleo no CPU.

No Python3, existe uma biblioteca, chamada *multiprocessing* [98], que permite criar um número específico de processos e fazer a gestão dos mesmos evitando o problema do bloqueio do interpretador. Assim, criou-se uma fila (*queue*) e cinco eventos (*events*) com fins bem definidos. A fila destina-se a retornar à função principal qual a peça reconhecida através do seu número identificativo. Relativamente aos eventos, um é responsável por dar indicação da existência de uma correspondência de um objeto, de modo a informar os restantes processos que não é necessário continuar a pesquisa, e os quatro restantes dão indicação do término de cada processo. Logo que os processos são iniciados, existe um ciclo de espera até que uma das duas condições seja estabelecida. Ou o evento da correspondência é lançado, ou os quatro eventos do termino de cada processo são emitidos, o que se traduz numa não correspondência com as peças da base de dados. Depois de ultrapassado este ciclo e caso exista uma correspondência, é verificado qual o número da peça identificada através da informação disponibilizada na fila.

Dado que o computador onde são realizados os testes desta secção é constituído por 2 núcleos físicos e 2 núcleos virtuais, como foi mencionado no início deste capítulo, é importante apresentar o *hyperthreading* [99]. Este conceito, de uma forma sintetizada, consiste em dividir os núcleos físicos em núcleos virtuais que são tratados como se fossem núcleos físicos pelo sistema operativo. Assim, consegue executar 2 processos diferentes no mesmo núcleo físico do CPU. O objetivo desta secção é também concluir qual o número de processos que permite retirar o maior proveito dos recursos do CPU.

Experiências

Com o objetivo de analisar o impacto temporal, numa base de dados extensa, usou-se a base de dados já criada para os testes anteriores. Assim, testou-se a correspondência para cada imagem numa base de dados que obriga a percorrer toda a base de dados até encontrar a peça correspondente. Realizaram-se testes em que a peça correspondente se encontrava nas posições 432, 100 e 50. Para efeitos de testes, assumiu-se que metade do número total de peças, presente na base de dados, assumia diferenças ao nível da reflexão.

Realizaram-se testes, por um lado, com apenas 1 processo e, por outro lado, com o acréscimo de 2 e 4 processos, além do processo principal, sem alterar as condições de teste, de forma a verificar apenas o impacto dos processos no tempo computacional.

Resultados e discussão

Com base nos resultados obtidos na figura 5.24, para um conjunto de 432 peças, é possível concluir que, usando 4 processos adicionais, o sistema é, em média, 80.7% mais rápido que o sistema quando usa 1 processo. Além disso, usando os 4 processos, o sistema é, em média, 18.7% mais rápido que o sistema quando usa apenas os 2 processos. O conceito *hyperthreading* é a explicação para este fenómeno, logo a indicação que dá é que com 2 processos o CPU não aloca todos os recursos disponíveis deixando de parte alguns recursos para a execução de outras *threads*, não retirando o máximo partido do CPU.

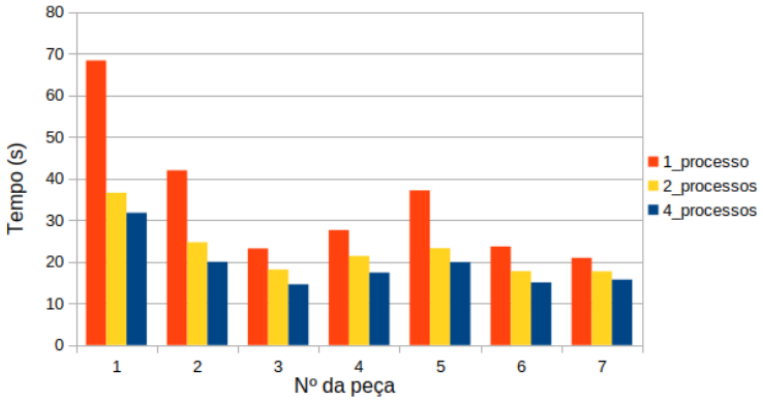


Figura 5.24 – Comparação dos tempos com 1, 2 e 4 processos num conjunto 432 peças.

As figuras 5.25 e 5.26 apresentam os resultados obtidos para um conjunto de 100 e 50 peças, respetivamente, com apenas 1 processo e com 4 processos adicionais, dado que no anterior teste, com 2 processos, a eficiência computacional era inferior. Num conjunto de 100 peças, usando 4 processos, o sistema é, em média 66.5% mais rápido do que o sistema com 1 único processo e, num conjunto de 50 peças, usando os mesmos 4 processos, o sistema é, em média 28.4% mais rápido. Com os testes realizados, pode-se concluir que a eficiência no uso dos 4 processos é tanto maior quanto maior a dimensão da base de dados. Acredita-se que isto ocorre devido á existência de uma sobrecarga inicial (*overhead*), que é independente do número de processos, quando estes são criados e inicializados. Portanto, quanto menor a dimensão da base de dados, maior é o impacto, desta sobrecarga, no desempenho do sistema.

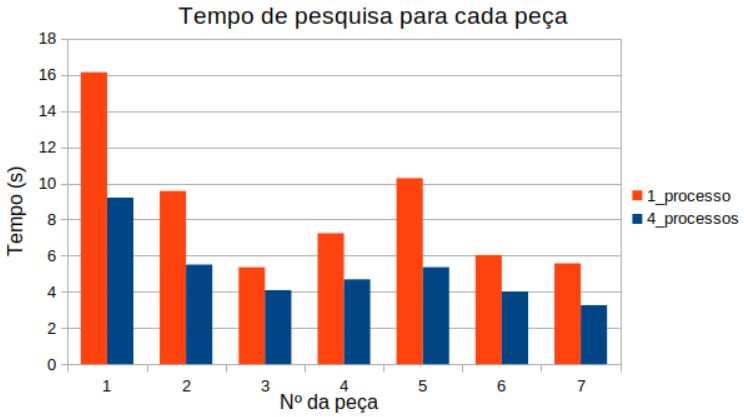


Figura 5.25 – Comparação dos tempos com 1 e 4 processos num conjunto 100 peças.

Contudo, os tempos apresentados são ainda insuficientes, uma vez que é inconcebível, tendo em conta os requisitos definidos, num conjunto de cerca de 430 peças demorar em média, por exemplo, para a peça Nº1 cerca de 32 segundos, ou num conjunto de apenas 50 peças demorar até cerca de 6 segundos a analisar toda a base de dados. Assim, desenvolveu-se um novo critério de pré-seleção baseado na área aproximada de cada peça, como se pode verificar na seguinte secção.

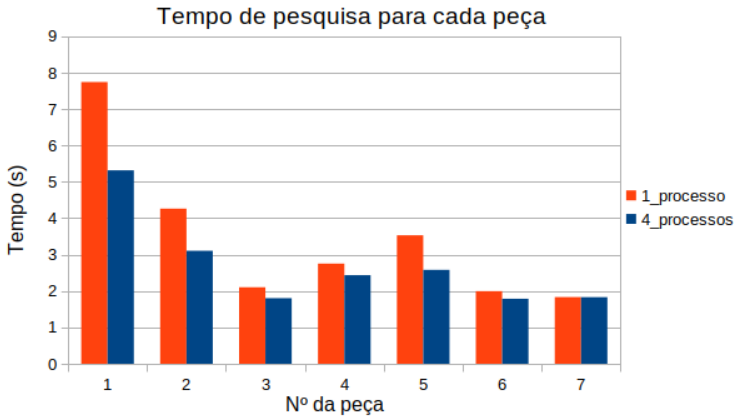


Figura 5.26 – Comparação dos tempos com 1 e 4 processos num conjunto 50 peças.

5.8.2 Filtragem baseada na área da peça

Dados os resultados da anterior subsecção, o objetivo é analisar o impacto com a adição de uma pré-filtragem baseada na área onde o propósito é reduzir o tempo computacional exigido.

Relativamente ao cálculo das áreas, esse é facilmente determinado com recurso a uma das propriedades dos contornos, apresentada na secção 2.5.4, que retorna a área aproximada do objeto, em número de píxeis diferentes de zero. De forma a garantir que as peças estão uniformemente distribuídas, de acordo com as áreas, pelos 4 processos, criou-se um algoritmo que divide as peças pela base de dados com base nas áreas da peças previamente armazenadas. Ou seja, na construção da base de dados, apresentado na figura 5.4, assim que é solicitado o armazenamento de uma nova peça, é calculada a sua área aproximada e guarda-se as peças por ordem crescente ou decrescente.

Assim, quando a pesquisa por correspondências é solicitada, o conjunto de peças armazenadas na base de dados origina dois vetores, um de imagens e outro de *features*, que por sua vez são distribuídos por 4 vetores, que são lançados em cada um dos 4 processos criados. Sabendo previamente que o vetor principal tem as peças armazenadas ordenadas segundo a sua área, a divisão pelos vetor auxiliares é feita de forma iterativa, como ilustrado na figura 5.27, onde os números representam áreas. Desta forma, garante-se a máxima eficiência no uso dos processos, uma vez que a exigência computacional está aproximadamente distribuída por igual pelos vários processos.

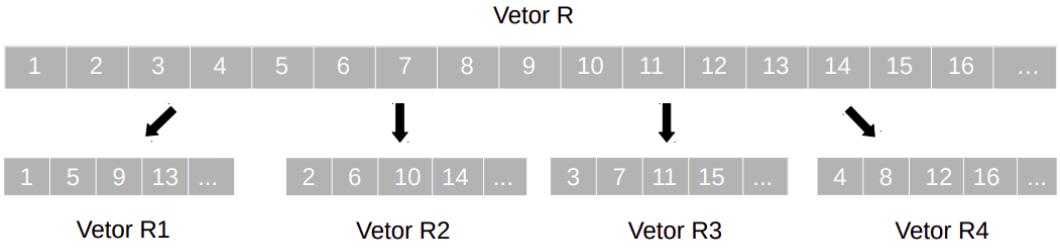


Figura 5.27 – Distribuição das peças baseado nas áreas, onde a numeração representa a área.

Experiências

Com o objetivo de analisar o impacto temporal, a base de dados extensa, usou-se uma base de dados já criada para os testes anteriores. Assim, usando 4 processos, testou-se a correspondência para cada peça numa base de dados que obriga a percorrer toda a base de dados até encontrar a peça correspondente.

Calcularam-se as áreas para um conjunto de 10 imagens da mesma peça e verificaram-se os seus valores médios e respetivos desvios-padrão, onde o propósito foi determinar área de cada peça, assim como a gama de tolerância a considerar.

Realizaram-se testes em que a peça correspondente se encontrava na posição 432, no entanto, para validar o impacto da filtragem assumiu-se que existiam peças com a mesma área da peça que se pretende corresponder. Seleccionou-se a peça Nº1 para estes testes, dado que foi a que obteve, normalmente, tempos computacionais mais elevados nos testes anteriores. Para efeitos de teste, considera-se que existia um determinado número de peças com a mesma área, nomeadamente, 10, 20, 40 e 100 peças com a mesma área, apesar de num caso prático se considerar que poderá existir, no máximo, entre 5 a 10 peças com a mesma área.

Resultados e discussão

As figuras 5.28a e 5.28b são as imagens resultantes, usando a função *findContours*, do cálculo aproximado das áreas, em píxeis, das peças Nº4 e Nº5, respetivamente.

(a) **(b)**

Figura 5.28 – Cálculo aproximado da área da peça. O ponto vermelho representa o centróide.

A tabela 5.9 apresenta as áreas, em média, de cada uma das peças, assim como, os respetivos desvios-padrão. Com base nestes dados, definiu-se, não só, o valor da área para cada peça definido pelo valor médio, assim como, uma margem de tolerância de 150 píxeis.

O gráfico da figura 5.29 apresenta os tempos computacionais obtidos com diferentes números de peças com a mesma área, num conjunto de 432 peças. Como se pode verificar os tempos reduzem-se significativamente quanto menor o número de peças com a mesma área.

Em suma, a filtragem baseada na área apresenta duas claras vantagens. Por um lado, o claro decréscimo do tempo computacional exigido, sendo que, no caso de existirem 10 peças com a mesma área, o subsistema demorará menos de dois segundos até encontrar

Nº da peça	Área (píxeis)	
	Média	Desvio-Padrão
1	58649	67
2	33523	45
3	9649	38
4	21217	106
5	23177	96
6	9235	83
7	6261	29

Tabela 5.9 – Área estimada para cada peça, num conjunto de 10 testes.

