FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sensorização 3D e Controlo de Manipuladores Industriais através de uma plataforma de baixo custo

Ana Catarina Bastos Simões

VERSÃO FINAL

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Hélio Mendes de Sousa Mendonça (Professor Doutor)

Coorientador: Pedro Luís Cerqueira Gomes da Costa (Professor Doutor)

13 de Julho de 2017

Resumo

O avanço tecnológico tem permitido o desenvolvimento de soluções robustas no âmbito do setor industrial, com especial destaque para a área da robótica, a qual se tem vindo afirmar cada vez mais. Os sistemas robóticos atualmente implementados na indústria, recorrem a computadores industriais como ferramenta de processamento central. A fiabilidade tipicamente associada a este tipo de computadores, é uma vantagem que permite dar resposta ao tipo de requisitos impostos por processos de manufatura. Estes requisitos incluem, por exemplo, repetibilidade, precisão e segurança na execução de tarefas, características estas que fazem dos denominados PCs industriais equipamentos de custo elevado. A constante procura pela minimização de custos tem potenciado o estudo do uso de soluções mais baratas no âmbito industrial.

Caracterizados por possuir funcionalidades semelhantes às de um computador convencional, microcomputadores como o Raspberry Pi pretendem revolucionar os sistemas robóticos industriais atuais, tornando-os mais baratos, compactos e flexíveis, garantido a mesma fiabilidade de um computador industrial.

Neste contexto, a presente dissertação tem como objetivo o estudo da viabilidade do uso da versão mais recente do Raspberry Pi, como unidade de processamento central de um sistema robótico baseado em visão computacional. Este estudo passa pelo desenvolvimento de uma solução automatizada para manipulação de objetos, isto é, realização de tarefas *pick-and-place* por parte de um braço robótico.

A solução apresentada baseia-se no uso do sensor Asus Xtion Pro Live como sistema de sensorização, do qual se obtém informação 3D acerca do cenário em causa. Estes dados são tratados no formato de nuvem de pontos, pelo que o seu processamento culmina no reconhecimento de um objeto específico, um cilindro. As fases que contemplam este ciclo incluem etapas de filtragem, segmentação, entre outras, que em conjunto tornam possível o cálculo das coordenadas do objeto. Posto isto, dá-se a atuação do manipulador Motoman, de modo a que este se desloque até ao objeto, transportando-o até um ponto conhecido usando, para o efeito, uma garra, concretizando-se assim a tarefa *pick-and-place* pretendida.

Abstract

With the advances in technology the development of robust solutions in the scope of the industrial sector have been allowed, with special emphasis on the robotics area, which has been increasingly affirming its presence. The robotic systems currently implemented in industry resort to industrial computers as the central processing tool. The reliability tipically associated with this type of computers is an advantage that allow the meeting of the requirements imposed by the manufacturing processes. These requeriments include, for instance, repeatability, precision, and safety in the execution of tasks, which gives the so-called industrial PCs equipments an associated high cost. The constant search in decreasing this costs has fostered the research in using cheaper solutions in the industrial scope.

Characterised by having features similar to those of a conventional computer, microcomputers like the Raspberry Pi intend to revolutionise today's industrial robotic systems, becoming cheaper, compact and flexible, and ensuring the same reliability of an industrial computer.

In this context, this dissertation has as the main goal the study of the viability of using the latest version of Raspberry Pi as the central processing unit of a robotic system based on computer vision. This study involves the development of an automated solution for manipulating objects, i.e. performing pick-and-place tasks by a robotic arm.

The proposed solution is based on the use of the Asus Xtion Pro Live sensor as the sensing system, from which 3D information about the scenery in question is obtained. These data are treated in form of point cloud, wherefore their processing culminates in the recognition of a specific object, a cylinder. This cycle complements several phases including filtering steps, segmentation, among others, that when combined allow to calculate the coordinates of the object. Given this, the Motoman manipulator actuates, moving torwards the object, transporting it to a known point using a claw, thus performing the desired pick-and-place task.

Agradecimentos

Agradeço ao meu orientador, o Professor Doutor Hélio Mendonça, e coorientador, o Professor Doutor Pedro Costa, por todo o conhecimento partilhado e apoio prestado no decorrer desta dissertação.

Também aos membros do INESC TEC, nomeadamente o Eng. Carlos Costa, pela enorme boa-vontade e gosto em partilhar o seu vasto conhecimento, o Doutor Héber Sobreira, pela amizade e atenção, o Doutor Luís Rocha, pela ajuda prestada e também o Professor Doutor António Paulo Moreira pelas oportunidades que me proporcionou.

Aos colegas com os quais partilhei o local de trabalho, o Ivo Sousa, a Cláudia Rocha e o Francisco Ferreira pelo companheirismo, boa disposição e amizade.

À Ana Sofia, à Liliana Antão, à Marta Costa, ao Pedro Alves e à Rita Rodrigues por terem tornado esta passagem pela FEUP mais feliz.

À Joana Ferreira, à Patrícia Sousa, ao Ricardo Silva e à Susana Pereira pela amizade, preocupação, lealdade e por serem verdadeiros pilares na minha vida.

Ao Valter, pela paciência e apoio nos bons e maus momentos do meu percurso académico.

Aos meus meninos, lobitos, e restantes escuteiros do agrupamento do 592 de São Pedro de Castelões, por me fazerem sentir realizada e me ensinarem tanto a cada atividade.

Este percurso não teria sido possível sem o apoio incondicional da minha família. Por isso, agradeço aos meus pais, Emília e Joaquim, pela oportunidade que me deram em estudar, pelo enorme sacrifício que fizeram e por estarem do meu lado nas decisões mais importantes.

Ao meu irmão Pedro, por suscitar em mim a vontade de aprender mais.

Ao meu padrinho e ao meu avô que sempre acreditaram em mim e, onde quer que estejam, tenho a certeza que gostariam de ver o resultado deste trabalho.

Ana Catarina Simões

"The best workers, like the happiest livers, look upon their work as a kind of game: the harder they play the more enjoyable it becomes"

Robert Baden-Powell

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ALU	Arithmetic Logic Unit
ARM	Acorn/Advanced RISC Machine
ARMHF	Advanced RISC Machine Hardware Floating
CISC	Complex Instruction Set Computing
CPU	Central Processing Unit
EABI	Embedded Aplication Binary Interface
FTP	File Transfer Protocol
FPS	Frames Per Second
FPU	Floating Point Unit
GB	Giga Byte
GHz	Giga Hertz
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
HMI	Human-Machine Interface
IIoT	Industrial Internet of Things
IoT	Internet of Things
ISA	Instruction Set Architecture
LCD	Liquid Cristal Display
NARF	Normal Aligned Radial Feature
OpenNI	Open Natural Interaction
PC	Personal Computer
PCB	Printed Circuit Board
PCD	Point Cloud Data
PCL	Point Cloud Library
PLC	Programmable Logic Controllers
RAM	Random Acess Memory
RANSAC	Random Sample Consensous
RISC	Reduced Instruction Set Computing
RGB-D	Red Green Blue Depth
SD	Secure Digital
SIFT	Scale-Invariant Feature Transform
SMID	Single-Instruction, Multiple Data
SoC	System-on-a-Chip
SSH	Secure Shell
SURF	Speeded Up Robust Features
ToF	Time-of-Flight
USB	Universal Serial Bus

Capítulo 1

Introdução

O presente capítulo tem como objetivo introduzir a dissertação "Sensorização 3D e Controlo de Manipuladores Industriais através de uma plataforma de baixo custo". Começa-se por apresentar uma contextualização do tema na secção 1.1, seguidamente é apresentada a motivação inerente ao desenvolvimento desta tese, secção 1.2. Por fim, enumeram-se os principais objetivos na secção 1.3.

1.1 Contexto

A procura pela automatização de processos de produção, nas mais variadas áreas do ramo industrial, surge com o objetivo de aumentar a eficiência e produtividade. Neste âmbito, insere-se a utilização de equipamentos pré-programados para realização de tarefas que, usualmente, são desempenhadas por operadores humanos. Paralelamente a esta realidade, destaca-se o avanço da tecnologia, mais propriamente da robótica industrial que, ao longo dos tempos, tem vindo a responder a esta necessidade. Deste modo, surgem os manipuladores robóticos que, quando aliados a um sistema de visão computacional, revelam-se uma mais-valia em qualquer ambiente industrial para aplicações que incluem controlo de qualidade, transporte de objetos, operações de *pick-and-place*, montagem, inspeção de peças, e até pintura ou soldagem [14, 15].

No âmbito da aquisição de informação relativa ao ambiente a manipular, estes sistemas de visão recorrem, normalmente, a câmaras e/ou sensores alocados numa zona estratégica ou fixadas diretamente no braço robótico. Finalizada a fase aquisição, segue-se a interpretação dos dados recolhidos para posterior atuação do robô. Em contexto industrial, todo este ciclo que compreende a recolha, processamento e manipulação é assegurado por computadores industriais. Estes são responsáveis por fazer o intercâmbio entre o sistema de visão e o de manipulação, garantindo os tempos de resposta estipulados por norma em chão de fábrica. Para além disto, aos denominados PCs industriais é exigido que confiram robustez, repetibilidade, precisão e segurança na execução de tarefas. Salienta-se também a necessidade de serem capazes de operar em diferentes e inesperadas condições que incluem, por exemplo, níveis de vibração e temperatura elevados. Estas e

outras caraterísticas fazem deles equipamentos de custo elevado, o que tem vindo a potenciar o estudo do uso de computadores mais baratos no meio industrial.

As soluções *low-cost* atualmente em estudo incluem os computadores de placa única, que se destacam pelas suas especificações, frequentemente comparadas às de computadores convencionais. Também o facto de se tratarem de aparelhos compactos, impulsiona a sua escolha para sistemas nos quais o espaço é reduzido.

Dada a popularidade dos microcomputadores, o número de opções disponíveis no mercado tem vindo a aumentar, surgindo assim sistemas cada vez mais competitivos, como é o caso do Raspberry Pi. Concretamente, a última versão deste dispositivo apresenta caraterísticas bastante promissoras que incentivam o seu uso em ambiente industrial. É neste contexto que se insere o tema desta dissertação.

1.2 Motivação

A constante evolução e inovação do ramo industrial de processos de manufatura tem em conta fatores competitivos como o custo e qualidade, sendo que esta dissertação pretende ir ao encontro deste padrão.

Com efeito, a utilização de manipuladores industriais requer cada vez mais o uso de células inteligentes, flexíveis e capazes de se adaptarem a diferentes e imprevistas condições de operação. Assim, o problema de identificação e determinação da posição e orientação dos objetos, bem como o seu reconhecimento, em diferentes ambientes e perspetivas, é de elevada importância. Este requer a existência de um sistema suficientemente sofisticado, capaz de responder rápida e eficazmente, com o menor erro possível. Os sistemas tradicionalmente implementados baseiam-se no uso de uma câmara RGB, um sensor laser e um computador, para o mapeamento 3D do ambiente, deteção da peça e planeamento de trajetória do manipulador até à mesma. Esta abordagem, para além do custo elevado, tem a si associado um elevado peso computacional, devido à necessidade de modelos matemáticos complexos para o cálculo da profundidade de cada *pixel*. Um sensor RGB-D incorpora num só dispositivo a câmara e sensor, fornecendo o valor da profundidade de forma automática.

Por outro lado, com o avanço da tecnologia surgiram os computadores de placa única, cuja utilização se afirma com maior destaque na área educativa e de investigação. Para além deste facto, o conceito IoT (*Internet of Things*) veio dar uma maior dimensão a estes dispositivos, que começam agora a suscitar curiosidade por parte dos empresários. Este interesse é justificado pelas especificações que estas ferramentas apresentam, face a outras soluções menos acessíveis em termos de preço.

Assim, a metodologia proposta nesta dissertação sugere a implementação e avaliação de um sistema suscetível de ser usado em ambientes industriais, pelo que a sua motivação assenta na premissa da redução de custos associados às tecnologias usadas, sem comprometer a qualidade de operação do sistema. Para o efeito, usar-se-á a placa Raspberry Pi 3 Modelo B, em detrimento de um computador, como unidade central de processamento. Para além da excelente relação

qualidade-preço, o modelo mais recente apresenta uma unidade GPU, que permite um incremento considerável na velocidade de processamento [5].

Relativamente à ferramenta de visão, selecionou-se o sensor Asus Xtion Pro Live, uma solução de baixo custo que, para além de possuir a tradicional câmara RGB, tem incorporado um sensor de profundidade capaz de fornecer a medida da profundidade de cada *pixel* da imagem proveniente da câmara.

Em suma, é de realçar o valor que esta solução acrescenta à indústria de manufatura, principalmente pelo baixo custo que tem associado, mas também pelo compromisso de rapidez, segurança e precisão na realização de tarefas repetitivas, comparativamente à sua execução por operadores humanos.

1.3 Objetivos

Esta dissertação tem como principal objetivo o estudo da viabilidade de um sistema de sensorização 3D baseado no acoplamento entre um sensor RGB-D, *Red*, *Green*, *Blue* e *Depth* , mais propriamente, Asus Xtion Pro Live e o microcomputador Raspberry Pi 3 Modelo B [5], para controlo de um manipulador industrial da Motoman.

Pretende-se tirar proveito das potencialidades de um sensor de baixo custo que, para além de possuir a componente RGB, incorpora um sensor de profundidade que permite a determinação da distância a objetos num determinado cenário e respeitando os intervalos de deteção. Estas funcionalidades do equipamento da Asus, serão aliadas às vantagens da placa Raspberry Pi 3, que se destaca pelo baixo custo, face à capacidade de processamento frequentemente equiparada à de um computador. A sua última versão tem vindo a despertar um interesse maior por parte da comunidade científica, devido à garantia de melhor desempenho, para aplicações de tempo real, quando comparada a modelos anteriores.

A simbiose entre estas tecnologias tem como intuito o reconhecimento de peças, mais propriamente a sua posição e orientação. Os dados recolhidos pelo sensor serão convertidos para o formato de nuvem de pontos que, por sua vez, é processada de modo a detetar-se o objeto pretendido. Posto isto, surge a necessidade de informar o manipulador acerca das coordenadas da peça, para que este se desloque até à mesma, transportando-a para outra posição conhecida. Os manipuladores industriais podem realizar diferentes tipos de tarefas, pelo que o processo descrito atrás pretende aproximar-se de uma aplicação industrial que envolve operações *pick-and-place* num objeto cilíndrico.

Existem diferentes sistemas de visão 3D com potencial aplicação nesta área, desde sistemas com um custo mais reduzido baseados em microcomputadores como o Raspberry Pi, até dispositivos industriais mais sofisticados e, por isso, mais dispendiosos.

Em suma, esta dissertação centra-se no estudo das potencialidades e limitações do Raspberry Pi 3 Modelo B, enquanto unidade de processamento central, para o suporte de processos tipicamente industriais. Assim, pretende-se mostrar que mesmo tarefas complexas podem operar a partir de um computador de placa única. Por isso, apresentar-se-á a viabilidade do uso desta ferramenta

de baixo custo, em detrimento de um computador industrial, que atualmente assegura o controlo de tarefas com precisão e eficácia, no entanto tem associado um preço elevado.

1.4 Estrutura do Documento

O presente documento apresenta o trabalho desenvolvido no âmbito da dissertação intitulada "Sensorização 3D e Controlo de Manipuladores Industriais através de uma plataforma de baixo custo", sendo que se divide em sete capítulos.

Primeiramente, o capítulo de **Introdução**, em 1, que começa por contextualizar a dissertação, apresentando também as motivações e objetivos inerentes ao seu desenvolvimento. Segue-se um capítulo de **Revisão Bibliográfica**, em 2, no qual são abrangidas todas as áreas que este trabalho compreende, desde sistemas de visão, processamento, até a atuação de manipuladores. Também se apresentam um conjunto de aplicações que usam o microcomputador selecionado nesta dissertação. O capítulo 3 apresenta a **Arquitetura Funcional** do sistema em causa, sendo listada a seleção dos componentes de *hardware*, bem como de *software*, acompanhada de uma justificação. O capítulo seguinte, em 4, apresenta os procedimentos realizados de forma a realizar o ***Setup* do Microcomputador**.

Posteriormente, em 5, aborda-se a implementação da aplicação desenvolvida, no capítulo de ***Software* de Aplicação**, com respetiva ilustração de resultados. No sentido de dar resposta ao objetivo central desta dissertação, surge o capítulo **Testes e Resultados**, 6 que apresenta o comportamento do Raspberry Pi para diferentes cenários. Por último, fecha-se o documento com uma conclusão e apresentação de possíveis melhorias, no capítulo **Conclusão e Trabalho Futuro**, presente em 7.

Deste documento constam ainda dois anexos, mais propriamente, o anexo A no qual são apresentados **Fundamentos sobre Arquitetura de Computadores**, e o anexo B onde se encontra o ***Guideline* de Instalações** apurado da fase de *setup* do Raspberry Pi.

Capítulo 2

Revisão Bibliográfica

O presente capítulo apresenta a revisão bibliográfica realizada no âmbito desta dissertação. A mesma surge da pesquisa efetuada nas diversas plataformas disponíveis e tem como intuito a abordagem dos diferentes temas relacionados com o trabalho que se pretende desenvolver.

Desta pesquisa destacaram-se três diferentes áreas, que se encontram divididas nas seguintes secções, respetivamente: "Sistemas de aquisição e processamento", 2.1; "Visão artificial", 2.2; "Manipulação", 2.3.

No primeiro subcapítulo é apresentado um estudo comparativo de tecnologias de aquisição e processamento existentes no mercado e suscetíveis de serem utilizadas em aplicações robóticas. O subcapítulo "Aquisição", 2.1.1, apresenta as ferramentas usadas para reconhecimento visual do ambiente, enquanto que em 2.1.2, "Processamento", se enumeram unidades de processamento de dados. Podem destacar-se ainda, na subsecção 2.1.3 diversas publicações relacionadas com sistemas que recorrem a sensores RGB-D e ao microcomputador Raspberry Pi, tecnologias a ser usadas nesta dissertação.

O segundo subcapítulo pretende apresentar as principais técnicas e metodologias de processamento de imagem usadas em ambientes industriais para deteção e reconhecimento de objetos, 2.2.1, assim como os tipos de iluminação recorrentes em visão computacional, 2.2.2.

O terceiro subcapítulo encontra-se relacionado com a área dos manipuladores robóticos. Começa-se por referir as operações *pick-and-place*, 2.3.1. Posteriormente, é feita uma descrição introdutória destes equipamentos, incluindo uma apresentação dos seus tipos, 2.3.2.

2.1 Sistemas de Aquisição e Processamento

No ramo da robótica industrial, a escolha de tecnologias trata-se de um fator preponderante, tendo um impacto relevante em função do fim a que se destinam. A correta seleção de ferramentas e métodos pode definir a qualidade dos resultados, levando a uma atuação e controlo assertivos do equipamento robótico, seja ele um manipulador ou robô móvel. Para esta escolha, destacam-se duas áreas predominantes: aquisição e processamento.

2.1.1 Aquisição

Em manipuladores industriais, para que seja possível a realização de um determinado tipo de tarefa, é frequente a necessidade de perceção do ambiente em que estes se encontram inseridos. Para aplicações de *pick-and-place*, a aquisição compreende a recolha de dados relevantes que, por sua vez, permitirão a criação de um mapa para localização do próprio robô, bem como o reconhecimento de objetos, deteção de obstáculos e outros elementos presentes neste mesmo ambiente. De seguida, encontram-se listadas algumas técnicas de reconstrução 3D usadas no ramo industrial, finalizando com uma comparação entre as mesmas.

2.1.1.1 Sistemas de múltiplas câmaras

Neste âmbito insere-se a visão por estereoscopia, ilustrada na figura 2.1, que usa duas câmaras para perceção do ambiente em que o robô está inserido, retratando o ponto de vista humano. As imagens adquiridas são correlacionadas através da técnica de triangulação, com vista à extração das coordenadas do espaço tridimensional [2].

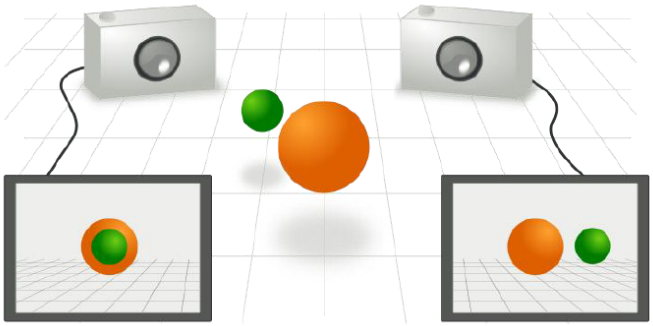

Figura 2.1: Princípio de múltiplas câmaras [1]

O número de câmaras necessárias depende da aplicação em questão, pelo que podem ser usadas mais do que duas câmaras para a reconstrução 3D do ambiente.

Estes sistemas oferecem uma maior precisão nos resultados obtidos, o que se explica pela sua capacidade de perceção da região de interesse a partir de diversos ângulos. São adquiridas imagens de diferentes pontos de vista, dependendo da posição de cada câmara e, tendo em conta a diferença entre estas, é possível estimar as coordenadas de um objeto, recorrendo a um conjunto de expressões matemáticas [16].

Em suma, inicialmente, os sistemas mais utilizados baseavam-se em visão por estereoscopia, no entanto, atualmente estes estão a ser substituídos por câmaras *Time-of-Flight* e sistemas de luz estruturada, por estes garantirem uma maior precisão. [17].

2.1.1.2 Luz estruturada

Os sensores de luz estruturada baseiam-se na emissão de um sinal laser ou de um padrão de luz sobre o objeto a detetar, sendo a região de interesse capturada por uma câmara. Neste tipo

de metodologia, o emissor e recetor tratam-se de componentes independentes, alocados em zonas distintas.

Um exemplo de aplicação desta técnica são os sistemas baseados em triangulação câmara-laser, que se encontram amplamente implementados na indústria, principalmente em aplicações que envolvam inspeção por visão computacional.

O seu método de funcionamento baseia-se na emissão de um sinal infravermelho, que cria, com precisão, uma linha de luz, a qual incidirá no objeto a detetar. Simultaneamente, uma câmara que se encontre estrategicamente localizada, em relação ao laser, recolhe imagens que, posteriormente, permitirão a deteção de corpos rígidos. A figura 2.2 ilustra o princípio de funcionamento deste sistema [2].

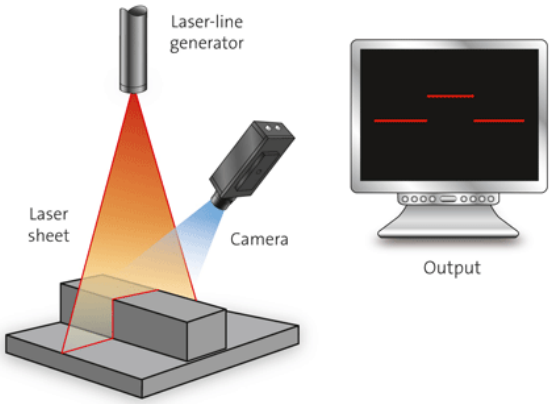

Figura 2.2: Princípio Triangulação Câmara-Laser [2]

Normalmente, a câmara e o laser encontram-se alocados numa calha linear. A abordagem mais comum fixa ambos os componentes na calha, porém existem aplicações em que se torna vantajoso dar ao laser a possibilidade de se mover ao longo de uma viga, o que exige um maior número de calibrações, em função das *frames* adquiridas [16].

Baseado no mesmo princípio de funcionamento da técnica de triangulação câmara-laser, surge o método de reconstrução 3D através de luz estruturada, representado na figura 2.3. O último destaca-se por emitir um sinal luminoso proveniente de um projetor, em detrimento de uma luz infravermelha. Este sinal luminoso é emitido na forma de riscas, as quais incidirão no objeto, desencadeando um padrão de luz que variará em função da forma do objeto. Este padrão é posteriormente captado por uma câmara, com vista à sua interpretação. Assim, pela análise do padrão presente na *frame* adquirida é possível obter a informação da profundidade para cada *pixel* [1].