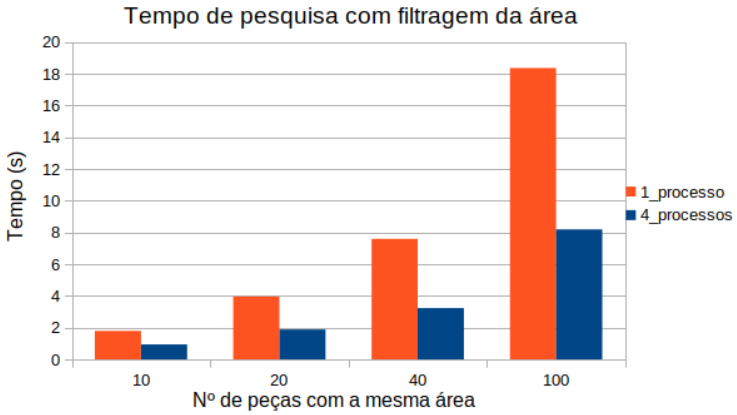


Figura 5.29 – Resultados usando a filtragem com base na área, num conjunto de 432 peças.

uma correspondência numa base de dados com 432 peças e calcular a respetiva localização e orientação. Por outro lado, aumenta a robustez no que diz respeito aos falsos positivos, dado que, caso as falsas correspondências sejam de peças com áreas distintas estas passam a ser facilmente suprimidas. Com base nos dados apresentados na tabela 5.9 e considerando a tolerância definida de 150 píxeis, o problema dos falsos positivos apresentado na tabela 5.5 é totalmente solucionado, pois nenhuma das peças apresenta a mesma área. Assim, com base nesses resultados e aplicando a filtragem baseada na área obtêm-se uma taxa de sucesso superior a 98.6%, nas condições de teste apresentadas.

Dadas estas conclusões, considera-se que foi obtido um subsistema de reconhecimento de peças com uma exigência computacional ajustada, tendo em conta o tempo máximo requisitado de 4 segundos. Ou seja numa base de dados com 432 peças, esta solução cumpre os requisitos mesmo com cerca de 40 peças com a mesma área da peça pesquisada. Além disso, apresenta para estes casos de teste uma taxa de sucesso superior a 98.6% o que se considera razoável num sistema deste género.

Capítulo 6

Metodologia e resultados na manipulação robótica

Neste capítulo é apresentado o trabalho prático direcionado para o subsistema de manipulação robótica de peças, assim como os testes, os resultados obtidos e a discussão dos mesmos. Ao longo de todo o capítulo são apresentadas e comparadas diferentes soluções para os problemas que foram enfrentados no decorrer deste trabalho, assim como as devidas justificações das decisões tomadas.

6.1 Introdução

O presente capítulo foca-se no desenvolvimento de soluções para o subsistema de manipulação das diferentes peças, depois de reconhecidas pelo subsistema de reconhecimento de peças. Segundo os requisitos apresentados na secção 1.2, pretende-se que a estação de manipulação seja constituída por um manipulador e uma garra capazes de manipular, devidamente, as peças presentes no tapete transportador e as forneça corretamente ao marcador a laser. Assim, é apresentado o trabalho desenvolvido em relação à calibração extrínseca da câmara, à manipulação de peças em ambiente simulado e real, assim como a marcação a laser das peças. O sistema físico usado é apresentado no capítulo 5 na figura 5.1, incluindo o manipulador UR10e com a respetiva garra de ventosas.

Tendo em consideração o material disponível no *Intelligent Robotics and Systems* (IRIS) *Laboratory*, optou-se por escolher o UR10e, pois apresenta características compatíveis com os requisitos mencionados. Assim, este manipulador dá garantias de compatibilidade com tarefas onde se privilegia a liberdade e a precisão dos movimentos, a velocidade de execução e a fácil implementação num ambiente industrial. Além disso, existe uma grande diversidade de garras que podem ser facilmente acopladas ao manipulador, de modo a adaptar o robô para qualquer tipo de tarefa, demonstrando assim a sua versatilidade. Quanto ao marcador a laser, não foi possível ter acesso a esse material e, como tal, realizaram-se os testes com uma pequena variante onde se usou um protótipo de um marcador acoplado ao manipulador.

6.2 Arquitetura proposta

O subsistema de manipulação robótica mencionado na figura 1.1 é, neste capítulo detalhadamente explicado, onde a arquitetura proposta está representada na figura 6.1. No subsis-

tema de reconhecimento de peças, após uma imagem de uma peça ser capturada, processada e comparada com a base dados, são publicados um conjunto de tópicos, onde é possível obter informações sobre a peça reconhecida, nomeadamente a sua dimensão, posição, orientação e posição da marcação a laser. O subsistema dedicado à manipulação subscreve esta informação e cria cenários de colisões, estima a posição e a orientação num referencial conhecido do robô e estima a posição da marcação nesse mesmo referencial. Tendo esta informação, é planeada a respetiva trajetória usando um planeador de trajetórias disponível no MoveIt!.

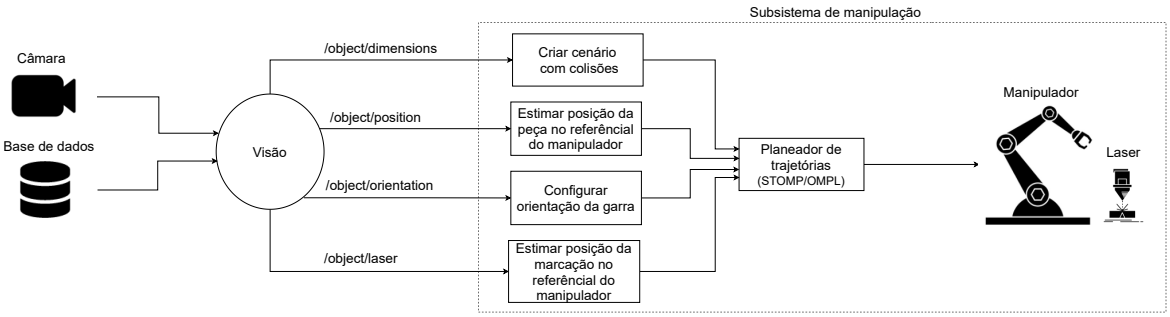


Figura 6.1 – Arquitetura do subsistema da manipulação robótica.

6.3 Configuração do MoveIt!

Como já referido no Capítulo 4, o MoveIt! é um conjunto de pacotes integrados com o ROS que disponibiliza algumas funcionalidades como, por exemplo, o planeamento de trajetórias. Assim, o MoveIt! analisa o URDF do robô e cria um ficheiro de configuração que dá a informação sobre os segmentos do manipulador que estão sempre em colisão e os que nunca podem colidir. Deste modo, minimiza os cálculos de colisões para futuros planeamentos e evita imprevistos futuros. Neste ficheiro, são também definidos dois grupos de manipulação, um chamado *manipulator* e outro *endeffector*. O grupo *manipulator* refere-se aos segmentos e juntas que constituem o robô, responsáveis pelo movimento do manipulador, enquanto que o grupo *endeffector* representa os segmentos e juntas que constituem a garra, sendo que este grupo é considerado a referência para as posições objetivo do planeamento.

Relativamente ao calculador de cinemática inversa, não se realizaram alterações ao calculador selecionado por defeito no MoveIt!. Trata-se de um calculador de cinemática inversa iterativo baseado no algoritmo *Kinematics and Dynamics Library* (KDL). No futuro poderão ser testados outros algoritmos como, por exemplo, o IKFast.

6.4 Planeadores de trajetórias

O objetivo deste secção é apurar qual o planeador que oferece melhores garantias usando o UR10e, nesta aplicação em específico. Nesta fase, retirou-se partido das vantagens do simulador Gazebo e os testes são realizados em ambiente de simulação, de forma a serem mais fáceis, rápidos e sem desgaste do *hardware*. Existem diversos planeadores disponíveis no MoveIt! e não é o objetivo desta dissertação testá-los a todos. Por esse motivo, realizou-se alguma pesquisa de casos de estudo relacionados com esta temática, de forma a verificar quais são os que apresentam, por norma, os melhores resultados.

Com base nos casos de estudo apresentados na secção 3.5.4, realizaram-se testes com os seguintes algoritmos: RRTConnect, PRM* e BiTRRT. Além destes, realizaram-se também testes com o planeador STOMP, dadas as características apresentadas. A análise dos planeadores baseou-se nas seguintes métricas:

- Variação das juntas - Resultado do somatório das diferenças absolutas dos valores das posições das juntas, em radianos, entre cada ponto consecutivo do planeamento;
- Tempo de planeamento - Tempo total despendido para solucionar o planeamento da trajetória;
- Tempo de execução - Tempo calculado pelo planeador para concluir a trajetória;
- Distância percorrida pela garra - Distância percorrida pela garra, em metros, na execução do planeamento;
- Taxa de Sucesso - Percentagem de testes realizados com sucesso.

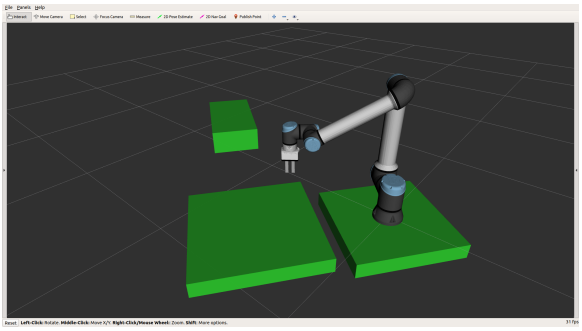
As trajetórias ROS, abordadas na secção 4.3.1, são definidas como um conjunto de pontos representado por mensagens do tipo *JointTrajectory*. Com vista a registar as variações das juntas para o conjunto de pontos planeados, calculou-se o módulo das diferenças das posições das juntas, em radianos, obtendo-se a variação total das juntas em toda a trajetória. Relativamente ao tempo de execução, é dado pelo campo *time_from_start* da mensagem criada pelo planeador. Quanto à distância percorrida, recorreu-se à classe *tf.TransformListener*, mais concretamente à função *lookupTransform*, onde se calcula a transformação entre os segmentos *base_link* e *gripper_link*, obtendo-se a matriz transformação que as relaciona que permite calcular a distância percorrida pela garra. Considera-se que o planeador falha quando excede o tempo de planeamento definido. Quando ocorre a falha, é realizado o replaneamento da trajetória e, caso falhe várias vezes seguidas, há 2 soluções possíveis: mover ligeiramente o manipulador para mudar as circunstâncias do planeamento, ou optar, pontualmente, por outro planeador.

De forma a selecionar o planeador que melhor se adequa ao trabalho desenvolvido nesta dissertação, elaborou-se um sistema de pontuação, normalizado para o pior caso, onde são atribuídos pesos percentuais a 3 dos fatores, que se consideram ser dos mais importantes na área industrial, nomeadamente o tempo de planeamento e execução, a eficiência energética e a taxa de sucesso. Quanto ao tempo, está associada a cadência dos processos automáticos, enquanto que a eficiência refere-se aos gastos energéticos no funcionamento dos processos, ou seja, as variações das juntas e a distância percorrida pela garra. Por fim, e o mais importante, a taxa de sucesso, ou seja, o número de vezes que os processos trabalham corretamente. Dada esta descrição, atribuíram-se as seguintes percentagens: 35% aos tempos de planeamento e execução, 20% à eficiência energética e 45% à taxa de sucesso. A atribuição destas percentagem não foi baseada em dados concretos, são apenas pesos relativos que parecem fazer sentido, tendo em conta as prioridades num sistema deste tipo.

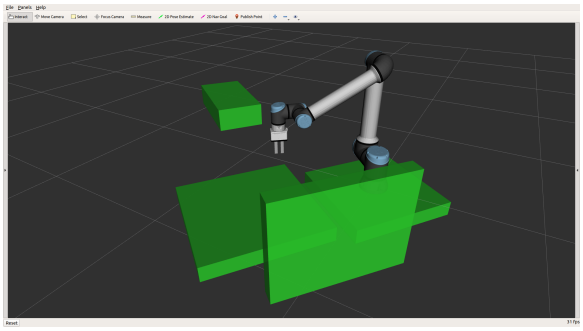
Experiências

De forma a visualizar os testes realizados, utilizou-se o RViz, pois este tem um conjunto de ferramentas que permitem visualizar o manipulador assim como as respetivas trajetórias planeadas e os objetos de colisão criados como, por exemplo, ilustrado na figura 6.3.

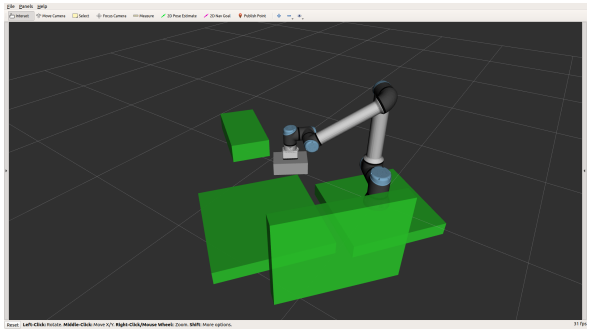
Definiram-se 3 cenários de testes, com diferentes restrições de movimento, logo o que os distingue é, essencialmente, o grau de dificuldade exigido. Para todos os cenários é exigido ao manipulador que crie 4 planeamentos, simulando um movimento típico neste tipo de aplicação, ou seja, o manipulador posiciona-se numa posição inicial, depois posiciona-se sobre a peça, de seguida, retorna a peça ao marcador a laser e, posteriormente, volta para a posição inicial. Para cada trajetória, de 4 planeamentos, realizaram-se 100 testes.