Ao contrário do método descrito anteriormente, este exige que os objetos sejam estáticos. A maior vantagem prende-se no facto de esta possuir um tempo de resposta tipicamente mais rápido que outros métodos [2].

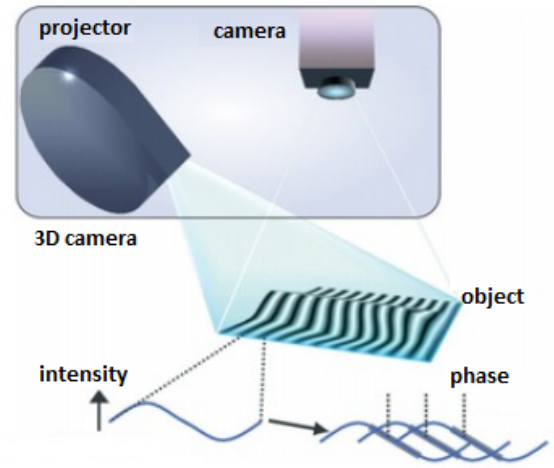

Figura 2.3: Princípio da luz estruturada [2]

2.1.1.3 *Time-of-Flight*

O conceito *Time-of-Flight* aplicado à deteção de objetos baseia-se no cálculo do tempo de voo de um impulso laser, para a obtenção da distância entre o aparelho de medida e o corpo a reconhecer, tendo em conta a velocidade da luz.

Como representa a figura 2.4, é emitida uma luz infravermelha, a qual colide com a superfície do objeto. A intensidade do sinal refletido, bem como o intervalo de tempo medido no retorno deste, permitem a reconstrução 3D da região de interesse.

O alcance destes sensores é por vezes limitado, devido aos limites de potência admissível de radiação laser. Para além disso, a qualidade dos resultados obtidos, depende do tipo de superfície a detetar, enfrentando dificuldades quando incidem em materiais pouco refletores [1].

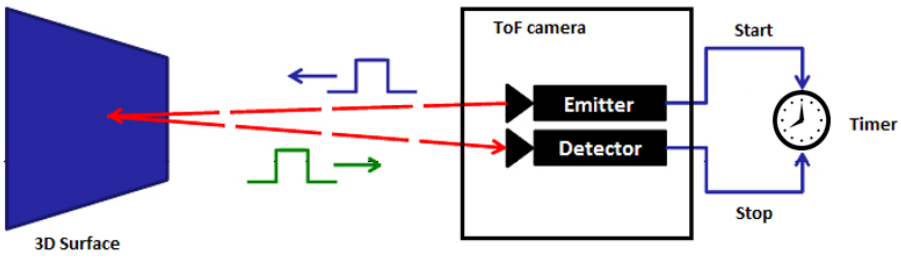

Figura 2.4: Princípio de funcionamento de sensores *Time-of-Flight*

2.1.1.4 Sensores RGB-D

Desde a sua introdução no mercado, os sensores RGB-D têm sido exaustivamente usados em diferentes áreas de estudo. A curiosidade e interesse no uso desta tecnologia advém do baixo custo associado, quando comparado às câmaras *Time-of-Flight*, por exemplo. Este sensor incorpora uma

câmara RGB, *Red*, *Green* e *Blue*, idêntica a uma câmara convencional, capaz de captar o mundo à sua volta, bem como um sensor de profundidade, *Depth*, responsável por fornecer a medida distância a que determinado objeto se encontra da câmara. O fornecimento da profundidade de cada *pixel* torna o sistema mais leve a nível computacional.

As vantagens inerentes a este tipo de equipamento, permitiram o desenvolvimento de aplicações nas mais diversas áreas, desde o reconhecimento de objetos até à deteção de movimento humano, sendo também amplamente usados para mapeamento e localização de robôs móveis [4]. A figura 2.5 apresenta dois sensores RGB-D, sendo que à esquerda se encontra o Microsoft Kinect e à direita o Asus Xtion Pro Live. Estas ferramentas, baseadas na tecnologia da fabricante PrimeSense, foram inicialmente desenvolvidas no âmbito da indústria de videojogos, no entanto as suas funcionalidades, bem como a boa relação preço qualidade, rapidamente cativaram a comunidade robótica.

Figura 2.5: Esquerda: Microsoft Kinect; Direita: Asus Xtion Pro Live [3]

Pinto *et al.* apresentam em [17] uma comparação entre diferentes sensores de profundidade, entre eles duas versões do Kinect, 1 e 2. Quando comparado com outros sensores, tais como o Mesa Imaging ou o Structure Sensor, o Kinect destaca-se pela fácil utilização, para além do baixo custo já referido. No entanto, este apresenta certas limitações, por exemplo, no intervalo de distâncias a que opera. Para se obterem resultados com baixa taxa de ruído associada, é aconselhado o uso do Kinect para detetar objetos que estejam entre 1 a 3 metros de distância do mesmo [17]. As dimensões deste sensor poderão ser também um fator desvantajoso, dependendo da aplicação.

Estudos indicam que as ferramentas baseadas na técnica ToF adquirem informação mais fiável relativamente à profundidade quando comparadas às tecnologias que usam sensores de luz estruturada. Essa discrepância pode observar-se através da comparação do Kinect 1 com o Kinect 2. A primeira versão baseia-se na técnica de luz estruturada, enquanto que a segunda se trata de um sensor *Time-of-Flight*, pelo que os resultados associados à última garantem mais precisão e qualidade [17].

Independentemente do princípio de operação destas, salienta-se o facto de se tratarem de sistemas RGB-D, pelo que estes fornecem o valor da profundidade de cada *pixel* sem necessidade de processamento adicional, o que torna o sistema computacionalmente mais leve.

A figura 2.6 pretende, de uma forma geral, apresentar os componentes e arquitetura que define este tipo de sensor.

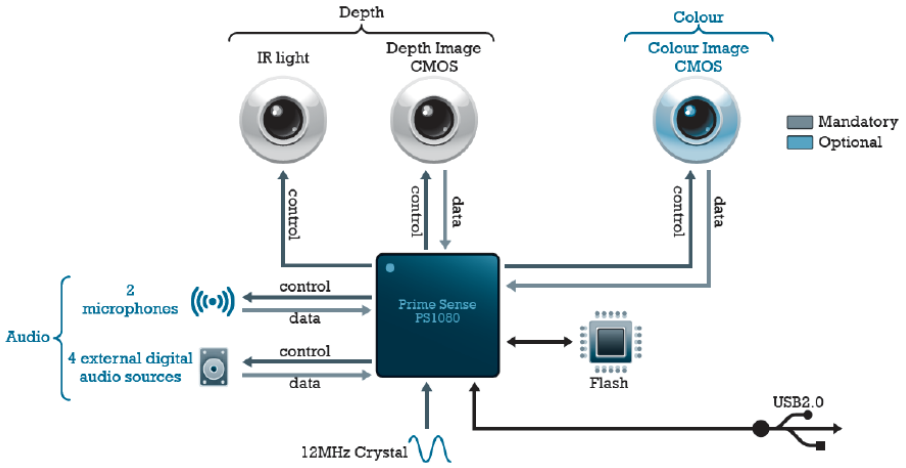

Figura 2.6: Arquitetura funcional do Microsoft Kinect [4]

Dado o crescente interesse no uso desta tecnologia existem dois modelos desta ferramenta, desenvolvidos exclusivamente para aplicações alheias aos videojogos. A tabela 2.1 apresenta as principais características que distinguem as versões 1 e 2 do Kinect.

Tabela 2.1: Distinção entre especificações do Kinect 1 e Kinect 2 [1]

	Kinect 1	**Kinect 2**
Técnica de sensorização	Luz estruturada	*Time-of-Flight*
Câmara RGB (resolução)	640x480 *pixels*, 30 FPS	1920x1080 *pixels*, 30 FPS
Câmara Depth (resolução)	320x240 *pixels*	512x424 *pixels*
Máxima Profundidade	4.5 m	4.5 m
Mínima Profundidade	0.8 m	0.5 m
Campo de visão horizontal	57º	70º
Campo de visão vertical	43º	60º
Motor de inclinação	sim	não
Nº de articulações definidas	20	26
Nº pessoas detetadas em simultâneo	2	6
USB Standard	2.0	3.0
Preço	≈ 100€	≈ 150€

Bastante equiparado à primeira versão do Microsoft Kinect surge o Asus Xtion Pro Live. Baseado na mesma técnica de sensorização, este distingue-se pelas suas dimensões reduzidas, no entanto é desprovido de motor de inclinação, sendo esta feita manualmente. A ausência de motor permite que este seja alimentado via USB, ao contrário do aparelho da Microsoft que necessita de alimentação externa. As suas principais características encontram-se apresentadas na seguinte tabela,

[1] http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1

Tabela 2.2: Especificações do Asus Xtion Pro Live [2]

	Asus Xtion Pro Live
Técnica de sensorização	Luz estruturada
Câmara RGB (resolução)	1280 x 1024 *pixels*, 30 FPS
Câmara Depth (resolução)	640x480 *pixels*
Máxima Profundidade	3.5 m
Mínima Profundidade	0.8 m
Campo de visão horizontal	58º
Campo de visão vertical	45º
Motor de inclinação	não
USB Standard	2.0
Preço	≈ 100€

Graças a esta tecnologia, sensores como o Kinect ou o Asus Xtion Pro Live permitem obter uma quantidade considerável de informação RGB-D a uma *framerate* elevada. Para além deste facto, o preço destes dispositivos é baixo quando comparado com sensores laser e câmaras ToF, o que fez com que fossem adotados para inúmeras aplicações, principalmente na área robótica.

2.1.2 Processamento

Num sistema de robótica, a unidade de processamento de dados tem a seu cargo o controlo de todo o sistema, sendo por isso necessária uma escolha ponderada da mesma.

Existem duas abordagens possíveis para o tratamento de dados. Este pode ser feito numa perspetiva *offline*, em que os dados adquiridos são enviados para uma estação central, na qual as decisões são tomadas e enviados comandos para a atuação do manipulador. A abordagem alternativa defende que o sistema seja dimensionado de maneira a que o processamento se realize em modo *online*, isto é, recorrendo a um equipamento robusto capaz de ter a perceção do ambiente e atuar em função dessa leitura, em tempo real [3].

A seguir apresentam-se algumas opções para o processamento de dados e controlo de processos.

2.1.2.1 Computadores industriais

A solução pioneira para controlo de processos recorre a computadores industriais. Estes destacam-se pela sua fiabilidade, sendo amplamente usados para as mais variadas aplicações. A figura 2.7 ilustra um computador industrial.

Os PLCs, *Programmable Logic Controllers*, são um exemplo de computadores digitais com interface de entradas e saídas para controlar, por exemplo, linhas de produção industrial. A sua escolha pode ser limitadora quando se pretende uma aplicação computacionalmente sofisticada, dotada de, por exemplo, um sistema de visão computacional.

[2] https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/specifications/

Para aplicações na área da robótica industrial, a realidade mais comum inclui computadores de tecnologia mais avançada, com especificações similares aos computadores *desktop*. Embora sejam capazes de fazer face a requisitos temporais restritos, exigidos por norma no chão de fábrica, os computadores industriais possuem como desvantagem o seu custo elevado. Este custo explica-se pelo facto dos equipamentos serem fabricados de modo a suportar condições adversas de operação em ambiente fabril. Para além disto, as suas dimensões não são as mais desejáveis, o que condiciona a sua portabilidade e flexibilidade.

Figura 2.7: PC industrial com um processador *Intel Atom N2800* [3]

2.1.2.2 Computadores convencionais

Para aplicações robóticas no âmbito da investigação científica, isto é, quando o ambiente de operação dos sistemas não é industrial, é frequente o uso de computadores convencionais. Mais comummente denominados computadores *desktop*, figura 2.8, estes são semelhantes aos industriais no que toca a componentes de *software* e *hardware*, tais como o CPU, RAM, *mother-board* e outros. A sua maior vantagem prende-se no custo em relação aos computadores apresentados acima. É notável o esforço que tem vindo a ser feito, por parte das fabricantes, em reduzir as dimensões deste tipo de equipamentos, porém a questão da baixa portabilidade encontra-se ainda por resolver com o uso de computadores comerciais.

Figura 2.8: Esquerda: Computador fixo; Direita: Computador Portátil [4]

[3] www.directindustry.com

Tavares [16] recorreu a um computador convencional como unidade de processamento no controlo do manipulador robótico UR5, aplicando algoritmos de planeamento de trajetórias para tarefas *pick-and-place*.

2.1.2.3 Computadores de placa única

A chegada dos microcomputadores ao mercado trouxe um vasto leque de oportunidades no desenvolvimento de aplicações robóticas. Caracterizados pelo seu reduzido tamanho, estes são frequentemente equiparados a computadores convencionais, no que toca às especificações que os definem. A maior vantagem inerente a esta abordagem reflete-se no baixo custo a si associado, o que leva a que seja adotada para diferentes fins.

Compactos, eficientes e fiáveis são alguns adjetivos que definem estes dispositivos, sendo por isso uma solução cada vez mais estudada para uso na área da automação, monitorização, controlo de equipamentos e outras aplicações na indústria.

Enquanto que estes computadores são tipicamente menos poderosos que os apresentados atrás, esta área da tecnologia encontra-se em notório crescimento, pelo que a tendência remete para o aparecimento de computadores de placa única com melhor poder de resposta a requisitos industriais. É possível verificar que soluções mais recentes já apresentam uma maior capacidade de processamento, oferecendo uma memória RAM da ordem dos GB, no entanto reconhecem-se as limitações que estes ainda apresentam no que toca, por exemplo, à velocidade do processador.

2.1.2.4 Raspberry Pi

O Raspberry Pi, presente na figura 2.9, uma das mais populares representações de computadores de placa única, trata-se da ferramenta de processamento selecionada para esta dissertação.

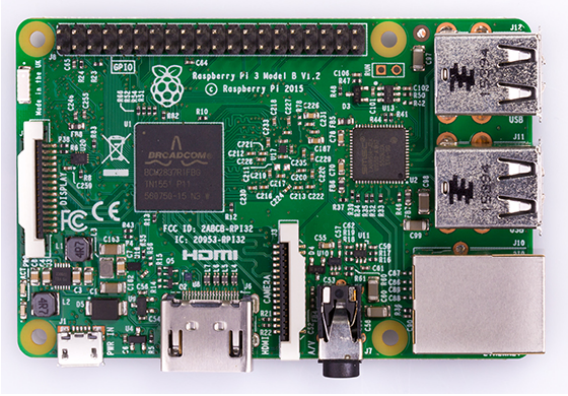

Figura 2.9: Raspberry Pi [5]

Szabó *et al.* apresentam em [18] o controlo de braço robótico a partir de um Raspberry Pi, para ordenação de peças numa mesa. A única limitação, associada à escolha deste equipamento

[4] http://tribogamer.com

no sistema em questão, assenta na duração do processo de compilação de software, o qual chegou a atingir 4 horas.

Ainda que sem uma justificação clara, Lauschner *et al.* afirmam que a ferramenta Raspberry Pi não é suficientemente poderosa para suportar o processamento de imagem, quando associado ao sensor Microsoft Kinect [19]. Esta afirmação pode explicar-se pelo facto do estudo ter sido realizado previamente ao lançamento da última versão do Raspberry, em 2016.

A versão mais recente deste microcomputador incorpora novas funcionalidades que vieram fazer face a algumas limitações encontradas nas anteriores. A tabela 2.3 apresenta as especificações do Raspberry Pi 3.

Tabela 2.3: Especificações do Raspberry Pi 3 Modelo B [5]

	Raspberry Pi 3 Modelo B
SoC	Broadcom BCM2837
CPU	4 x ARM Cortex-A53, 1.2GHz
GPU	*dual-core* Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 *Ethernet*, 2.4 GHz 802.11n *wireless*
Bluetooth	*Bluetooth 4.1 Classic, Bluetooth Low Energy*
Storage	microSD *card*
GPIO	*40-pin header, populated*
Ports	HDMI, 3.5mm *analogue audio-video jack*, 4 x USB 2.0, *Ethernet*, *Camera Serial Interface* (CSI), *Display Serial Interface* (DSI)

Efetivamente, há um vasto número de aplicações nas quais o Raspberry Pi pode ser integrado, desde o controlo de tarefas simples até sistemas mais complexos na área da robótica. Ainda assim, o seu uso destaca-se com maior evidência no ramo educativo e de investigação. De salientar que têm vindo a ser apresentadas soluções baseadas neste dispositivo, objetivando-se a utilização no meio industrial.

A empresa alemã MASS, responsável por comercializar PCs industriais, apresenta o conjunto RPi 07, presente na figura 2.10, que inclui a mais recente versão desta placa, bem como uma consola LCD táctil, para servir de HMI. Para ir ao encontro das tensões de operação definidas por norma para o meio industrial, este sistema solicita uma tensão de alimentação de 24V.

Figura 2.10: Sistema RPi07 [5]

Ainda no mesmo âmbito, surge o Revolution Pi, uma solução proposta pela fabricante alemã Kunbus, especializada em redes *Ethernet* industrial no ramo da automação. Trata-se de um computador industrial baseado no Raspberry Pi Compute Module.

O Compute Module possui as mesmas especificações que um Raspberry Pi convencional, no que concerne ao tipo de processador e memória RAM. O objetivo do seu lançamento centra-se no interesse da criação de uma solução suscetível de ser usada na indústria. Assim, o dispositivo foi desenvolvido para ser agregado a PCBs, que, por sua vez, possuem as interfaces necessárias, por exemplo, GPIO, HDMI, USB, entre outras. Estas placas de circuito impresso são desenvolvidas em função da aplicação a que se destinam, podendo hospedar múltiplos módulos, conforme a complexidade dos processos a controlar. O conceito apresentado pelo Compute Module destaca-se pela flexibilidade e modularidade que pode fornecer a um sistema. Aquando do lançamento deste módulo, a fabricante disponibilizou também uma placa à qual este pode ser agregado. Este conjunto encontra-se ilustrado na figura 2.11.

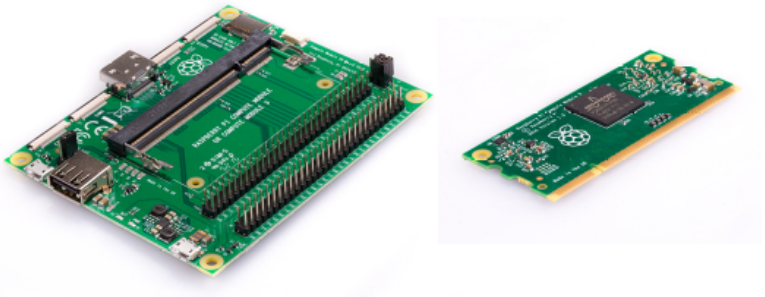

Figura 2.11: Raspberry Pi Compute Module 3 [5]

[5] http://www.mass.de/en/products/panel-pcs/product-details/rpi-07.html

Assim, o Revolution Pi traduz um computador industrial modular de baixo custo, equipado com o Raspberry Pi Compute Module. Objetivando-se a sua instalação em quadros elétricos, este dispositivo encontra-se alocado numa caixa de trilho DIN, disponibilizando também diversas interfaces, as quais, conexões USB, *Ethernet* e HDMI. De acordo com a *datasheet*, este produto cumpre os requisitos necessários para operar em ambiente industrial como um PC real, alimentado a 24V, respeitando o padrão seguido pela indústria. A modularidade do Revolution Pi, ilustrado na figura 2.12, permite que este seja conectado facilmente a outros módulos, por exemplo, de entradas e saídas digitas/analógicas. Dependendo da versão, o preço desta ferramenta pode ir dos 140€ aos 200€.

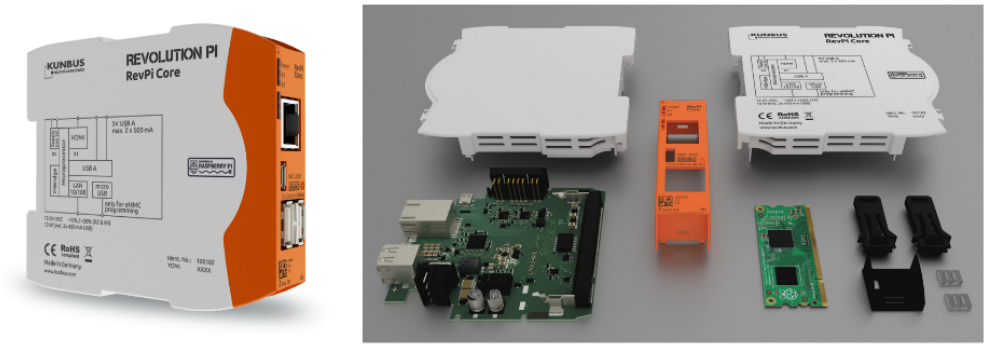

Figura 2.12: Revolution Pi [6]

Esta não se trata da única proposta baseada no Compute Module, também a fabricante Embedded Micro Technology aproveitou as funcionalidades desta placa para criar o My-Pi. Este produto foi criado no âmbito da IIoT com o intuito de conectar uma vasta gama de equipamento industrial.

É evidente o esforço que tem vindo a ser feito no sentido de trazer as potencialidades dos microcomputadores, mais propriamente do Raspberry Pi, para a indústria.

Não obstante, um fator limitante para o uso desta placa num meio tão exigente está associado, por exemplo, à capacidade de processamento necessária para executar determinada tarefa em tempo real. No entanto, esta necessidade pode variar significativamente de sistema para sistema, o que não descarta o seu uso neste ramo. Salienta-se também o facto de este tipo de abordagem ser recente, pelo que a sua aplicabilidade na indústria encontra-se ainda em investigação, porém existem já algumas soluções viáveis.

2.1.3 Aquisição e processamento através de sensor RGB-D e Raspberry Pi

Foi feito um levantamento de publicações que usam um sensor RGB-D em conjunto com o microcomputador Raspberry Pi. O principal objetivo desta pesquisa foi o de obter algum *feedback* relativamente ao sucesso que resulta da simbiose entre estas tecnologias.

[6] https://revolution.kunbus.com/
[6] http://www.embeddedpi.com/

2.1.3.1 SLAM - *Simultaneuos Localization and Mapping*

O conceito de SLAM define uma metodologia de localização e mapeamento simultâneos para robôs móveis. Neste tipo de aplicações *indoor* é frequente o uso de sensores como o Kinect para obter o sistema de coordenadas do robô e reconstruir, em tempo real, o ambiente 3D onde está inserido. Os dados provenientes do sensor são processados pelo Raspberry Pi, que fica encarregue de gerar mapas do ambiente e planear o movimento do robô que, por sua vez, se move em função dos comandos recebidos.

É referido em [20] que o uso destas ferramentas para este fim compreende tempos de processamento de aproximadamente 0,4 segundos. Zheng e colaboradores, propõem em [6] um algoritmo baseado em *multi-frame graph matching* aplicado a nuvens de pontos, provenientes de uma câmara. Este método destaca-se pela sua fiabilidade e robustez quando comparado, por exemplo, ao *pairwise matching*. Para provar a aplicabilidade do algoritmo foi desenvolvido um sistema equipado com um sensor Kinect e um Raspberry Pi, anexados a um robô, mais propriamente o iRobot, como se pode visualizar na figura 2.13. Os resultados comprovam a eficácia deste algoritmo, bem como a correta seleção de tecnologias.

Figura 2.13: Sistema iRobot com respetiva unidade de aquisição e processamento [6]

2.1.3.2 Deteção de objetos e *collision avoidance*

Tendo em conta que o Kinect, para além da imagem RGB, fornece a informação da profundidade de cada pixel presente numa *frame*, o cálculo da distância a objetos torna-se simples. Esta funcionalidade é bastante útil para certos sistemas baseados em visão, uma aplicação recorrente trata-se da implementação de técnicas de *collision avoidance* para diferentes fins.