(a) Cenário 1.



(b) Cenário 2.



(c) Cenário 3.

Figura 6.2 – Ilustração dos 3 cenários de teste com diferentes graus de dificuldade.

No cenário 1, apenas são consideradas as colisões da mesa, do manipulador, do tapete transportador, da câmara e dos segmentos do próprio manipulador (Figura 6.2a). No cenário 2, é exigido ao manipulador que crie os planeamentos tendo como base as colisões do cenário 1. No entanto, é ainda adicionado um objeto de colisão próximo da zona do marcador a laser, obrigando a que o manipulador contorne esse objeto limitando assim, os seus movimentos (Figura 6.2b). No cenário 3, é solicitado ao manipulador que crie um planeamento considerando o cenário de colisões do cenário 2, contudo, é ainda adicionado um objeto de colisão alocado à garra do manipulador (Figura 6.2c).

No cenário 3, de modo a obter colisões dinâmicas, subscreveu-se o tópico `object/dimensions`, que é publicado pelo nó ROS responsável pela subsistema de visão. Assim, sabendo a dimensão da peça a manipular, acoplou-se à garra do robô, um objeto paralelepípedo com as proporções conhecidas. Contudo, usando a função *attach_box* da classe *PlanningSceneInter-*

face, persistiu o problema do objeto criado não detetar colisões com o resto do cenário de colisões criado. Após inúmeras tentativas de aferir a origem do problema, não se conseguiu chegar a uma solução, logo optou-se por uma alternativa que apenas é válida para efeitos de teste. Deste modo, modificou-se o URDF do manipulador UR10e e criou-se um *link* com uma junta estática acoplada ao *frame* da garra do manipulador (*gripper_link*) com o formato de um objeto paralelepípedo de dimensões 20x15x5cm (Figura 6.2c). Esta alternativa não é de todo dinâmica e, por essa razão, é apenas uma aproximação bruta das dimensões dos objetos que se pretende manipular.

Dentro do diversificado conjunto de planeadores existentes, o BiTRRT e o PRM* pertencem aos algoritmos que otimizam a solução usando todos o tempo de planeamento especificado. Assim, obteve-se o tempo de planeamento especificado com base num método iterativo, onde se incrementava o tempo de planeamento e verificava-se se era tempo suficiente para o planeador conseguir calcular as trajetórias.

Para o cenário 1 e 2, consideraram-se os tempos de planeamento 0.2 e 0.4 segundos para os algoritmos PRM* e BiTRRT. Apesar de os restantes algoritmos não otimizarem a solução com base no tempo disponível, definiu-se também o tempo de planeamento limite. Nos 3 cenários definiu-se o tempo de 2 segundos para o planeador RRTConnect. Quanto ao STOMP definiu-se 2 segundos para o cenário 1 e 2, no entanto, no cenário 3 definiu-se o tempo limite de 6 segundos. De forma a verificar o comportamento destes algoritmos com um limite temporal superior, testou-se com mais 2 segundos que o tempo mínimo estimado. Estes tempos foram definidos de forma a aferir o impacto do limite do tempo de planeamento no compromisso entre o tempo de planeamento e a taxa de sucesso, apesar de não existir otimização de trajetória.

Relativamente à velocidade de execução do manipulador esta foi definida como 20% da velocidade máxima permitida pelo mesmo. De notar que esta componente tem impacto no tempo de execução e, por essa razão, foi mantida a mesma velocidade para todos os testes.

Resultados e discussão

A figura 6.3 representa a visualização, no Rviz, das trajetórias executadas no cenário 3, tendo em consideração o cenário de colisões criado.

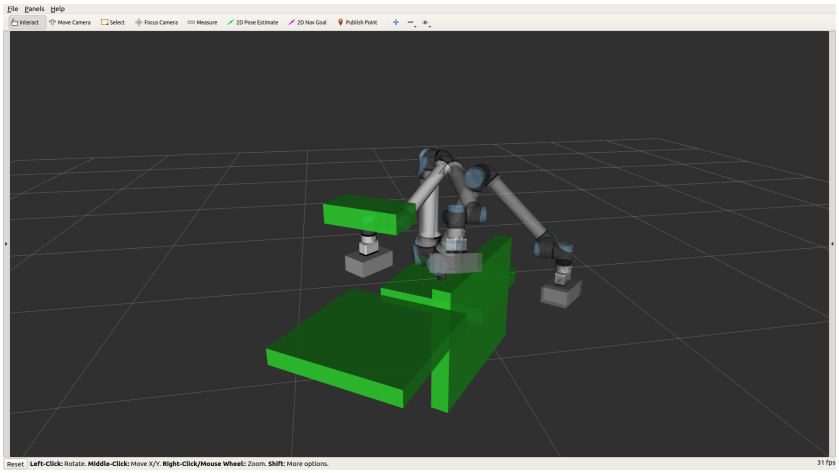


Figura 6.3 – Exemplo de uma trajetórias planeada e executada pelo planeador STOMP. As várias posições representam alguns dos pontos da trajetória executada.

As tabelas 6.1, 6.2 e 6.3 representam os resultados obtidos dos vários planeadores, em diferentes condições de teste, nomeadamente, nos cenários 1, 2 e 3, respetivamente. De notar que, para cada trajetória, são realizados 4 planeamentos e, por isso, os resultados apresentados nas tabelas são o somatório dos 4 planeamentos.

Planeador	**Variação das juntas (rad)**		**Tempo de planeamento (s)**		**Tempo de execução (s)**		**Distância percorrida (m)**		**Sucesso (%)**
	Média	**Desvio-padrão**	**Média**	**Desvio-padrão**	**Média**	**Desvio-padrão**	**Média**	**Desvio-padrão**	
RRTConnect (2 s)	7.93	5.48	0.12	0.04	8.88	2.31	4.38	2.25	100
STOMP (2 s)	4.17	0.13	1.37	0.28	8.79	0.04	3.12	0.00	100
PRM* (0.2 s)	7.50	5.27	0.85	0.02	8.79	2.22	4.29	2.21	100
PRM* (0.4 s)	7.67	5.37	1.67	0.03	9.13	2.56	4.53	2.39	100
BiRRT (0.2 s)	8.55	6.09	0.20	0.06	10.02	2.72	4.75	2.45	100
BiRRT (0.4 s)	7.98	5.76	0.17	2.54	9.78	2.54	4.51	2.35	100

Tabela 6.1 – Resultados de um conjunto de 100 testes para cada planeador no cenário 1.

Planeador	**Variação das juntas (rad)**		**Tempo de planeamento (s)**		**Tempo de execução (s)**		**Distância percorrida (m)**		**Sucesso (%)**
	Média	**Desvio-padrão**	**Média**	**Desvio-padrão**	**Média**	**Desvio-padrão**	**Média**	**Desvio-padrão**	
RRTConnect (2 s)	12.21	5.23	0.12	0.04	10.74	2.45	6.05	2.19	100
STOMP (2 s)	5.01	0.32	5.10	1.09	9.12	0.33	3.57	0.60	100
PRM* (0.2 s)	10.11	5.21	0.85	0.02	9.82	2.40	4.91	1.89	98.5
PRM* (0.4 s)	9.73	5.99	1.67	0.02	9.34	2.13	4.47	1.87	99.25
BiTRRT (0.2 s)	10.81	6.07	0.22	0.05	10.19	2.62	4.80	1.40	100
BiTRRT (0.4 s)	10.59	6.66	0.18	2.61	9.98	2.61	4.63	1.52	100

Tabela 6.2 – Resultados de um conjunto de 100 testes para cada planeador no cenário 2.

A nível energético, o STOMP destaca-se em todos os testes, dado que é o que tem uma menor variação das juntas e, consequentemente, uma menor distância percorrida pela garra. Além disso, é o que apresenta um menor desvio-padrão, tanto nas variações das juntas como nas distâncias percorridas pela garra, de modo que é o melhor em termos de repetibilidade de movimentos.

De um modo geral, os tempos de planeamento mantém-se ou aumentam ligeiramente, do cenário 1 para o 2, dadas as restrições de movimento exigidas, onde o STOMP é o que sofre um aumento mais significativo. No cenário 3, a complexidade é acrescida e isso reflete-se no incremento significativo dos tempos em todos os algoritmos. O RRTConnect destaca-se pelo facto de obter os tempos mais reduzidos no conjunto dos 3 testes. No cenário 1 e 2, apresenta uma boa repetibilidade, no entanto, no cenário 3, o desvio-padrão aumenta significativamente. No cenário 1 e 2, o STOMP é o mais lento e no cenário 3 é, indiscutivelmente, o PRM* o mais lento.

Relativamente aos tempos de execução, o STOMP é o planeador com tempos mais reduzidos no conjunto dos testes nos 3 cenários, além de que apresenta sempre um baixo desvio-padrão quando comparado com os restantes planeadores. Deste modo, é o melhor em termos de repetibilidade no tempo de execução.

Em termos de taxa de sucesso nos cenários 1 e 2, as diferenças que existem não são significativas, sendo que todos apresentam um sucesso de 100% ou superior a 98.5%, logo

Planeador	Variação das juntas (rad)		Tempo de planeamento (s)		Tempo de execução (s)		Distância percorrida (m)		Sucesso (%)
	Média	Desvio-padrão	Média	Desvio-padrão	Média	Desvio-padrão	Média	Desvio-padrão	
RRTConnect (2 s)	17.13	6.40	1.18	2.21	12.54	2.21	6.22	2.33	80.25
RRTConnect (4 s)	16.68	5.95	2.35	1.96	12.35	1.96	6.40	2.11	81.75
STOMP (6 s)	5.48	0.14	6.23	0.06	9.35	0.06	3.89	0.27	96.75
STOMP (8 s)	5.55	0.11	6.67	0.05	9.42	0.05	3.95	0.04	99.50
PRM* (3 s)	13.63	6.79	12.24	2.01	11.13	2.01	4.61	1.42	75.75
PRM* (5 s)	14.55	6.86	20.23	2.65	11.17	2.65	4.54	1.81	83.25
BiTRRT (3 s)	13.67	8.73	1.81	3.21	11.22	3.21	5.46	3.05	82.75
BiTRRT (5 s)	12.56	7.48	2.70	1.37	12.60	2.32	5.13	1.51	86.25

Tabela 6.3 – Resultados de um conjunto de 100 testes para cada planeador no cenário 3.

nenhum se destaca particularmente. Quanto aos testes no cenário 3, já apresentam uma redução considerável da taxa de sucesso, sendo que o STOMP (6 s) e (8 s) são os que mais se destacam com a maior taxa de sucesso, seguido do BiTRRT (3 s).

No conjunto dos 3 cenários, o aumento da especificação do tempo de planeamento nos algoritmos PRM* e BiTRRT, não contribui consideravelmente para a qualidade do planeamento nem para o aumento significativo da taxa de sucesso, ao ponto de compensar o incremento de tempo de planeamento. Relativamente ao cenário 3, testou-se o RRTConnect e o STOMP com mais 2 segundos em relação ao tempo mínimo de planeamento determinado. Em termos energéticos, os planeadores encontraram uma solução com menos variações das juntas, no caso do RRTConnect, e com mais variações, no caso do STOMP, sendo que ambas apresentaram uma distância superior percorrida pela garra. Quanto à taxa de sucesso, aumentou ligeiramente em ambos os algoritmos, sendo que no caso do STOMP (8 s) obteve-se uma taxa de sucesso de 99.5% e os tempos de planeamento pouco aumentaram em relação aos testes onde se usou 6 segundos de tempo de planeamento.

Tendo em consideração os resultados obtidos na figura 6.4a e 6.4b, os planeadores apresentam pontuações relativamente semelhantes, onde nenhum se destaca particularmente. No entanto, quando o grau de dificuldade aumenta, como ilustrado na figura 6.4c, o STOMP, destaca-se significativamente. Além disso, o STOMP apesar de não ser o planeador com maior pontuação no cenário 1 e 2, acaba por ser dos mais bem pontuados. Dadas estas observações, conclui-se que o STOMP é o melhor planeador para o UR10e nestas condições de teste.

Além disso, tendo em atenção os requisitos definidos de um tempo associado à manipulação a rondar, no máximo, os 8 segundos verifica-se que o STOMP não compromete o requisito, dado que nos 3 cenários esse tempo nunca é excedido. O tempo de execução claramente excede esse requisito, no entanto isso apenas se sucede devido ao facto de os testes terem sido realizados a apenas 20% da velocidade máxima de operação.