Em [4] é apresentado um protótipo de apoio a deficientes visuais, com o objetivo de proporcionar a um indivíduo invisual uma maior facilidade na sua movimentação, através do fornecimento de informação do ambiente que o rodeia. Para o efeito, foi desenvolvida uma bengala, denominada bengala PAVEDI, que usa o sensor Kinect como sistema de visão, juntamente com a placa Banana Pi. A associação destas tecnologias permite a apresentação de um sistema que, em tempo

real, é capaz de calcular distâncias e presença de obstáculos, bem como desníveis no ambiente visualizado, alertando a pessoa em função daquilo que observa.

Uma abordagem para um sistema de controlo de um *quadcopter* com *collision avoidance* que recorre ao Kinect e ao Raspberry Pi surge em [21].

Conrad usa as mesmas ferramentas para criar um sistema de focagem automática para a lente de uma máquina fotográfica, ilustrado na figura 2.14. Partindo do principio que o sensor Kinect consegue detetar o movimento de um objeto e fornecer a distância a que este se encontra, é possível ajustar a focagem da lente, recorrendo a um motor [7].

Figura 2.14: Sistema de focagem automática [7]

2.1.3.3 Reconhecimento e interpretação de movimento

O sensor Kinect tem a capacidade de reconhecer as articulações do corpo humano, [17], pelo que esta funcionalidade permitiu a criação de aplicações criativas e inovadoras, segue-se a apresentação de algumas delas.

Madhubala e Umamakeswari apresentam em [22] um sistema de deteção de quedas baseado em visão, sendo o público alvo pessoas idosas. Para o efeito, foi usada um sensor RGB-D para aquisição de imagem, o Kinect, uma vez que é capaz de detetar movimento humano, e um Raspberry Pi para o processamento das *frames* adquiridas. O algoritmo usado baseia-se na extração das características de uma imagem de forma a identificar a forma de uma pessoa, bem como na utilização de uma classificação baseada na média, capaz de distinguir um movimento normal de uma queda. Quando é detetado um movimento anómalo, isto é, quando se deteta uma queda é enviado um alerta para uma pessoa específica através de um *modem*. Usando um sensor de baixo custo, juntamente com um Raspberry Pi, o conjunto de movimentos que o sistema consegue detetar baseiam-se em três tipos de ações: sentar, levantar e cair. A abordagem de visão computacional aqui proposta conseguiu um modelo estático que é resistente às variações na iluminação do ambiente e proporciona ótimos resultados.

Também baseado em deteção de movimento humano, [8] sugere um sistema de controlo e orientação de um robô através do reconhecimento de gestos, por parte de um utilizador, para a monitorização de tarefas domésticas. A figura 2.15 ilustra os resultados do algoritmo de reconhecimento de gestos, bem com o robô a ser controlado.

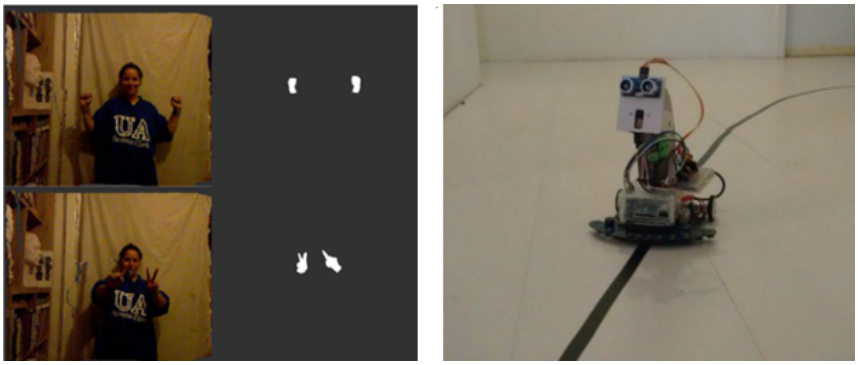

Figura 2.15: Esquerda: Reconhecimento de gestos; Direita: Robô responsável por monitorizar tarefas domésticas [8]

Este sistema foi testado de modo a que o robô possa seguir diferentes caminhos, recebendo comandos e ordens dependendo dos gestos feitos pelo operador à distância.

Em [23] é apresentado um sistema de escrita que reconhece caracteres escritos no ar, sem necessitar de outro dispositivo. Os dados de entrada para a unidade de processamento, Raspberry Pi, são fornecidos pelo utilizador, ao mover a sua mão, e adquiridos pelo sensor Kinect. Estes dados são processados e posteriormente reproduzidos. O trabalho em questão envolveu técnicas sofisticadas de processamento de imagem que se traduziram em resultados bastante satisfatórios, o que remete para a escolha correta do microcomputador, uma vez que a sua capacidade de processamento não limitou de maneira nenhuma o sucesso sistema.

2.2 Visão Computacional

A forma mais comum de um sistema artificial ter a perceção do mundo real é através de visão, sendo que o seu objetivo prende-se em permitir que um robô veja o ambiente que o rodeia da mesma forma que os humanos. Robôs dotados de um sistema de visão caraterizam-se por uma maior autonomia e flexibilidade, tendo a capacidade de interagir com o meio que os rodeia, de forma a realizar tarefas em tempo real.

De entre a elevada diversidade de sistemas que recorrem a visão computacional destacam-se aqueles com aplicação mais recorrente na área da robótica, como por exemplo, para localização de robôs móveis (SLAM), reconhecimento de objetos, *collision avoidance*, entre outros.

Este subcapítulo focar-se-á em métodos de reconhecimento de objetos para operações de *pick-and-place*. Adicionalmente, serão abordados tipos de iluminação existentes.

2.2.1 Deteção e reconhecimento de objetos

A deteção e reconhecimento de objetos por visão computacional advém do interesse constante na automatização de processos na indústria, por exemplo, na inspeção de produtos e tarefas de manipulação.

O conceito de visão computacional implica, muitas vezes, a existência uma fase inicial de pré-processamento aplicado às *frames* adquiridas, para remoção de ruído. A iluminação do ambiente, bem como o tipo de ferramenta de visão utilizada, são exemplos de fontes de ruído frequentes. Posteriormente, surge a etapa de processamento, na qual se insere um conjunto de técnicas e algoritmos capazes de responder à necessidade de deteção e reconhecimento de objetos. De seguida serão apresentados alguns dos algoritmos existentes.

2.2.1.1 *Random Sample Consensous*

O método *Random Sample Consensous* serve para deteção de objetos, sendo responsável por interpretar uma nuvem de pontos, descartando os dados que representam erros, isto é, *outliers* e validando aqueles que traduzem boas estimações, mais propriamente *inliers*, em função de um determinado *threshold*. Isto permite apurar os parâmetros de um modelo matemático que define as caraterísticas de um objeto [24]. A qualidade dos resultados após a aplicação deste algoritmo pode observar-se na figura 2.16, sendo que à esquerda se pode ver a imagem de treino. Adquirida uma *frame* do mesmo ambiente, a partir de outra perspetiva, é possível identificar na última imagem, parâmetros presentes na primeira.

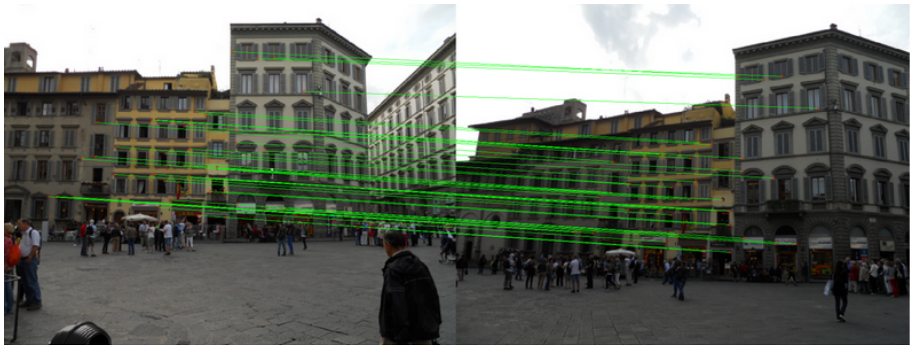

Figura 2.16: Esquerda: Imagem de treino; Direita: Resultados da aplicação do algoritmo RANSAC[7]

Este procedimento assenta numa abordagem oposta às convencionais técnicas de suavização. O RANSAC inicia o seu processo recorrendo a uma quantidade reduzida de dados, selecionando três pontos aleatórios. À medida que o número de iterações aumenta, o modelo vai ganhando consistência, o que se traduz na relevância dos resultados. Métodos mais rudimentares optam por manipular um conjunto considerável de pontos numa primeira fase. Esta técnica promoveu uma

[7] http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15463-f12/www/proj6/www/junyanz/

evolução do princípio dos mínimos quadrados, uma vez que é capaz de evitar o efeito dos *outliers* que muitas vezes danificam as estimativas dos parâmetros [25].

Baseado numa metodologia não determinística, este algoritmo iterativo é bastante utilizado para deteção de objetos e respetiva orientação [26]. A quantidade de erros e o número de parâmetros que definem os objetos podem tornar este processo mais lento, o que se traduz numa desvantagem [25].

2.2.1.2 *Scale-Invariant Feature Transform*

Para garantir um reconhecimento de objetos fiável, é imperativa a seleção prévia de caraterísticas, que possam ser detetadas, independentemente de modificações provocadas, por exemplo, por ruído, mudanças na escala e orientação da imagem ou variações no tipo de iluminação usado.

O descritor 2D *Scale-Invariant Feature Transform*, proposto em [9], veio responder à necessidade de deteção de objetos em diferentes ambientes e condições. O seu princípio de funcionamento tem por base uma fase de treino, na qual se extraem caraterísticas invariantes a mudanças de escala, projeções no espaço 3D ou rotações. O processo de extração de caraterísticas é feito de forma iterativa, sendo o seu resultado armazenado numa base de dados.

A extração de *keypoints* começa por aplicar técnicas de filtragem, capazes de identificar parâmetros invariantes no espaço. De seguida, são definidos índices que classificam o objeto, os quais servirão de *input* na aplicação do método do vizinho mais próximo que, por sua vez, verifica a correspondência entre o objeto previamente classificado e o candidato. A validação final de correspondência é realizada a partir do método de mínimos quadrados.

A qualidade das características selecionadas tem influência direta na eficácia deste método. Apesar da sua elevada robustez, o SIFT tem a si associado um tempo de processamento não ideal. Uma aplicação deste algoritmo encontra-se demonstrada na figura 2.17.

Figura 2.17: Esquerda: Objetos a ser reconhecidos; Direita: Resultado da aplicação do detetor SIFT [9]

2.2.1.3 *Speeded Up Robust Features*

A obtenção de certas características que podem ser usadas para distinguir objetos compõe o processo de deteção. É neste sentido que se insere o algoritmo *Speeded Up Robust Features*, capaz de detetar pontos-chave de uma *frame*, tais como contornos, mudanças de intensidade, entre outros. À volta de cada *keypoint* é definido um vetor e um descritor. Segue-se a combinação de descritores que se baseia na distância entre os vetores, sendo este um fator determinante na eficácia do processo [27].

Primeiramente, o método SURF extrai o pontos que correspondem à região de interesse, isto é, o objeto, tendo por base a análise da intensidade de cores presentes no mesmo. Assim, torna possível a estimação das características que definem a região selecionada, recorrendo a um sistema de reconhecimento 2D. O facto de ser bidimensional pode ser um problema deste detetor, no sentido em que não é capaz de distinguir se o objeto em questão se trata do objeto real ou se está incorporado noutro [25].

Este algoritmo traduz-se num dos mais usados em visão computacional para deteção de características numa imagem. Inspirado pelo método SIFT, o SURF distingue-se por ser mais rápido e robusto a transformações que a imagem pode sofrer [24]. Porém, esta abordagem não resolve ainda a limitação associada ao facto de ser um detetor bidimensional. A figura 2.18 mostra a capacidade deste detetor em reconhecer regiões de interesse previamente definidas.