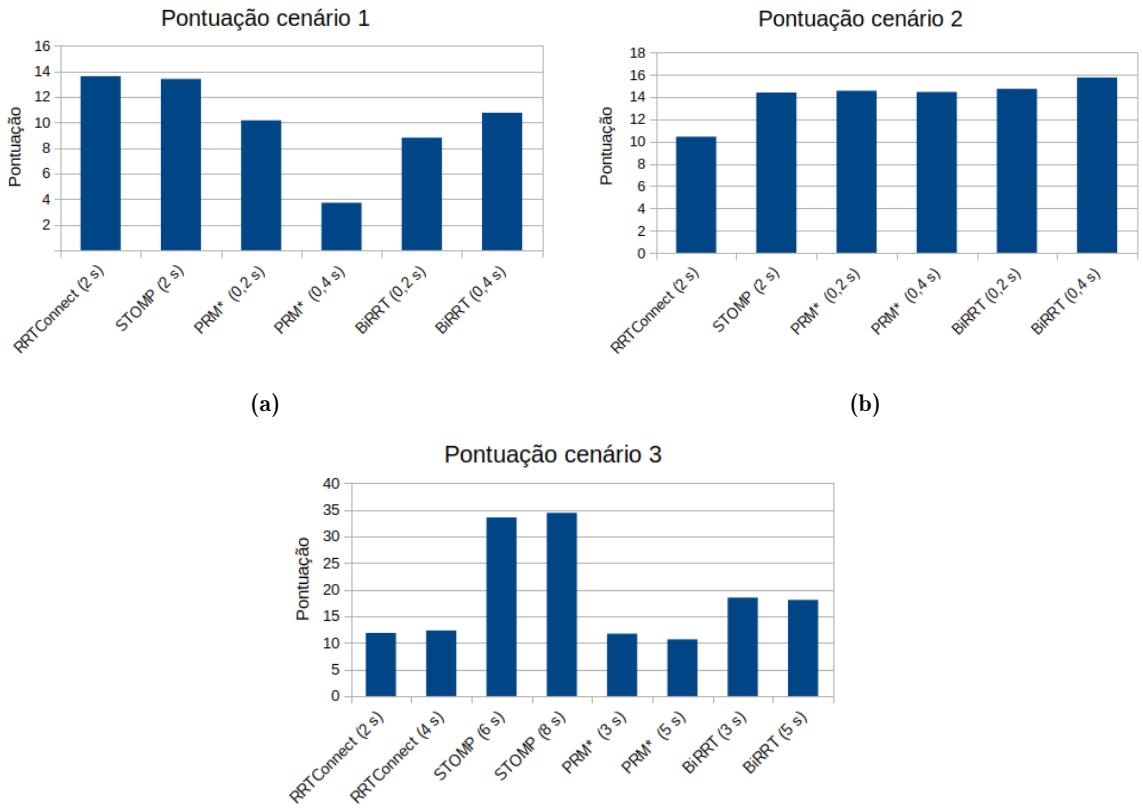


Figura 6.4 – Comparação das pontuações dos planeadores nos cenários 1, 2 e 3.

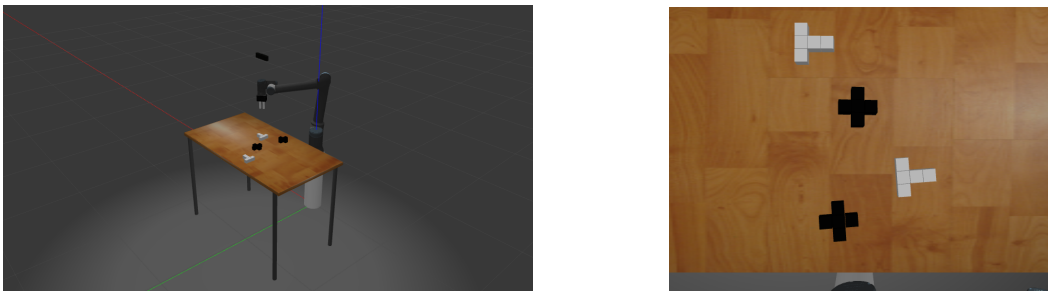
6.5 Manipulação de objetos em ambiente simulado

Nesta secção, tendo consciência das diferenças entre o ambiente simulado e o ambiente real, avaliou-se a capacidade do subsistema de manipulação de manipular os objetos previamente identificados pelo subsistema de visão. Assim, desenvolveu-se um mundo em ambiente de simulação, no Gazebo, que se assemelha à situação do problema real, ou seja, constituído por uma mesa onde se encontram vários objetos que se pretendem manipular, o modelo do robô UR10e sobre uma base de apoio e uma câmara posicionada sobre a mesa, como se pode verificar na figura 6.5a. Para isto, foi necessário também adicionar alguns *plugins* dedicados à garra e à câmara, de forma a permitir a correta simulação do sistema no Gazebo.

Para efeitos de demonstração, colocaram-se de forma aleatória objetos de diversos formatos em cima de uma mesa e a figura 6.5b ilustra a imagem publicada pela câmara através do tópico `/camera/rgb/image_raw_sim`. Posteriormente, a imagem é sujeita aos algoritmos de pré-processamento e reconhecimento e deteção de objetos apresentados no capítulo 2.

Manipulador

Relativamente ao modelo do robô UR10e, usou-se um pacote chamado *iris_ur10e* [100], onde é possível encontrar o modelo URDF, os controladores do manipulador, as configurações



(a) (b)

Figura 6.5 – Ilustração do mundo criado no simulador Gazebo na figura 6.5a e uma imagem capturada pela câmara em ambiente de simulação na figura 6.5b.

do MoveIt!, assim como os seus *drivers*. Quanto à componente de controlo da manipulação propriamente dita, recorreu-se ao pacote *iris_sami* [101]. Este pacote é uma interface de manipulação desenvolvida no IRIS *Lab*, onde é possível encontrar bibliotecas dedicadas à manipulação tanto do robô como da garra, usando os planeadores disponíveis no MoveIt!.

Garra

O modelo URDF do UR10e usado não dispõe da capacidade de agarrar e manipular objetos. Logo, caso se tentasse agarrar os objetos, os modelos apresentavam comportamentos indesejados como oscilar ou escorregar, pois o mecanismo de física não está otimizado para gerir este tipo de colisões propositadas. A definição, no URDF, de um *plugin* dedicado a este objetivo permite contornar este problema.

Usou-se o plugin *libgazebo_grasp_fix.so* [102] responsável por fixar o objeto que quer manipular ao modelo do robô, mais concretamente aos segmentos das garras do robô, nomeadamente *left_finger_link* e *right_finger_link*. Um objeto é agarrado assim que duas forças opostas são aplicadas pelos segmentos da garra e o objeto é fixado através de um *link* estático. Assim que estas forças deixam de ser aplicadas, o objeto é libertado novamente.

O *plugin* permite definir um conjunto de parâmetros como, por exemplo, a taxa de atualização, o ângulo de tolerância das forças, a distância de tolerância entre o objeto e as garras, entre outros. Os valores padrão foram mantidos e apenas se alteraram as assinaturas dos *links* associados, nomeadamente:

- `arm_name`: ur10e;
- `palm_link`: wrist_3_link;
- `gripper_link`: left_finger_link;
- `gripper_link`: rigth_finger_link.

Câmara

Tendo em conta os objetivos, era também necessário adicionar um *plugin* que permitisse simular um sensor óptico para a aquisição de imagens sobre a forma de tópicos. Usou-se o *plugin libgazebo_ros_camera.so* que permite definir um conjunto de parâmetros como, por

exemplo, o formato e a resolução da imagem, a taxa de atualização, a nomenclatura dos tópicos a publicar, os coeficientes de distorção, entre muitos outros. Definiram-se os seguintes parâmetros:

- `Resolução`: 640x480 píxeis;
- `Formato`: BGR8;
- `Taxa de atualização`: 30 *fps*;
- `Tópico da imagem`: /camera/rgb/image_raw_sim;
- `Distorção`: Sem distorção.

Experiências

A simulação começa com o manipulador na sua configuração inicial, como ilustrado na figura 6.5a, e, posteriormente, é repetida a seguinte sequência de trajetórias, para cada um dos objetos a manipular:

1. O nó dedicado à manipulação subscreve os 4 tópicos publicados pelo nó responsável pelo reconhecimento de objetos, nomeadamente os tópicos `/object/position`, `/object/orientation`, `/object/dimensions` e `/object/laser`, onde é possível obter informações sobre os objetos a manipular, mais concretamente, a sua posição, orientação, dimensão e posição da marcação a laser;
2. É planeada e executada uma trajetória, com base na indicação de uma posição global. Isto faz com o que o manipulador apresente uma configuração onde a garra se encontra sobre o objeto a cerca de 10 cm, garantindo que esta se encontra virada para baixo e com a mesma orientação da peça (Figura 6.6a);
3. É planeado um movimento relativo vertical de aproximação ao objeto mantendo a configuração da garra (Figura 6.6b);
4. A garra é acionada e agarra o objeto (Figura 6.6c);
5. É planeado, novamente, um movimento relativo vertical, em sentido contrário, retornando à posição do ponto 2 (Figura 6.6d);
6. O manipular desloca o objeto para uma posição de descarga de objetos e a garra larga o objeto (Figura 6.6e e 6.6f). Depois repete-se o ciclo.

Resultados e discussão

A figura 6.6 ilustra o procedimento de manipulação testado no simulador Gazebo. As trajetórias planeadas e as posições objetivo para a garra vão sendo desenhadas no RViz, no decorrer da simulação.

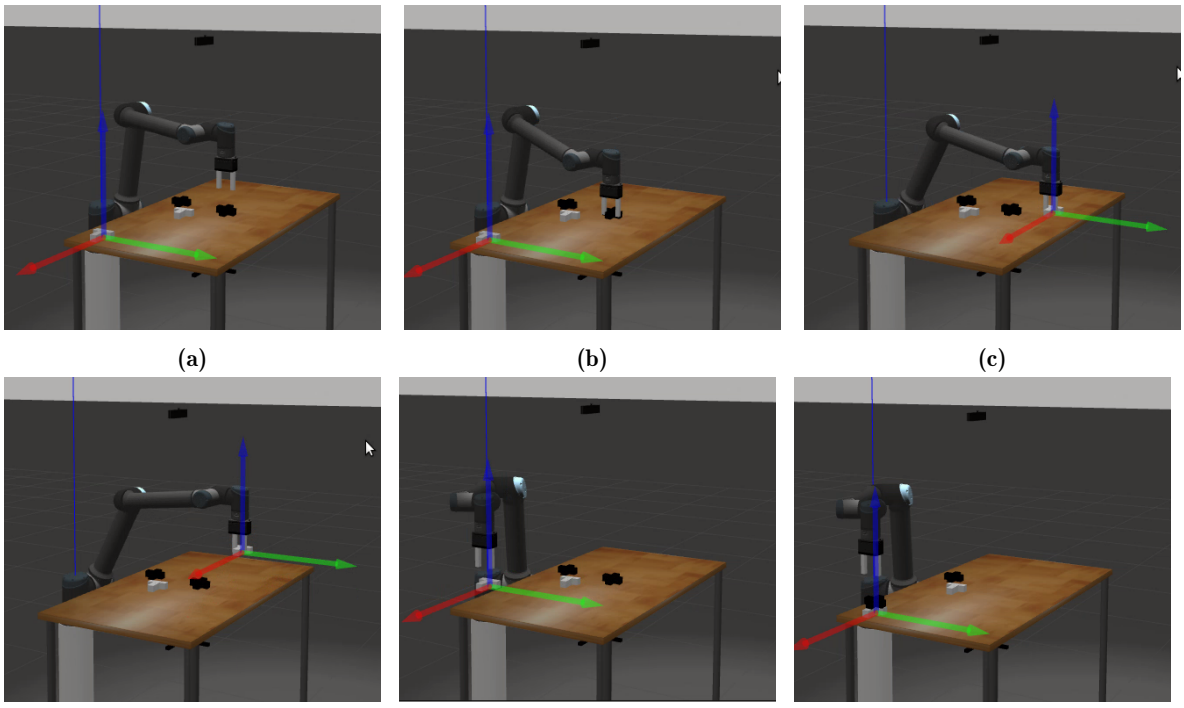


Figura 6.6 – Manipulação de objetos usando uma câmara que fornece a localização das peças.

Esta experiência valida a manipulação de objetos em ambiente simulado, dando boas indicações do seu funcionamento na prática. Contudo, existem alguns fatores importantes que fazem com que esta simulação se diferencie bastante da aplicação prática. Por exemplo, a câmara trata-se de um modelo ideal sem distorções, o que na realidade não se verifica. Além disso, os materiais usados nos objetos são constituídos de madeira e com uma altura de cerca de 2 cm, o que não representa a realidade, dado que, as peças são essencialmente constituídas por aço inoxidável e com cerca de 2 mm de altura. Logo a garra deverá ser, obrigatoriamente, de outro tipo.

6.6 Calibração extrínseca

No simulador, é facilmente determinada a posição e a orientação exata da câmara, uma vez que estas são definidas no ficheiro SDF que descreve o mundo. No entanto, numa aplicação real, o processo não é assim tão simples e, por essa razão, é necessário usar a calibração *Hand-eye* abordada na secção 3.3. Para este efeito, recorreu-se ao pacote *aruco_ros* em conjunto com o pacote *easy_hand_eye* [103] de forma a proceder à calibração extrínseca da câmara. O pacote *aruco_ros* é responsável por detetar o ArUco e reconhecer a respetiva posição e orientação do mesmo. Assim, desenvolveu-se um nó ROS responsável por subscrever os serviços disponibilizados pelo pacote *easy_hand_eye* e retirar sucessivas amostras do ArUco.

O que se pretende, na prática, é saber a posição e orientação exata da câmara relativamente ao manipulador, mais concretamente à tf da base do robô (*base_link*). Após a calibração, obtém-se uma matriz transformação entre a *base_link* e a tf criada para a câmara (*camera_link*), como representado na figura 6.7. Depois de realizada a calibração é pretendido

que se determine o erro associado.

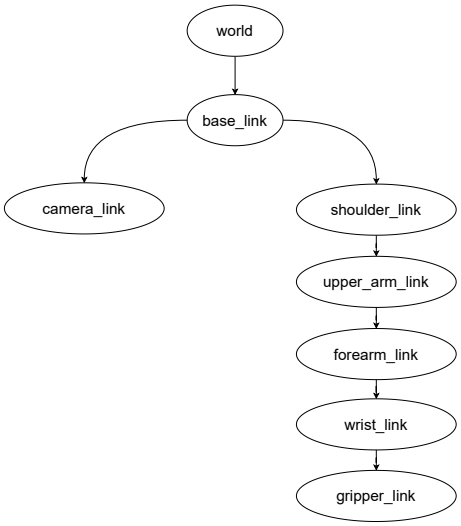


Figura 6.7 – Árvore tf simplificada do sistema.

Experiências

(a) **(b)**

Figura 6.8 – Processo da calibração (a) e validação (b) da calibração extrínseca *Hand-eye*.

Subscrevendo os serviços disponibilizados pelo pacote *easy_hand_eye*, nomeadamente, o `/take_sample`, o `/compute_calibration` e o `/save_calibration`, e seguindo as indicações abordadas na secção 3.3, acoplou-se o ArUco à garra do manipulador e posicionou-se o robô sob a câmara a cerca de 20 cm, como ilustrado na figura 6.8a. De seguida, com rotações de cerca de $\pm\pi/16$ rad nos eixos *pitch* e *yaw*, como ilustrado na figura 6.9, adquiriram-se cerca de 50 amostras, sendo que, para cada posição, são retiradas 3 amostras, prevenindo a existência de amostras *outliers*. Para a visualização do *frame* criado para o ArUco usou-se a ferramenta `rqt_image_view`.

Com o objetivo de verificar a exatidão da calibração extrínseca, usou-se a deteção da posição e orientação do ArUco e, posteriormente, ordenou-se ao manipulador para se po-

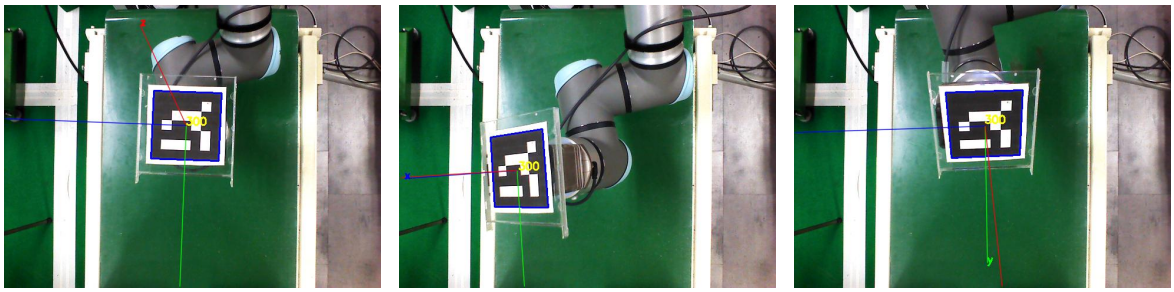


Figura 6.9 – Deteção da posição do ArUco em várias posições.

sicionar sobre o ArUco centrado com o centro geométrico do mesmo e marcou-se sobre o padrão a posição estimada. Realizaram-se testes com o ArUco posicionado verticalmente sobra a câmara, ou seja, com um deslocamento nulo do centro desta e, posteriormente a uma distância de 5 cm, 10 cm, 15 cm e 20 cm e 25 cm do ponto referente ao centro da câmara.

Resultados e discussão

O RViz permite visualizar não só as tf do próprio manipulador como, por exemplo, a *base_link*, como também a tf criada para a posição e orientação estimada da câmara (*camera_link*) e a tf associada à deteção do ArUco, como ilustrado na figura 6.10.

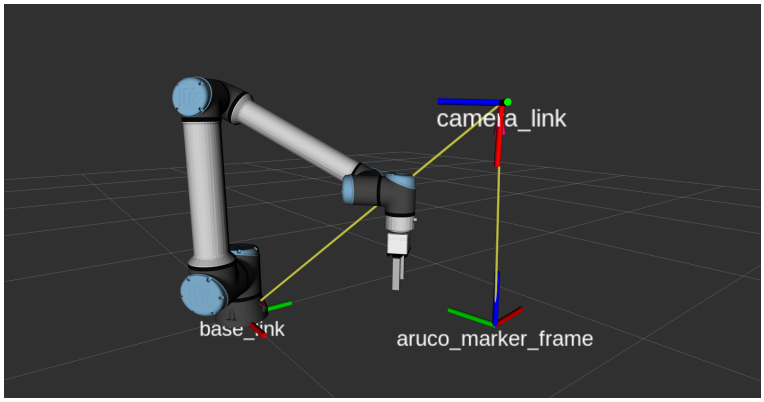


Figura 6.10 – *Frame* estimado da câmara pela calibração *Hand eye* e *frame* publicado para o ArUco.

A tabela 6.4 representa os erros obtidos na calibração para diferentes localizações do ArUco.

Distância (cm)	Erro (mm)
25	4.0
20	3.5
15	2.5
10	2.0
5	2.0
0	1.0

Tabela 6.4 – Erro da calibração em função da distância do ArUco ao centro da câmara.

Tendo por base o artigo que apresenta a calibração *Hand-eye* [57], as possíveis fontes de erro podem ser:

- Distância entre a câmara e o *ArUco*;
- Pequenas oscilações que podem originar pontos *outliers*;
- Calibração dos parâmetros intrínsecos da câmara;
- Qualidade de impressão do ArUco.

Tentou-se minimizar cada um dos pontos anteriormente referidos, no entanto, não se conseguiu obter a exatidão desejada. Assim, os resultados obtidos, ilustrados na tabela 6.4, apesar de não serem os idealmente pretendidos estão próximos dos requisitos estabelecidos, ou seja, de um erro recomendado a rondar os 2 mm. No entanto, tento em conta as condições de teste acredita-se que é um erro aceitável. Verifica-se ainda um incremento do erro em função da distância o que de certa forma é expectável devido às limitações impostas pela calibração tanto intrínseca como extrínseca.

Não foram testados outros pacotes de calibração, porém acredita-te que recorrendo a estes, o erro pode ser minimizado. Usou-se um ArUco para a calibração, no entanto, usar um ChArUco pode resultar em calibrações mais precisas, uma vez que este padrão pode não estar totalmente visível, pois este é composto por vários ArUcos, logo essa falta de visibilidade não é um problema [104], contudo não foram realizados testes com este padrão.

Além disso, acredita-se que não só reduzir a distância entre a câmara e o objeto a ser detetado, como usar uma câmara de maior resolução, pode contribuir para a obtenção de resultados mais exatos.

6.7 Manipulação de objetos em ambiente real

Nesta secção, o trabalho desenvolvido visa manipular as peças, previamente identificadas pelo subsistema de visão, em ambiente real, depois de efetuada a calibração extrínseca. Com vista à correta manipulação das peças, é necessário, antes de planear a trajetória a executar, subscrever os tópicos `object/position`, `object/orientation`, `/object/dimensions` e `/object/laser`, de modo, a obter a posição, orientação e dimensões da peça, assim como, a posição da marcação. Caso o subsistema de visão reconheça a peça como invertida, esta fica impossibilitada de se proceder à marcação a laser no lado desejado. Assim, o manipulador pode, por exemplo, colocar a peça numa zona de recolha para este efeito.

Como referido na secção 5.5, a posição da peça é calculada com base no modelo da câmara *Pinhole* e, desta forma, obtém-se as coordenadas do objeto relativamente à tf da câmara estimada (*camera_link*). Dado que os comandos da manipulação são realizados com coordenadas relativas à tf da base do robô (*base_link*), surgiu a necessidade de converter as coordenadas recorrendo à árvore tf do sistema. Assim, usando a classe *TransformListener*, mais concretamente a função *transformPose*, calculou-se as coordenadas do objeto relativamente à base do manipulador, de modo a ser possível a manipulação do mesmo.

6.7.1 Desenvolvimento de uma garra

Devido ao formato, massa e textura das peças que se pretendem manipular, a configuração de uma garra comum baseada em fricção não é uma solução fidedigna. Tipicamente, estas peças apresentam uma altura de apenas 2 mm e são constituídos por um material baseado em aço inoxidável, como já foi mencionado. Dependendo da totalidade do material constituinte das peças, estas podem apresentar uma componente ferro-magnética, no entanto, neste caso, isso não se verifica e, por esse motivo, descartou-se a possibilidade de usar uma garra baseada num eletroíman. Assim sendo, pensou-se em usar uma garra baseada em vácuo com um sistema elétrico de controlo. Uma vez que não se dispunha desse tipo de material, desenvolveu-se um protótipo de uma garra constituída por 7 ventosas de 22 mm de diâmetro como ilustrado na figura 6.11. A garra foi desenvolvida num software de modulação 3D denominado Fusion360 e, posteriormente, foi impresso numa impressora 3D usando como matéria prima o *Poliatic Acid* (PLA).

Figura 6.11 – Protótipo da garra a vácuo com 7 ventosas de 22mm de diâmetro.

A solução apresenta algumas limitações, uma vez que as dimensões das ventosas e a disposição das mesmas podem contribuir para a incorreta manipulação dos objetos. Além disso, este sistema apenas possibilita agarrar as peças sem a capacidade de as largar devidamente. O desenvolvimento desta garra tem como objetivo ser uma prova de conceito e não ser uma solução final prática, contudo, foram realizados testes de forma a validar a usabilidade deste protótipo, nomeadamente, a taxa de sucesso que a garra obtêm em agarrar as peças.

Experiências

De forma a validar a manipulação das peças usando este protótipo, fizeram-se 18 testes de manipulação com 3 das 7 peças, usando o STOMP. Entre cada teste fez-se uma rotação de cerca de 20° e, para cada posição, é calculada a sua posição, baseada no centróide da peça, e a orientação da mesma. A estimativa da orientação da peça é também fundamental, uma vez

que a garra pegará na peça tendo isso em consideração. Deste modo, a peça é sempre entregue ao marcador a laser na mesma posição e orientação, com vista à marcação identificativa.

O procedimento começa com o movimento do robô até à sua posição inicial, como ilustrado na figura 6.5a, e, posteriormente, é repetida a seguinte sequência de ações, para cada um das peças a manipular:

1. O nó ROS, dedicado à manipulação, subscreve os 4 tópicos publicados pelo nó responsável pelo reconhecimento de objetos, nomeadamente os tópicos `/object/position`, `/object/orientation`, `/object/dimensions` e `/object/laser`, onde é possível obter informações sobre os objetos a manipular, mais concretamente a sua posição, orientação e dimensão e a posição da marcação a laser.

2. Garante-se que a posição inicial de cada uma das juntas é sempre a mesma (Figura 6.12a). Depois de obtida a posição da peça através do tópico `/object/position` é planeada e executada uma trajetória, com base na indicação de uma posição global, que faz com o que o manipulador apresente uma configuração onde a garra se encontra sobre o objeto, virada para baixo e a cerca de 10 cm (Figura 6.12b). Além disso, com a subscrição do tópico `/object/orientation` é garantindo que o eixo *roll*, do *endeffector*, se encontra com a mesma orientação da peça (Figura 6.12c).

3. É planeado e executado um movimento relativo vertical de aproximação da peça, mantendo a configuração da garra (Figura 6.12d). Assim, pressiona-se as ventosas sobre a peça, de modo a agarrá-la;

4. É planeada e executada uma trajétoria com base na indicação das posições das juntas e o manipulador desloca a peça para uma posição pré-definida de forma a disponibilizá-la ao marcador a laser sempre na mesma configuração (Figura 6.12e).