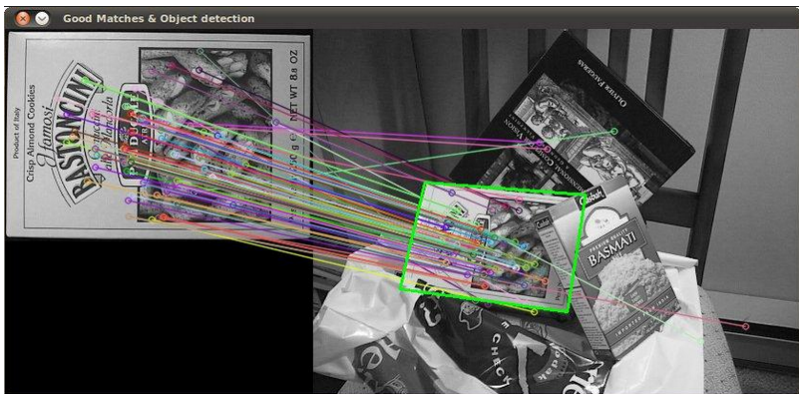

Figura 2.18: Resultado da aplicação do detetor SURF [8]

2.2.1.4 *Normal Aligned Radial Feature*

O método *Normal Aligned Radial Feature*, [10], surge com a necessidade da deteção de objetos tridimensionais.

Esta técnica de deteção de pontos de interesse recorre a imagens RGB, nas quais a distribuição da cor é dada em função da profundidade de cada *pixel*. Adquirida a imagem RGB de profundidade do ambiente, recorre-se à deteção dos contornos dos objetos para identificar esses pontos [25].

[8] http://robocv.blogspot.pt/2012/02/real-time-object-detection-in-opencv.html

A deteção dos pontos de interesse tem em conta caraterísticas dos contornos do objeto, tais como a forma. Além disso, seleciona pontos suscetíveis de serem detetados a partir de uma perspetiva diferente.

Após deteção desses pontos, os mesmos são avaliados e quantificados em função dos pontos que o rodeiam, sendo enaltecidos aqueles cuja diferença de profundidade com a vizinhança é maior. É também determinada a direção dominante na qual ocorrem mudanças bruscas de profundidade e, em função disso, é calculado um ponto de interesse que represente as diferenças entre as direções dominantes, bem como as mudanças existentes na superfície em que está inserido o ponto. Posteriormente, é aplicada uma suavização nos valores de interesse, seguida de uma supressão não máxima para apurar os pontos finais.

Por fim, calcula-se o descritor NARF. Este processo, ilustrado na figura 2.19, compreende a sobreposição de um padrão sobre um ponto de interesse, cujos raios possuem diferentes valores, refletindo a mudança de profundidade verificada na direção definida pelos mesmos.

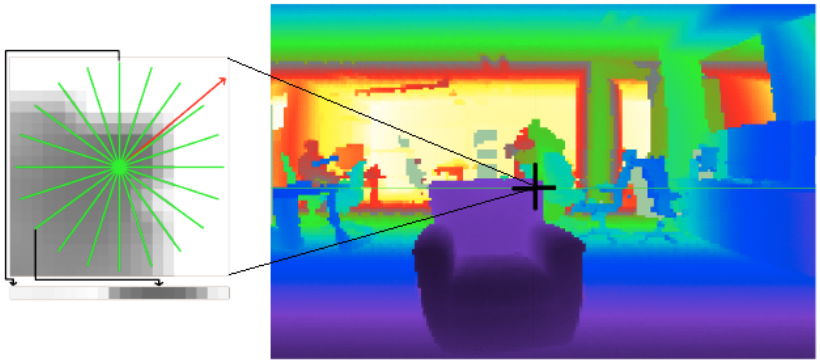

Figura 2.19: Esquerda: Padrão a partir do qual se obtém o descritor NARF; Direita: Ponto de interesse [10]

Da observação da figura 2.19, pode identificar-se à direita, na imagem de profundidade, o ponto de interesse sinalizado a preto no canto superior direito do objeto. A avaliação deste ponto em função da sua vizinhança, usando o padrão presente à esquerda, permitirá a extração do descritor de NARF [10].

2.2.2 Técnicas de iluminação

O tipo de iluminação de um sistema baseado em visão artificial é um fator de extrema relevância, que influenciará a necessidade de pré-processamento das *frames* adquiridas. Para uma escolha assertiva e adequada, devem ser tidos em conta aspetos como as caraterísticas da superfície dos objetos a detetar, o que conduzirá a melhores resultados. A seguir, sugerem-se os tipos de iluminação a ser tomados em consideração, acompanhados de uma representação ilustrativa.

Das técnicas de iluminação existentes, é possível fazer uma primeira classificação quanto à direção do feixe de luz emitido, podendo esta ser direta, figura 2.20 (a) , quando a fonte de luz é instalada acima do objeto a detetar, ou difusa, na qual os feixes de luz são emitidos em diferentes

direções. A abordagem apresentada em (b), da mesma figura, é recorrente quando as características da superfície do objetos são relevantes.

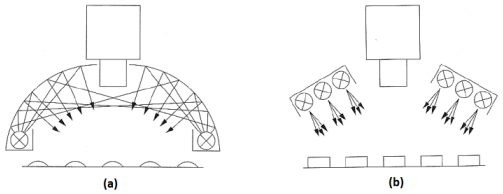

Figura 2.20: Classificação quanto à direção do feixe. (a) Direta e (b) Difusa [11]

Dependendo da aplicação, a posição da fonte de luz relativamente ao objeto e à câmara pode ser um fator preponderante nos resultados. Assim, podem distinguir-se duas posições possíveis para o emissor de luz. A técnica de iluminação frontal posiciona a fonte de luz do mesmo lado da câmara, 2.21 (a). Enquanto que num sistema em contraluz, o objeto encontra-se entre a fonte luminosa e a câmara, (b).

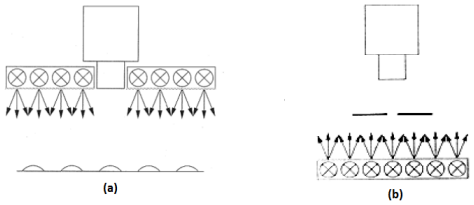

Figura 2.21: Classificação quanto à posição da fonte luminosa. (a) Técnica de luz frontal; (b) Técnica de contraluz [11]

Por fim, é possível fazer a distinção de técnicas em função da quantidade de luz emitida. Neste âmbito, surge a técnica de iluminação em campo claro, figura 2.22 (a), quando a maior parte de luz é refletida para a câmara. Contrariamente, nas situações em que maior parte da luz não atinge a câmara, a iluminação em causa é a realizada em campo escuro, (b).

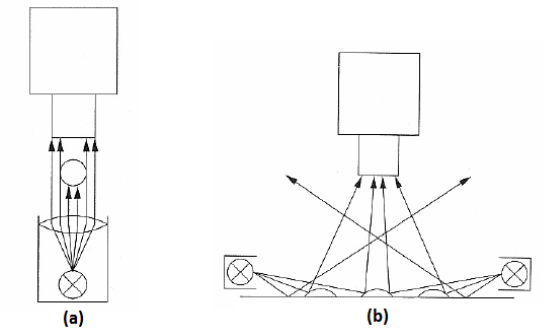

Figura 2.22: Classificação quanto quantidade de luz emitida (a) Campo claro e (b) Campo escuro [11]

2.3 Manipuladores Industriais

O uso de manipuladores no meio industrial tem vindo a crescer nos últimos anos. Esta tendência assenta no facto de estes serem capazes de desempenhar tarefas com maior velocidade, precisão e qualidade que um operador humano. Assim, os manipuladores robóticos permitem influenciar positivamente a eficiência de processos na indústria. Para além desta realidade verificada na indústria, os manipuladores robóticos despertam constantemente o interesse da comunidade científica, no âmbito do desenvolvimento e criação de soluções versáteis e inovadoras.

2.3.1 Operações *pick-and-place*

As operações de *pick-and-place* são frequentemente associadas a manipuladores robóticos, pelo que o seu uso vai desde a indústria alimentar, figura 2.23, até à indústria eletrónica [28, 14, 15].

Este tipo de tarefa, realizada por um manipulador robótico, compreende o transporte de um objeto do ponto inicial, no qual se encontra, até um destino, seguindo uma trajetória assente num algoritmo de pesquisa. O tipo de robô mais apropriado para uma aplicação deste género depende de fatores como a velocidade exigida na execução da tarefa.

Figura 2.23: Operação de *pick-and-place* na indústria alimentar [9]

É possível dividir uma operação de *pick-and-place* em duas fases distintas: sensorização e atuação. A sensorização trata a deteção e reconhecimento dos objetos em causa, enquanto que a atuação compreende o processo associado ao movimento do manipulador até ao destino e à manipulação das peças, através de uma garra.

Recorrendo a uma metodologia de visão baseada em múltiplas câmaras, Gecks e Henrich apresentam em [29] um sistema de cooperação homem-robô, capaz de realizar tarefas *pick-and-place*. Trata-se de um sistema de processamento de imagem de tempo real com a capacidade de detetar e evitar obstáculos, com efeito na alteração da trajetória.

[9] https://www.packworld.com/

2.3.2 Tipos de manipuladores

Os braços robóticos podem ser classificados quanto ao tipo de geometria em dois tipos: manipuladores em série e em paralelo. O tipo de manipulador em série baseia-se na alocação de elementos rígidos desde a base do robô até à ferramenta de manipulação. Por outro lado, a abordagem em paralelo descreve um manipulador dotado de um determinado grau de liberdade em função do número de articulações que possui, pelo que o único elemento fixo é a sua base [28] [16]. Facilmente se pode destacar a arquitetura paralela em detrimento da arquitetura em série, pela sua elevada velocidade, precisão e rigidez mecânica. Porém, o custo elevado que dela advém, leva muitas vez à escolha de manipuladores série.

A mobilidade de um manipulador é dada pela presença de articulações, sendo que estas podem ser prismáticas, isto é, deslizantes, ou rotativas [30]. Para ambiente industrial, salientam-se os seguintes tipos de configurações mais frequentes em manipuladores robóticos:

- Antropomórfico (RRR), figura 2.24 - Composto por três articulações rotativas, é a abordagem de movimento que mais se assemelha ao braço humano;

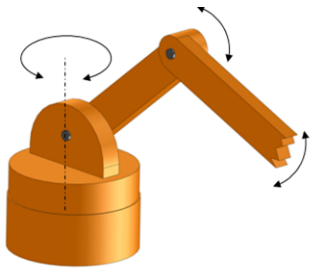

Figura 2.24: Manipulador Antropomórfico (RRR)

- Cartesiano (PPP), figura 2.25: Definido por três articulações prismáticas, este tipo de manipuladores possui uma pequena área de trabalho e caraterizam-se pelo elevado grau de rigidez mecânica e simplicidade no controlo [30].

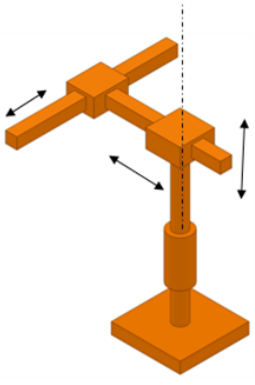

Figura 2.25: Manipulador Cartesiano (PPP)

- Esférico (RRP), figura 2.26: Compreende duas articulações rotativas e uma prismática, possuindo uma área de trabalho superior ao manipulador cilíndrico. Ao mesmo tempo, é menos rígido e possui um controlo mais exigente [30];

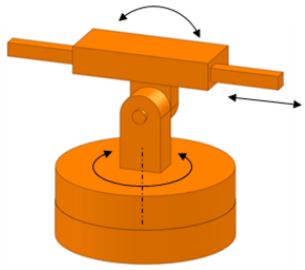

Figura 2.26: Manipulador Esférico (RRP)

- Cilíndrico (RPP), figura 2.27: Configuração inclui uma articulação rotativa e duas prismáticas. Possui uma rigidez mecânica inferior ao tipo cartesiano, tendo a si associado um controlo mais complexo; [30].