5. O manipulador volta para a sua posição inicial (Figura 6.12f), e o ciclo repete-se;

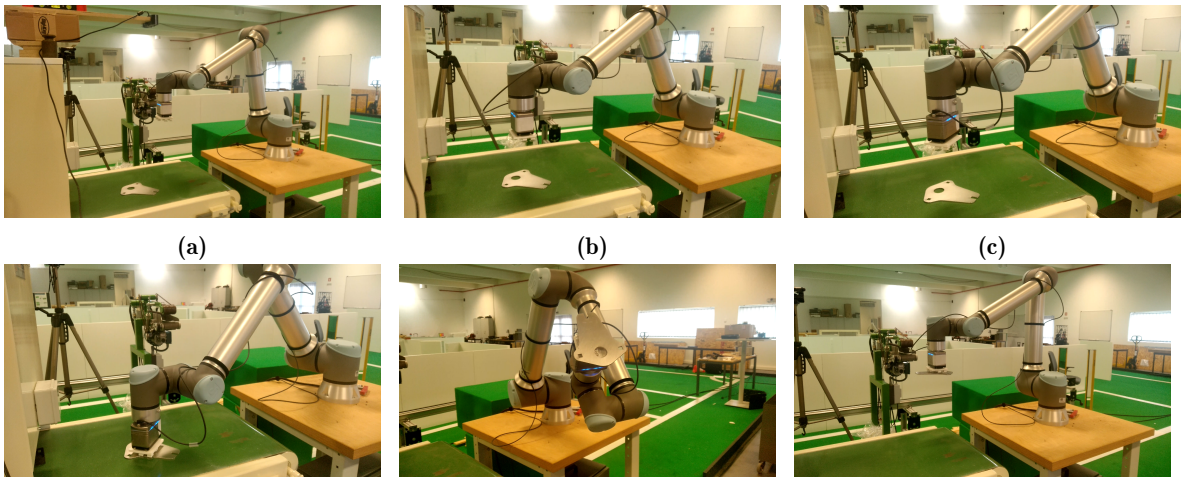
Fez-se o teste para as peças Nº4, Nº5 e Nº7, segundo a nomenclatura definida na secção 5.1. A peça Nº7 é considerada fácil dado que é pequena, leve e com poucos orifícios. A peça Nº5 é de dificuldade média dado que, apesar de ter poucos orifícios, apresenta maiores dimensões que a peça Nº7, o que faz com que o sua massa também aumente. A peça Nº4 é considerada difícil, uma vez que tem uma superfície com muitas irregularidades o que faz com que dificulte a criação de pressão do ar no interior da ventosa.

Relativamente à velocidade de execução, por questões de segurança, apenas se realizaram testes a 20% da velocidade máxima

Resultados e discussão

Segundos o resultados representados na tabela 6.5, num conjunto de 18 testes, este sistema conseguiu agarrar na peça Nº7 17 vezes, apenas falhando uma vez, e, relativamente às peças Nº4 e Nº5, falhou em ambas 3 vezes.

As causas das falhas devem-se, por um lado, à degradação das ventosas depois de consecutivos testes, o que provoca com que estas percam a capacidade de criar pressão interna. Por outro lado, ao facto de, por vezes, a garra não se posicionar com a máxima exatidão sobre a peça, devido aos erros obtidos na calibração intrínseca e extrínseca. Esta imprecisão pode provocar um ligeiro desvio da posição ideal, levando a que as ventosas encontrem um



(a) (b) (c)

(d) (e) (f)

Figura 6.12 – Procedimento da manipulação da peça.

Nº da Peça	Variação das juntas (rad)		Tempo de planeamento (s)		Tempo de execução (s)		Nº de falhas
	Média	Desvio-padrão	Média	Desvio-padrão	Média	Desvio-padrão	
4	8.38	0.52	1.52	0.33	16.67	0.75	3
5	8.00	0.11	1.56	0.36	16.27	0.27	3
7	8.75	0.23	1.59	0.26	16.20	0.25	1

Tabela 6.5 – Testes de manipulação num conjunto de 18 testes para cada peça.

orifício ou uma extremidade da peça que não permite criar a pressão interna necessária para a agarrar.

Com o desenvolvimento deste protótipo, conclui-se que o sistema a vácuo, semelhante a este sistema de ventosas, aparenta ser uma solução válida, sendo ainda necessário um sistema elétrico para o controlo do mesmo, convertendo assim um sistema passivo num sistema ativo. A dimensão das ventosas é um aspeto fundamental para o sucesso do sistema, uma vez que, dada a massa, formato e orifícios apresentados pelas diferentes peças, quanto menor for a dimensão das ventosas e maior o seu número, acredita-se que a probabilidade sucesso poderá ser maior.

Para cada trajetória criaram-se 4 planeamentos e, por isso, os resultados apresentados na tabela 6.5, recorrendo ao STOMP, são um somatório dos resultados obtidos em cada um dos planeamentos. Quanto aos tempos de planeamento obtidos, estes são ajustados tendo em conta os requisitos definidos. Definiu-se que o tempo máximo aceitável seria por volta dos 8 segundos, num ciclo total de manipulação, portanto tendo em conta os tempos de planeamento obtidos e sabendo que a velocidade de execução está apenas a 20% da velocidade máxima, considera-se que os requisitos foram cumpridos. Não se realizaram testes com maiores velocidades, não porque o sistema não o permite, mas por uma questão de segurança.

6.8 Marcação a laser em ambiente real

O objetivo desta secção é efetuar a marcação a laser em peças depois de corretamente reconhecidas junto da base de dados. A proposta inicial baseia-se num marcador a laser fixo, sendo que o robô é responsável por manipular e facultar a peça devidamente.

Uma vez identificada a posição e a orientação da peça, a garra do manipulador posiciona-se sobre a peça e toma a orientação segundo a mesma. Desta forma, é garantido que a peça é sempre disponibilizada ao marcador a laser na mesma posição e orientação. A precisão nesta configuração do sistema é garantida, não só, pela precisão da estimativa da posição e da orientação da peça, mas também pela estimativa da posição da câmara. Como se verificou em testes anteriores, a estimativa tanto da posição como da orientação apresenta erros próximos dos requisitos definidos. Porém, no caso da estimativa da posição da câmara, ou seja, a calibração extrínseca, relevou-se ter alguma imprecisão.

A proposta inicial era o laser ser fixo e o manipulador disponibilizar a peça ao marcador a laser. No entanto, com vista a validar a exatidão e precisão da marcação a laser, testou-se uma outra possibilidade da marcação a laser que passa por acoplar o laser ao manipulador. Assim, o manipulador posiciona-se sobre a peça, mais concretamente, segundo as coordenadas da marcação, ilustradas na figura 6.13, subscritas no tópico `/object/laser`. O ponto a preto ilustrado nessa figura é estimado com base nas coordenadas previamente definidas e armazenadas na base de dados e no ângulo estimado entre a peça modelo e a peça presente no tapete transportador. Este ponto representa a posição da marcação a laser.

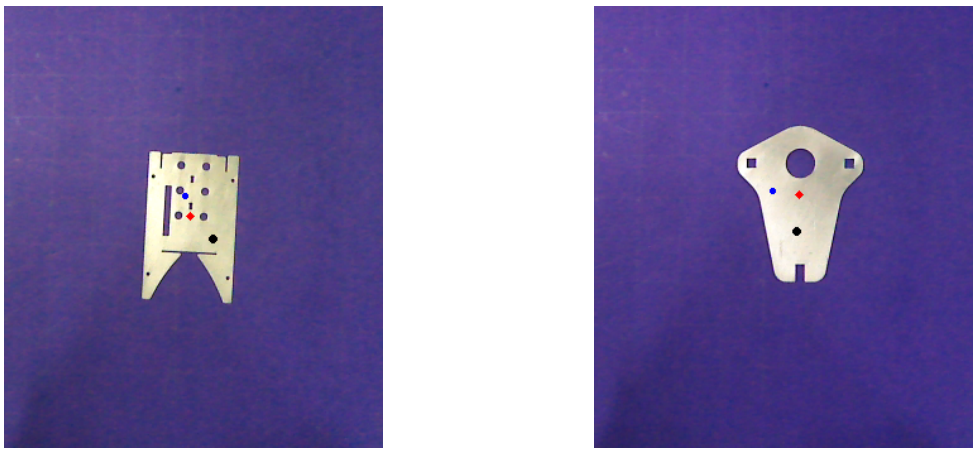


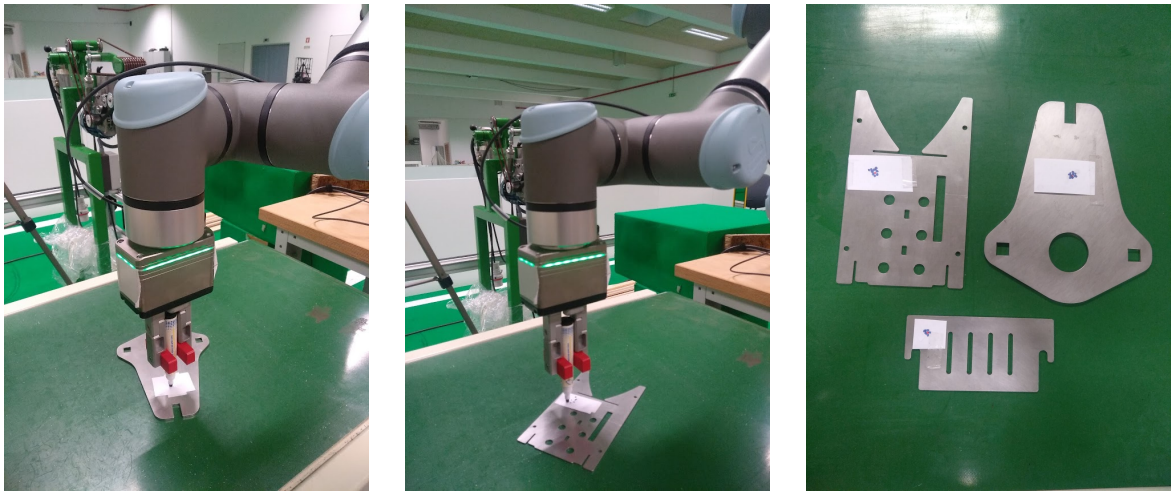
Figura 6.13 – Ilustração das coordenadas da marcação a laser representadas pelo ponto preto, onde o ponto azul é o centro da imagem e o ponto a vermelho o centróide da peça.

Experiências

Com vista a validar esta solução, considerando as coordenadas de marcação arbitrárias atribuídas às peças Nº4, Nº5 e Nº6, registou-se a marcação realizada pelo manipulador, como ilustrado nas figuras 6.14a e 6.14b. Entre cada marcação rodava-se a peça cerca de 36° até completar uma rotação completa. No total realizaram-se 10 marcações para cada uma das peças e estas foram registadas, como ilustrado na figura 6.14c.

Resultados e discussão

A figura 6.14 ilustra o processo de marcação a laser realizado com base nas coordenadas de marcação previamente definidas.



(a) (b) (c)

Figura 6.14 – Simulação do marcador a laser acoplado à garra.

Considerando um referencial arbitrário, construíram-se os gráficos da figura 6.15, onde é possível verificar a dispersão que é efetuada na marcação a laser. Os pontos a vermelho representam as marcações efetuadas pelo manipulador e o ponto a verde o que se pretendia marcar.

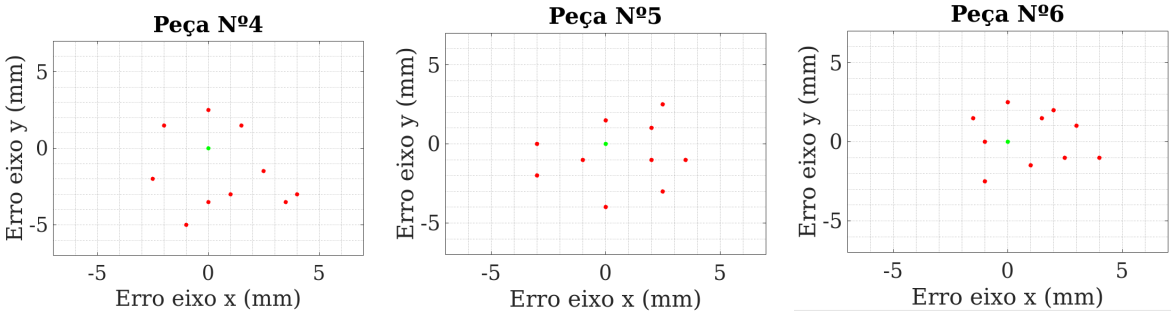


Figura 6.15 – Representação das marcações das peças com base no referencial da peça.

A tabela 6.6 apresenta o erro médio da distância, entre o ponto de marcação esperado e os pontos efetivamente marcados.

Nº da Peça	Erro Distância (mm)	
	Média	Desvio-padrão
4	3.50	1.12
5	2.91	0.98
6	2.50	0.84

Tabela 6.6 – Erro da distância na marcação a laser.

Com base nos resultados obtidos da tabela 6.6, verifica-se que a exatidão do marcador a laser tem um erro médio, nas 3 peças, entre 2.50 mm e 3.50 mm. Estes resultados encontram-se próximos dos erros máximos idealmente pretendidos a rondar os 2 mm. Além disso, tendo em consideração os desvios-padrão, a rondar no máximo os 1.12 mm, pode-se concluir que os resultados obtidos são também consideravelmente precisos e ajustados aos requisitos estabelecidos. Considerou-se que o erro obtido pode ter origem na estimativa da posição e orientação da peça, que por sua vez é agravado pela não exata calibração intrínseca e extrínseca da câmara.

Capítulo 7

Conclusão e trabalho futuro

7.1 Conclusão

Neste trabalho propôs-se um sistema, de alta cadência, dedicado à marcação laser de peças, essencialmente constituídas por aço inoxidável, de diversos formatos. Dividiu-se o sistema geral em dois subsistemas. Um dos subsistemas é responsável pelo reconhecimento das peças e o outro pela manipulação das mesmas, de forma a facultá-las ao marcador a laser. Um conjunto de requisitos foram definidos e alguns deles foram totalmente atingidos, outros encontram-se significativamente próximos dos resultados obtidos.

Relativamente ao subsistema de reconhecimento de peças, inicialmente, fez-se um estudo sobre o reconhecimento de peças e da estimativa da orientação baseada na correlação entre duas imagens. Este método, além de não ser robusto o suficiente para o reconhecimento de peças, apresentou erros significativos na estimativa da orientação, quando comparados com os erros obtidos na solução final. Em adição, os tempos obtidos são totalmente desajustados para uma aplicação de elevada cadência.

Assim, tendo em conta os resultados obtidos no método anteriormente referido, foi necessário estudar outras abordagens. A solução final apresentada, baseada em *features*, obteve resultados satisfatórios, tanto ao nível de taxa de sucesso, como ao nível do tempo computacional exigido. No decorrer desta dissertação, estes foram sempre os dois pontos fundamentais a respeitar e, por isso, tentou-se sempre encontrar um compromisso entre eles dando mais prevalência à taxa de sucesso. Realizaram-se testes não só a algoritmos de extração e descrição de *features*, como também, aos métodos de correspondência de *features*, de forma a avaliar quais os que apresentavam melhores resultados segundo os critérios definidos. O ORB, na extração e descrição, e o FLANN, na correspondência, foram os algoritmos usados.

De forma a contribuir para a redução do tempo computacional, recorreu-se a um sistema baseado em multiprocessamento, com 4 processos, e com filtragem baseada na área da peça. Além disso, após a calibração dos parâmetros intrínsecos e da aplicação do método desenvolvido, obtiveram-se graus de precisão próximos dos impostos pelos requisitos, no que diz respeito, não só, à estimativa da posição da peça, como à estimativa da sua orientação.

Todo o sistema real foi também modelado e simulado usando o simulador Gazebo, o que permitiu, de uma forma fácil e rápida, que partes do desenvolvimento se processassem em simulação. Assim, a simulação permitiu validar os dois subsistemas, com vista à aplicação real.

Quanto à manipulação em ambiente real, obteve-se um subsistema que não sendo ideal

revelou-se funcional, onde é possível manipular objetos usando o protótipo de garra desenvolvido para o efeito. Realizou-se também uma análise comparativa de vários planeadores de trajetórias disponíveis no MoveIt!, no sentido de escolher o mais adequado para o manipulador UR10e neste género de aplicação. Os testes efetuados demonstram que o STOMP é o planeador mais indicado. Por fim, no que diz respeito à marcação a laser das peças, conseguiu-se uma exatidão próxima da requisitada. Considerou-se que a falta de exatidão obtida pode ter origem no erro associado à estimativa da posição e orientação da peça, que por sua vez é agravado pela não exata calibração intrínseca e extrínseca da câmara.

7.2 Trabalho futuro

Um dos objetivos desta dissertação era fazer o reconhecimento de peças planares alojadas numa base de dados local. Assim, no futuro, o objetivo passa por fazer reconhecimento de objetos 3D, tendo em conta os diferentes desafios inerentes como, por exemplo, na questão da orientação em que não se realiza apenas num plano como no caso de peças planares. Além disso, o objetivo é implementar este sistema numa rede industrial e, por isso, a base de dados deve ser, no futuro, integrada num servidor dedicado.

Outro aspeto que não se teve em consideração foi qual a informação que se pretendia marcar na peça logo, quando se criou a base de dados, não existiu a preocupação em armazenar essa informação. No futuro, o objetivo passa por associar a cada peça uma marcação específica onde, posteriormente, se pode proceder à verificar se a marcação foi corretamente realizada.

Relativamente à estimativa da posição e orientação da peça obtiveram-se erros que apesar de ajustados aos requisitos tem ainda alguma margem de progresso. Com vista à melhoria dos resultados obtidos, discutiu-se também que os resultados obtidos podem estar dependentes de dois fatores que poderiam contribuir significativamente para a melhoria dos mesmos, nomeadamente o uso de uma câmara com uma maior resolução, ou ainda, a redução da distância entre as peças e a câmara.

Quanto à posição da marcação a laser esta foi atribuída pelo utilizador relativamente à imagem modelo. No futuro, a posição poderia ser fornecida pelo ficheiro CAD referenciada relativamente ao centróide da peça. Assim, sabendo o ângulo da peça, do ficheiro CAD, relativamente à imagem modelo, pode-se estimar a posição da marcação. A estimativa do ângulo entre a imagem modelo e a peça do ficheiro CAD, poderia basear-se no raciocínio apresentado na secção 5.6.1. Neste caso seria exequível aplicar este género de algoritmo, dado que apenas se realizará 1 vez, por cada vez que se adiciona uma peça à base de dados.

Quanto à componente de manipulação, usou-se o algoritmo de cálculo de cinemática inversa selecionado por padrão no MoveIt! e não se realizaram testes a outros algoritmos. Deste modo, no futuro, outros algoritmos podem ser testados como, por exemplo, o IKFast, de forma, a diminuir os tempos de computação associados na componente de manipulação.

Por fim, tanto a calibração intrínseca como a extrínseca da câmara, revelaram algumas limitações ao nível da exatidão e, por isso, deve-se, no futuro, testar outras metodologias como, por exemplo, outros pacotes de calibração de modo a obter uma calibração com um maior rigor, de modo a obter tanto a manipulação das peças como a marcação a laser com menor erro.

Bibliografia

[1] Department of Economic United Nations e Social Affairs. *"World Population Prospects 2019"*. https://population.un.org/wpp/Publications/Files/WPP2019_Highlights.pdf, Last access 2021-06-10. 2019.

[2] International federation of robotics. *"IFR presents World Robotics Report 2020"*. https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe, Last access 2021-05-10. 2020.

[3] Dana Harry Ballard e Christopher M. Brown. *"Computer Vision"*. 1st. Prentice Hall Professional Technical Reference, 1982. ISBN: 0131653164.

[4] D. Forsyth e J. Ponce. *"Computer Vision: A Modern Approach"*. Alan R. Apt book. Prentice Hall, 2003. ISBN: 9780131911932. URL: https://books.google.pt/books?id=a3-TQgAACAAJ.

[5] Marco F. Duarte et al. "Single-pixel imaging via compressive sampling". Em: *IEEE Signal Processing Magazine* 25.2 (2008), pp. 83–91. DOI: 10.1109/MSP.2007.914730.

[6] Adrian Kaehler e Gary Bradski. *"Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library"*. 1st. O'Reilly Media Inc, 2016. ISBN: 1491937998.

[7] MATrix LABoratory. *"What Is Camera Calibration?"* https://www.mathworks.com/help/vision/ug/camera-calibration.html, Last access 2021-09-13.

[8] Hoang Minh Nguyen et al. "3D models from the black box: investigating the current state of image-based modeling". Em: Vaclav Skala-UNION Agency, 2012.

[9] Open Source Computer Vision Library. *"OpenCV - Home"*. http://opencv.org/, Last access 2021-08-10. 2021.

[10] Raman Maini e Himanshu Aggarwal. "Study and comparison of various image edge detection techniques". Em: *International journal of image processing (IJIP)* 3.1 (2009), pp. 1–11.

[11] Paul L Rosin. "Measuring corner properties". Em: *Computer Vision and Image Understanding* 73.2 (1999), pp. 291–307.

[12] Carmelo Cassisi et al. "Similarity Measures and Dimensionality Reduction Techniques for Time Series Data Mining". Em: set. de 2012. ISBN: 978-953-51-0748-4. DOI: 10.5772/49941.

[13] Kwon Lee et al. "Fast object detection based on color histograms and local binary patterns". Em: *TENCON 2012 IEEE Region 10 Conference*. IEEE. 2012, pp. 1–4.

[14] Xinwei Qi e Ligang Miao. "A Template Matching Method for Multi-Scale and Rotated Images Using Ring Projection Vector Conversion". Em: *IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*. 2018, pp. 45–49. DOI: 10.1109/ICIVC.2018.8492726.

[15] Simon Korman et al. "FasT-Match: Fast Affine Template Matching". Em: *IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2331–2338. DOI: 10.1109/CVPR.2013.302.

[16] Shiliang Zhang et al. "USB: Ultrashort binary descriptor for fast visual matching and retrieval". Em: *IEEE Transactions on Image Processing* 23.8 (2014), pp. 3671–3683.

[17] Patricio Loncomilla. "Object Recognition using Local Invariant Features for Robotic Applications: A Survey". Em: *Pattern Recognition* 60 (mai. de 2016). DOI: 10.1016/j.patcog.2016.05.021.

[18] Shaharyar Ahmed Khan Tareen e Zahra Saleem. "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk". Em: *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*. IEEE. 2018, pp. 1–10.

[19] David G Lowe. "Distinctive image features from scale-invariant keypoints". Em: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[20] Mahmoud Hassaballah, Aly Amin Abdelmgeid e Hammam A Alshazly. "Image features detection, description and matching". Em: *Image Feature Detectors and Descriptors*. Springer, 2016, pp. 11–45.

[21] Herbert Bay et al. "Speeded-Up Robust Features (SURF)". Em: *Computer Vision and Image Understanding* 110.3 (2008). Similarity Matching in Computer Vision and Multimedia, pp. 346–359. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2007.09.014. URL: https://www.sciencedirect.com/science/article/pii/S1077314207001555.

[22] Piotr Porwik e Agnieszka Lisowska. "The Haar-wavelet transform in digital image processing: its status and achievements". Em: *Machine graphics and vision* 13.1/2 (2004), pp. 79–98.

[23] Edward Rosten e Tom Drummond. "Machine learning for high-speed corner detection". Em: *"In European Conference on Computer Vision"*. 2006, pp. 430–443.

[24] Mustafa Ozuysal et al. "Fast Keypoint Recognition Using Random Ferns". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.3 (2010), pp. 448–461. DOI: 10.1109/TPAMI.2009.23.

[25] Michael Calonder et al. "Brief: Binary robust independent elementary features". Em: *European conference on computer vision*. Springer. 2010, pp. 778–792.

[26] Stefan Leutenegger, Margarita Chli e Roland Y Siegwart. "BRISK: Binary robust invariant scalable keypoints". Em: *2011 International conference on computer vision*. Ieee. 2011, pp. 2548–2555.

[27] Ethan Rublee and Vicent Rabuad and Kurt Konolige and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". Em: (2011). Ed. por Dimitris N. Metaxas et al., pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544. URL: https://doi.org/10.1109/ICCV.2011.6126544.

[28] Q. Cui, H. Liu e C. Wang. "Robust Multi-scale ORB Algorithm in Real-Time Monocular Visual Odometry". Em: (2017), pp. 244–249. DOI: 10.1109/ACPR.2017.101.

[29] Pablo Fernández Alcantarilla, Adrien Bartoli e Andrew J. Davison. "KAZE Features". Em: *Computer Vision – ECCV 2012*. Ed. por Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 214–227. ISBN: 978-3-642-33783-3.

[30] P. Perona e J. Malik. "Scale-space and edge detection using anisotropic diffusion". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.7 (1990), pp. 629–639. DOI: 10.1109/34.56205.

[31] Pablo F Alcantarilla e T Solutions. "Fast explicit diffusion for accelerated features in nonlinear scale spaces". Em: *IEEE Trans. Patt. Anal. Mach. Intell* 34.7 (2011), pp. 1281–1298.

[32] X. Yang e K. Cheng. "LDB: An ultra-fast feature for scalable Augmented Reality on mobile devices". Em: *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)* (2012), pp. 49–57.

[33] J. Howse e J. Minichino. *"Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning, 3rd Edition"*. Packt Publishing, 2020. ISBN: 9781789530643. URL: https://books.google.pt/books?id=ef%5C_RDwAAQBAJ.