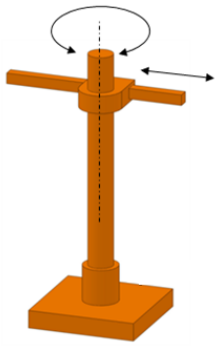

Figura 2.27: Manipulador Cilíndrico (RPP)

- SCARA (RRP), figura 2.28: Possui duas articulações rotativas e uma prismática, sendo adequado para manipular pequenos objetos.

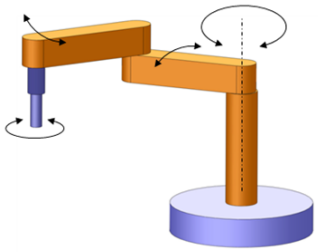

Figura 2.28: Manipulador SCARA (RRP)

Os vários tipos de configurações para manipuladores são bastante distintos, pelo que cada um desempenha uma diferente tarefa, o que leva a concluir que não existe uma solução que se destaque em relação às outras.

A abordagem adotada para o desempenho de tarefas de *pick-and-place* compreende, maioritariamente, o uso de manipuladores do tipo antropomórfico, pela capacidade de se aproximar a um braço humano. Este facto permite incutir ao manipulador a realização de tarefas repetitivas, que normalmente seriam efetuadas por um operador.

Esta dissertação recorre a um manipulador paralelo dotado de seis graus de liberdade, mais propriamente um manipulador MOTOMAN. Este tipo de manipulador encontra-se limitado a uma pequena área de trabalho, tendo a si associado modelos de cinemática e dinâmica complexos, para além do controlo de movimento ser exigente [28].

Capítulo 3

Arquitetura do Sistema

Uma vez que esta dissertação trata o estudo do uso de um microcomputador para o desempenho de tarefas do ramo da robótica industrial, é imperativa a apresentação da arquitetura da solução encontrada. Assim, em primeiro lugar, serão apresentados neste capítulo os componentes de *hardware*, em 3.1, bem como o *software* usado, em 3.2. Em segundo lugar, descreve-se a arquitetura funcional resultante da escolha previamente fundamentada das ferramentas, na secção 3.3.

3.1 Componentes de *Hardware*

Neste subcapítulo serão enumeradas as diferentes ferramentas usadas no decorrer desta dissertação, assim como a justificação da sua escolha.

3.1.1 Ferramenta de aquisição

Para o objetivo pretendido nesta dissertação, foi necessária a seleção de um sensor para aquisição de informação do cenário a três dimensões. Como referido em 2.1.1, existem diferentes tecnologias para o efeito, pelo que a sua seleção foi feita em função dos dispositivos disponíveis, entre os quais, Kinect 1, Kinect 2 e Asus Xtion Pro Live, apresentados em 2.1.1.4.

Tipicamente, soluções industriais que envolvam visão computacional 3D, recorrem ao uso de sensores *Time-of-Flight* (ToF). Esta escolha pode ser justificada pelo facto da sua tecnologia ser menos sensível a mudanças nas condições de iluminação [17, 31].

Um exemplo de um sensor ToF da lista acima indicada é o Kinect 2. No entanto, este não foi selecionado pois necessita de ser conectado a portas USB 3.0, devido à elevada quantidade de dados que envia, enquanto que o Raspberry Pi 3 apenas possui portas USB 2.0. A grande distinção entre estes tipos de portas prende-se na velocidade a que efetuam a transferência de dados, mais concretamente, portas USB 2.0 permitem uma velocidade de 280 Mbit/s, enquanto que em dispositivo de conexão USB 3.0 são dotados de uma velocidade na ordem dos 3.2Gbit/s. Esta caraterística, apesar de não inviabilizar o seu uso, pode levar à perda de dados .

Relativamente aos restantes sensores, Kinect 1 e Asus, estes apresentam caraterísticas muito similares, sendo que o selecionado foi o segundo. Esta escolha deveu-se ao facto de, após instalação das *drivers* (bibliotecas) necessárias no microcomputador, a aquisição de dados apenas foi possível através do Asus Xtion Pro Live. Esta afirmação encontra-se fundamentada no subcapítulo 4.1.2.

Como sensor RGB-D, este dispositivo baseado na técnica de luz estruturada, apresenta um emissor infravermelho (IV), bem como um recetor infravermelho, e ainda uma câmara RGB, identificados na figura 3.1. O emissor tem a seu cargo a projeção de um sinal infravermelho, o qual é capturado pela câmara monocromática (recetor). Por sua vez, a câmara RGB fornece a informação da cor de cada *pixel*, numa *frame* de resolução 640x480 *pixels*.

Figura 3.1: Identificação dos constituintes do sensor Asus

O esquemático da figura 3.2 apresenta o campo de visão horizontal e vertical do sensor selecionado, respetivamente, 58º e 45º. As especificações do Asus Xtion referem que este possui um intervalo admissível de profundidade de 0.8m a 3.5m, no entanto, foi possível obter medições num plano mais próximo, sendo por isso a profundidade mínima de, aproximadamente, 0.5m.

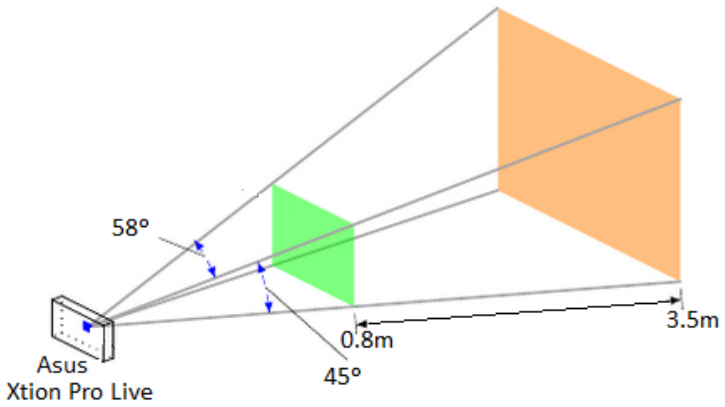

Figura 3.2: Ilustração do campo de visão do sensor Asus Xtion Pro Live e respetivo intervalo de profundidade admissível [1]

[1] Adaptada de [32]

De modo a alocar o sensor numa posição fixa, recorreu-se a um tripé, ao qual se agregou uma estrutura de fixação desenvolvida propositadamente para este fim. Esta peça de encaixe foi desenhada em SolidWorks[2]e produzida numa impressora 3D.

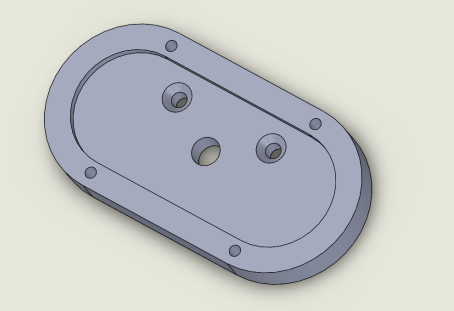

Figura 3.3: Desenho da estrutura de fixação do sensor no tripé, dimensionada em SolidWorks

3.1.2 Computador de desenvolvimento e teste

O estudo da viabilidade do uso do Raspberry Pi em aplicações industriais centra-se na avaliação do desempenho deste microcomputador, em detrimento de um computador industrial. Neste sentido, para o desenvolvimento do *software* de aplicação recorreu-se a um computador convencional, da fabricante Toshiba, pelo que a fase de teste do mesmo foi realizada em ambos os dispositivos. O computador convencional usado, apesar de não se tratar de um computador industrial, pelas suas especificações, poderá aproximar-se de tal.

A tabela 3.1 apresenta as principais diferenças entre os computadores selecionados. A distinção entre arquiteturas RISC e CISC encontra-se descrita na secção A.1.1. Para além destas caraterísticas, destaca-se o baixo preço do microcomputador selecionado, aproximadamente 50€, face ao computador usado e computadores industriais, em geral.

Tabela 3.1: Comparação de caraterísticas entre computador convencional e microcomputador usados na presente dissertação

	Raspberry Pi 3 Modelo B	Toshiba Satellite
Arquitetura	RISC	CISC
Sistema Operativo	Raspbian Jessie Pixel	Ubuntu 16.04.2 LTS
Processador	ARM Cortex-A53 *quad-core*	Intel Core i5-4200U *quad-core*
Velocidade do processador	1.2GHz	1.6GHz
Memória RAM	1GB	6GB

Como referido em 2.1.2.4, o sistema operativo do Raspberry Pi encontra-se alocado num cartão microSD, sendo para o feito usado um cartão de memória de classe 10, com um armazenamento de 32GB. A sua classe determina a velocidade de escrita e leitura que está subjacente, respetivamente, 10 MB/s e 45MB/s.

[2] *Software* de desenho 3D

3.1.3 Manipulador e garra

Para a realização de tarefas *pick-and-place*, recorreu-se ao manipulador Motoman presente na sala de robótica. Este braço robótico é do tipo antropomórfico, referido em 2.3.2, sendo constituído por seis articulações rotativas, suportando uma carga estipulada de 6Kg.

Associado ao controlador NX100, o manipulador aqui abordado, possui uma consola de interface com o utilizador, que permite o controlo manual ou remoto do manipulador. Durante esta dissertação o controlo do robô foi realizado remotamente, pelo que a comunicação com o mesmo é feita via *Ethernet*, usando o *File Transfer Protocol* (FTP).

Ao manipulador foi agregada uma garra, da fabricante FESTO, de modo a permitir o transporte do objeto pretendido. A garra usada carateriza-se por ser de atuação pneumática, recorrendo para o efeito a um sistema de ar comprimido.

A figura 3.4 apresenta o conjunto manipulador e garra, acoplados por meio de uma estrutura de encaixe, usados para operações *pick-and-place*.

Figura 3.4: Sistema de atuação composto pelo manipulador Motoman e garra pneumática

Uma vez que a abertura da garra disponível não era suficiente para abranger o objeto pretendido, optou-se por dimensionar uma extensão adaptável à mesma, que se encontra representada individualmente no desenho da figura 3.5 e acoplada à garra juntamente com o objeto, na figura 3.6. Os desenhos associados ao seu dimensionamento foram realizados, novamente, no SolidWorks.

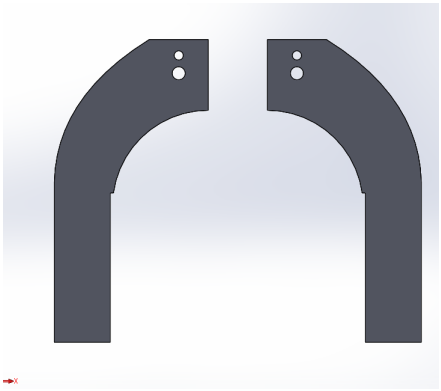

Figura 3.5: Desenho da extensão para a garra, dimensionada em SolidWorks

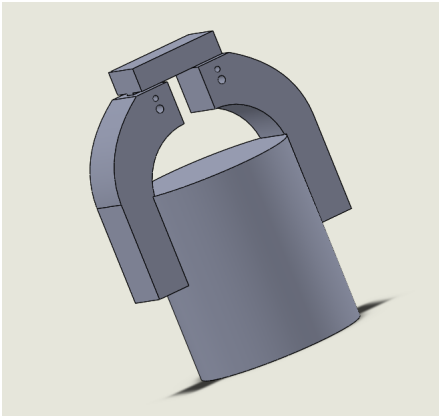

Figura 3.6: Desenho do conjunto: garra, extensão e objeto, dimensionada em SolidWorks

3.2 *Software*

Com o objetivo de suportar as ferramentas usadas e no sentido de se desenvolver o *software* de aplicação, foi necessário recorrer a um conjunto de bibliotecas, sendo que as de maior relevância em termos de utilização correspondem à *Point Cloud Library*, na subsecção 3.2.1 e à *Open Natural Interface*, em 3.2.2.

Adicionalmente, salienta-se o uso de um conjunto de funções para o controlo do manipulador Motoman. Para além disso, de modo a garantir a comunicação com o controlador do manipulador, usaram-se alguns módulos do *package* Qt-essentials. Para efeitos de compilação de código fonte, foi utilizada a ferramenta CMake.

Assim, a tabela 3.2 enumera um conjunto ferramentas usadas, bem como a sua versão no computador de desenvolvimento e no microcomputador de teste.