[34] Izrail Solomonovich radshteyn et al. *Table of integrals, series, and products; 8th ed.* Amsterdam: Academic Press, set. de 2014. DOI: 0123849330.

[35] Open Source Computer Vision Library. *"Open Source Computer Vision Library - Feature Matching"*. http://opencv.org/, Last access 2021-08-10.

[36] Marius Muja e David G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". Em: (2009). Ed. por Alpesh Ranchordas e Helder Araújo, pp. 331–340.

[37] Chanop Silpa-Anan e Richard Hartley. "Optimised KD-trees for fast image descriptor matching". Em: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587638.

[38] Marius Muja e David G. Lowe. "Scalable Nearest Neighbor Algorithms for High Dimensional Data". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (2014), pp. 2227–2240. DOI: 10.1109/TPAMI.2014.2321376.

[39] M. Aly et al. "Scaling object recognition: Benchmark of current state of the art techniques". Em: (2009), pp. 2117–2124. DOI: 10.1109/ICCVW.2009.5457542.

[40] Piotr Indyk e Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". Em: STOC '98 (1998), pp. 604–613. DOI: 10.1145/276698.276876. URL: https://doi.org/10.1145/276698.276876.

[41] S. Se, D. G. Lowe e J. J. Little. "Vision-based global localization and mapping for mobile robots". Em: *IEEE Transactions on Robotics* 21.3 (2005), pp. 364–375. DOI: 10.1109/TRO.2004.839228.

[42] Martin A. Fischler e Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". Em: *Commun. ACM* 24.6 (jun. de 1981), pp. 381–395. ISSN: 0001-0782. URL: https://doi.org/10.1145/358669.358692.

[43] Konstantinos G. Derpanis. "Overview of the RANSAC Algorithm". Em: *Image Rochester NY* 4.1 (2010), pp. 2–3.

[44] Opencv. *"Camera Calibration and 3D Reconstruction"*. https://docs.opencv.org/4.5.2/d9/d0c/group__calib3d.html#ga27865b1d26bac9ce91efaee83e94d4dd, Last access 2021-07-04. 2021.

[45] Manolis IA Lourakis et al. "A brief description of the Levenberg-Marquardt algorithm implemented by levmar". Em: *Foundation of Research and Technology* 4.1 (2005), pp. 1–6.

[46] Kimon P Valavanis e George N Saridis. *Intelligent robotic systems: theory, design and applications*. Vol. 182. Springer Science & Business Media, 2012.

[47] Guoqiang Hu, Wee Peng Tay e Yonggang Wen. "Cloud robotics: architecture, challenges and applications". Em: *IEEE Network* 26.3 (2012), pp. 21–28. DOI: 10.1109/MNET.2012.6201212.

[48] Carlos Henrique Gonçalves Campbell et al. *"Desenvolvimento de um robô manipulador industrial"*. Available at http://docplayer.com.br/18658673-Desenvolvimento-de-um-robo-manipulador-industrial.html, Last access 2021-07-24. 2008.

[49] Valdemir Carrara. *"Apostila de Robótica."* Available at http://gazebosim.org/tutorials?tut=quick_start, Last access 2021-09-8. 2004.

[50] Airton Almeida de Moraes. *"Robótica"*. Available at http://www.adororobotica.com/RBSENAI.pdf, Last access 2021-07-24. 2003.

[51] Universal Robots. *"Universal Robot: Technical details"*. https://www.universal-robots.com/, Last access 2021-06-16. 2014.

[52] Festo SE Co. KG. *"The Festo Group"*. Available at https://www.festo.com, Last access 2021-07-24.

[53] FANUC Iberia S.A.U. *"Industrial Robots - Top performers for smarter automation"*. Available at https://www.fanuc.eu/pt/pt, Last access 2021-07-24.

[54] António Mendes Lopes. *"robótica Industrial - Modelação Cinemática e Dinâmica de Manipuladores de Estrutura em Série."* Universidade do Porto. 2002.

[55] Simon Bøgh et al. "Autonomous industrial mobile manipulation (AIMM): from research to industry". Em: *Proceedings of the 42nd International Symposium on Robotics*. VDE Verlag GMBH. 2011.

[56] P. Robotics. *"O que são garras robóticas?"* Available at https://www.universal-robots.com/br/blog/o-que-s~ao-garras-rob\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\let\begingroup\endgroup\relax\let\ignorespaces\relax\accent19o\egroup\spacefactor\accent@spacefactorticas/, Last access 2021-08-05. 2020.

[57] R.Y. Tsai e R.K. Lenz. "A new technique for fully autonomous and efficient 3D robotics hand/eye calibration". Em: *IEEE Transactions on Robotics and Automation* 5.3 (1989), pp. 345–358. DOI: 10.1109/70.34770.

[58] OpenCV. *"Detection of ArUco Markers"*. Available at https://docs.opencv.org/4.5.2/d5/dae/tutorial_aruco_detection.html, Last access 2021-08-16. 2021.

[59] Oleg Kalachev. *"ArUco markers generator!"* Available at https://chev.me/arucogen/, Last access 2021-08-10. 2021.

[60] Eurico Pedrosa et al. "A General Approach to Hand–Eye Calibration Through the Optimization of Atomic Transformations". Em: *IEEE Transactions on Robotics* (2021), pp. 1–15. DOI: 10.1109/TRO.2021.3062306.

[61] Eduardo Lobo Lustosa Cabral. *"Cinemática Direta"*. Robôs Industriais, 2019.

[62] Jianyu Wang et al. "Inverse Kinematics of 6-DOF Robot Manipulator via Analytic Solution with Conformal Geometric Algebra". Em: *E International Conference on Robotics and Biomimetics (ROBIO)*. 2018, pp. 2508–2513. DOI: 10.1109/ROBIO.2018.8665317.

[63] Ioan A. Sucan, Mark Moll e Lydia E. Kavraki. "The Open Motion Planning Library". Em: *IEEE Robotics Automation Magazine* 19.4 (2012), pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[64] Matt Zucker et al. "Chomp: Covariant hamiltonian optimization for motion planning". Em: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1164–1193.

[65] Mrinal Kalakrishnan et al. "STOMP: Stochastic trajectory optimization for motion planning". Em: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574.

[66] Maxim Likhachev. *"Search-based Planning with Motion Primitives"*. Available at https://wiki.ros.org/Events/CoTeSys-ROS-School?action=AttachFile&do=get&target=robschooltutorial_oct10.pdf, Last access 2021-08-15.

[67] Jonathan Meijer, Qujiang Lei e Martijn Wisse. "Performance study of single-query motion planning for grasp execution using various manipulators". Em: *2017 18th International Conference on Advanced Robotics (ICAR)*. 2017, pp. 450–457. DOI: 10.1109/ICAR.2017.8023648.

[68] K. Lab. *"Available Planners"*. Available at https://ompl.kavrakilab.org/planners.html, Last access 2021-10-16.

[69] MoveIt Organization. *"STOMP Planner"*. Available at https://github.com/ros-planning/moveit_tutorials/blob/melodic-devel/doc/stomp_planner/stomp_planner_tutorial.rst, Last access 2021-10-12. 2019.

[70] Ricardo Daniel Correia Pinto. "Remodelação, simulação e controlo de um braço robótico". Tese de mestrado. Universidade de Aveiro, 2021.

[71] Andrea Tudico et al. "Improving and benchmarking motion planning for a mobile manipulator operating in unstructured environments". Em: *EPIA Conference on Artificial Intelligence*. Springer. 2017, pp. 498–509.

[72] Sandro Augusto Magalhães et al. "Path Planning Algorithms Benchmarking for Grapevines Pruning and Monitoring". Em: *EPIA Conference on Artificial Intelligence*. Springer. 2019, pp. 295–306.

[73] Universal Robots. *"UR10e technical details."* Available at https://www.universal-robots.com/media/1802458/ur10e-tech-specs-eng.pdf, Last access 2021-08-05.

[74] Morgan Quigley et al. "ROS: an open-source Robot Operating System". Em: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[75] O. S. R. Foundation. *"Documentation"*. Available at http://wiki.ros.org/, Last access 2021-07-04. 2020.

[76] Noel Martignoni. *"Schéma de fonctionnement de ROS"*. Available at https://commons.wikimedia.org/wiki/File:ROS-master-node-topic.png, Last access 2021-08-10. 2016.

[77] O. S. R. Foundation. *"Is ROS For Me?"* Available at https://www.ros.org/is-ros-for-me/, Last access 2021-08-10. 2018.

[78] P. Robotics. *"Concepts - moveit."* Available at https://moveit.ros.org/documentation/concepts/, Last access 2021-07-26.

[79] O. S. R. Foundation. *"rqt"*. Available at http://wiki.ros.org/rqt, Last access 2021-08-02. 2016.

[80] O. S. R. Foundation. *"Urdf"*. Available at http://wiki.ros.org/urdf, Last access 2021-08-10. 2019.

[81] O. S. R. Foundation. *Schéma de fonctionnement de ROS*. Available at https://commons.wikimedia.org/wiki/File:ROS-master-node-topic.png, Last access 2021-09-10.

[82] O. S. R. Foundation. *XML Robot Description Format (URDF)*. Available at http://wiki.ros.org/urdf/XML/model. 2012.

[83] O. S. R. Foundation. *XML Robot Description Format (URDF)*. Available at http://wiki.ros.org/urdf/XML/joint, Last access 2021-09-10.

[84] Tully Foote. "tf: The transform library". Em: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. IEEE. Abr. de 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373.

[85] James Bowman Patrick Mihelich. *"Vision_opencv"*. Available at http://wiki.ros.org/vision_opencv?distro=noetic/, Last access 2021-08-10. 2020.

[86] O. S. R. Foundation. *"Ros_control"*. Available at http://wiki.ros.org/ros_control, Last access 2021-09-10. 2020.

[87] Nathan Koenig e Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". Em: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[88] Lenka Pitonakova et al. "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators". Em: *Annual Conference Towards Autonomous Robotic Systems*. Springer. 2018, pp. 357–368.

[89] Zandra B. Rivera, Marco C. De Simone e Domenico Guida. "Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments". Em: *Machines* 7.2 (2019). ISSN: 2075-1702. DOI: 10.3390/machines7020042. URL: https://www.mdpi.com/2075-1702/7/2/42.

[90] Open Source Robotics Foundation. *"Gazebo - Robot simulation made easy."* Available at http://gazebosim.org/, Last access 2021-08-10. 2019.

[91] Open Source Robotics Foundation. *"Run Gazebo"*. Available at http://gazebosim.org/tutorials?tut=quick_start, Last access 2021-08-12.

[92] Wei Qian et al. "*Manipulation task simulation using ROS and Gazebo*". Em: *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. IEEE. 2014, pp. 2594–2598.

[93] Patrick Beeson e Barrett Ames. "TRAC-IK: An open-source library for improved solving of generic inverse kinematics". Em: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2015, pp. 928–935. DOI: 10.1109/HUMANOIDS.2015.7363472.

[94] Anshul Kanakia May. "Inverse Kinematics using ikfast on a 7 DOF Robotic Arm". Tese de mestrado. University of Colorado, 2012.

[95] Chi-Ju Wu. *"astra_camera "*. Available at http://wiki.ros.org/astra_camera, Last access 2021-10-14. 2019.

[96] Z. Zhang. ""A flexible new technique for camera calibration"". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[97] Python Software Foundation. *"Initialization, Finalization, and Threads"*. https://docs.python.org/3/c-api/init.html, Last access 2021-10-20. 2021.

[98] Python Software Foundation. *"multiprocessing — Process-based parallelism "*. Available at https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing, Last access 2021-10-22. 2021.

[99] Intel Corporation. *"O que é Hyper-Threading?"* Available at https://www.intel.com.br/content/www/br/pt/gaming/resources/hyper-threading.html, Last access 2021-09-22.

[100] Intelligent Robotics e Systems (IRIS) Lab. *"IRIS Universal Robot 10 e-Series"*. https://github.com/iris-ua/iris_ur10e, Last access 2021-10-10. 2020.

[101] Intelligent Robotics e Systems (IRIS) Lab. *"Simple Arm Manipulation Interface"*. https://github.com/iris-ua/iris_sami, Last access 2021-09-10. 2020.

[102] Jennifer Buehler. *"Gazebo Grasp Fix Plugin"*. Available at https://github.com/JenniferBuehler/gazebo-pkgs, Last access 2021-06-10. 2021.

[103] Interdisciplinary Research Laboratory at Computer Aided Medical Procedures. *"Easy_handeye: automated, hardware-independent Hand-Eye Calibration"*. https://github.com/IFL-CAMP/easy_handeye, Last access 2021-09-10. 2015.

[104] Petr Oščádal et al. "Improved pose estimation of aruco tags using a novel 3d placement strategy". Em: *Sensors* 20.17 (2020), p. 4825.