Tabela 3.2: Ferramentas de *software* instaladas e respetivas versões em cada plataforma de processamento

Toshiba Satellite	Raspberry Pi 3 Modelo B
PCL 1.8	PCL 1.8
OpenNI2	OpenNI2
Cmake 3.5.1	Cmake 3.6.1
Qt 5.5.1	Qt 4.8.6

3.2.1 *Point Cloud Library*

A *Point Cloud Library* (PCL) consiste numa biblioteca composta por um conjunto de classes implementadas em linguagem C++, utilizadas para o desenvolvimento de *software* aplicado a sistemas de visão 3D.

Nela encontram-se implementados algoritmos para o processamento de nuvens de pontos de diferentes dimensões e tipos. Uma nuvem de pontos corresponde a uma representação 3D de um determinado cenário. Deste modo, a biblioteca é composta por algoritmos de filtragem, estimação, reconstrução de superfícies, aproximação de objetos por um modelo, segmentação, entre outros. Os mesmos podem ser usados, por exemplo, para remover valores não desejados, associados a dados ruidosos, segmentação de partes relevantes de um cenário, cálculo de pontos-chave (*keypoints*) e descritores para o reconhecimento de objetos, com base na sua forma geométrica.

O formato de ficheiro tratado pela PCL, responsável por guardar o conjunto de pontos que definem uma nuvem denomina-se *Point Cloud Data* (PCD). Os algoritmos desta biblioteca, encontram-se subdivididos em módulos temáticos, alguns deles usados neste trabalho. [33]

3.2.2 *Open Natural Interface*

Para que seja possível o processamento de uma nuvem de pontos, usando a biblioteca acima referida, é necessário que haja um fase prévia de aquisição da nuvem. Para tal, recorreu-se à biblioteca *Open Natural Interface* (OpenNI), responsável por fazer o intercâmbio entre o sensor e o *software* de aplicação.

Esta ferramenta tem a seu cargo a leitura dos dados adquiridos do sensor e conversão dos mesmos para o formato nuvem de pontos, capaz de ser tratado pela PCL.

A OpenNI possui diferentes versões, associadas a sensores distintos, mais propriamente, OpenNI1 e OpenNI2, correspondendo, respetivamente, ao Kinect 1 e Asus Xtion Pro Live.

3.3 Arquitetura Funcional

Os componentes de *hardware* selecionados, acima listados, integram a arquitetura apresentada na figura 3.7.

De um modo geral, a aquisição dá-se por meio do sensor Asus Xtion Pro Live, o qual se encontra conectado a um *powered* USB *hub*[3], para garantir que a ele chega a corrente de que necessita para operar. Por sua vez, este *hub* estabelece a interface entre o sensor e o Raspberry Pi, que aloja, compila e testa o *software* de aplicação desenvolvido para o efeito.

Por outro lado, recorreu-se a um *router* para garantir a comunicação, via *ethernet*, entre o microcomputador e o controlador NX100. Consequentemente, o controlador tem a seu cargo a atuação do manipulador e garra, sendo que, para tal, a consola associada ao mesmo necessita de estar configurada no modo de controlo remoto.

De forma a não sobrecarregar o Raspberry com dispositivos periféricos (monitor, teclado e rato), isto é, usando este microcomputador numa abordagem *headless* [4], o acesso ao mesmo é feito via *Secure Shell* (SSH) através do computador da fabricante Toshiba. Neste tipo de acesso, a comunicação é feita via *ethernet*, recorrendo ao mesmo *router* referido acima. A não utilização do sistema operativo gráfico (Pixel), tem como objetivo a poupança de recursos de processamento

[3] Aparelho que incrementa o número de portas USB de um computador, tendo a especificidade de fornecer energia aos dispositivos a ele conectados

do microcomputador, tornando a sua utilização um processo mais leve. Assim, o computador convencional, Toshiba Satellite, funciona como interface com o utilizador e, para além disso, traduz a ferramenta de desenvolvimento do *software* de aplicação, servindo também como termo de comparação para com o Raspberry Pi.

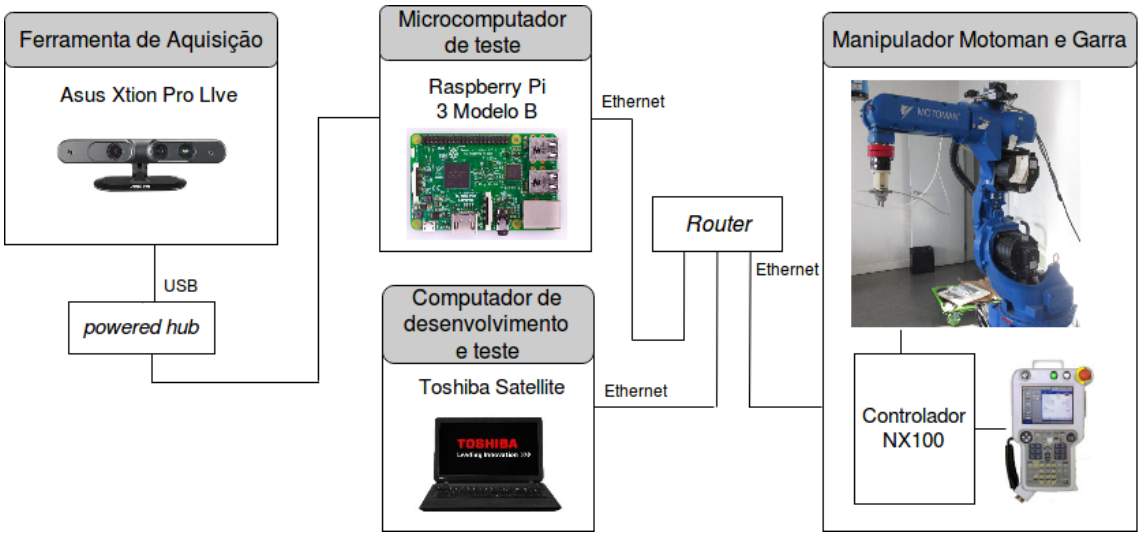

Figura 3.7: Arquitetura funcional do sistema em estudo

Deste modo, a escolha de equipamentos que constituem a arquitetura funcional aqui apresentada, tem como objetivo conferir a mínima sobrecarga ao microcomputador, de forma a apurar deste resultados satisfatórios.

Neste âmbito, o Raspberry Pi tem apenas a seu cargo o processamento de dados, pelo que não fornece energia a nenhum dispositivo externo, isto é, módulo de entrada ou saída.

[4]Sistema ou dispositivo configurado para operar sem monitor, teclado e rato

Capítulo 4

Setup do Microcomputador

O presente capítulo apresenta duas abordagens para o *setup* do Raspberry Pi. Entenda-se por *setup* do dispositivo, o conjunto de procedimentos a realizar com vista à preparação do mesmo, de modo a executar a aplicação pretendida. Por sua vez, o conceito de preparação do microcomputador inclui a instalação das bibliotecas necessárias, referidas na secção 4.1. Este capítulo termina com a secção 4.2, que apresenta uma discussão comparativa relativamente aos métodos apresentados nas subsecções anteriores.

A fase de *setup* aqui apresentada, mais propriamente, a compilação de bibliotecas para o microcomputador selecionado, teve por base o estudo deste tipo de arquitetura, anexado em A.

4.1 Instalação de Bibliotecas

O estudo iniciou-se com a especificação da aplicação a implementar neste microcomputador, bem como requisitos necessários para o seu desenvolvimento. Assim, estipulou-se que o programa a criar teria de ter a capacidade de detetar a presença, ou não, de um cilindro numa mesa, bem como as respetivas coordenadas do mesmo, objetivando-se a deslocação de um manipulador até ao mesmo para a execução de operações *pick-and-place*. A fase de reconhecimento do objeto exigiu a instalação de um conjunto de bibliotecas com funções implementadas para o efeito, mais precisamente, a *Point Cloud Library* e a *Open Natural Interaction*, referidas em 3.2. Assim, serão apresentadas as técnicas estudadas para a compilação das mesmas.

Num computador de sistema operativo baseado em Linux, a instalação de bibliotecas pode ser efetuada pela compilação do seu código fonte ou pela instalação de uma versão já compilada. Enquanto que na primeira abordagem é possível aceder à versão mais recente da biblioteca, bem como efetuar alterações na mesma, no segundo método, o código fonte não está acessível, e nem sempre esta corresponde à versão mais atualizada. A maior vantagem da instalação de bibliotecas já compiladas é exatamente o facto de já estarem compiladas, o que reduz a duração do processo. Porém, estes *packages* já compilados destinam-se a uma versão específica do sistema operativo, pelo que nem sempre estão disponíveis para todas as distribuições. Pelo contrário, o código fonte de uma biblioteca é transversal a qualquer distribuição.

Tendo em conta que o Raspberry Pi é uma plataforma ainda pouco usada num domínio de aplicações que incluam, por exemplo, o processamento de imagem a três dimensões, tanto a PCL como a OpenNI necessitam de ser compiladas a partir do seu código fonte. Já a ferramenta CMake, usada em comunhão com os compiladores para gerar código executável, assim como os módulos do *package* Qt-essentials, adquirido para efeitos de comunicação, foram instalados na sua versão já compilada, uma vez que se encontravam disponíveis para esta plataforma.

Neste sentido, é feita a apresentação de duas técnicas de compilação, comummente usadas em plataformas ARM, respetivamente, *cross-compilation*, 4.1.1, e compilação nativa, 4.1.2. No entanto, primeiramente, é imperativo entender do que trata a compilação de código fonte, ou seja, independentemente da abordagem, 4.1.1 ou 4.1.2, quais as fases que compreendem a compilação de código.

O processo de compilação, esquematizado na figura 4.1, faz uso de um conjunto de ferramentas específicas para produzir código executável. Começa por recorrer a um pré-processador (Cpp) para confirmar a existência de dependências incluídas nos ficheiros *c, c++* e respetivos *headers*, *h*. O código pré-processado, *i* e *ii*, passa pelo compilador (gcc/g++), que, por sua vez, o traduz em código *assembly* específico para a arquitetura em causa. Entenda-se por *assembly*, o código passível de ser executado por um processador. Para isso, este é interpretado por um *assembler* (*as*), que origina o código máquina, que resulta em código objeto. Em alguns casos, este pode ser executado imediatamente, no entanto, na maioria das situações, existem diferentes objetos que têm de ser conectados para funcionarem. Para o efeito, é usado um *linker* (*ld*) que agrupa todo o código num só ficheiro a executar.

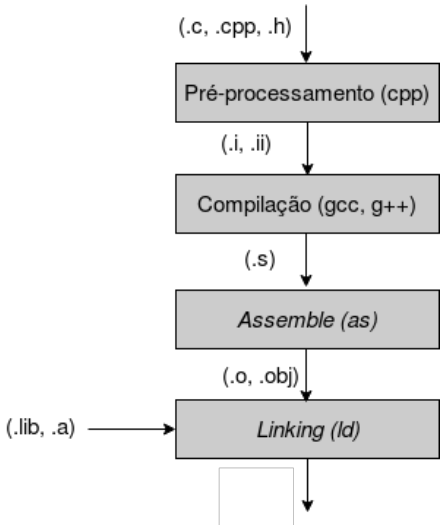

Figura 4.1: Fases que compreendem o processo de compilação

4.1.1 *Cross-compilation*

Um compilador de código não só é capaz de compilar código para o computador no qual se encontra, como também pode ser configurado para gerar código executável para um computador

de arquitetura distinta. É neste conceito de versatilidade de um compilador que assenta o processo de *cross-compilation*, ou compilação cruzada, sendo especialmente útil quando se pretende compilar código para correr em plataformas de baixo poder computacional. Por exemplo, os sistemas embebidos, que normalmente apresentam uma quantidade de memória RAM reduzida.

Este processo exige o uso de um compilador específico, denominado *cross-compiler*. Certos fabricantes disponibilizam esta ferramenta, no entanto, quando isto não se verifica, a mesma terá de ser criada no computador a partir do qual se pretende fazer a compilação cruzada. A figura 4.2 apresenta o conceito de criação de um compilador específico, bem como o de *cross-compile* propriamente dito.

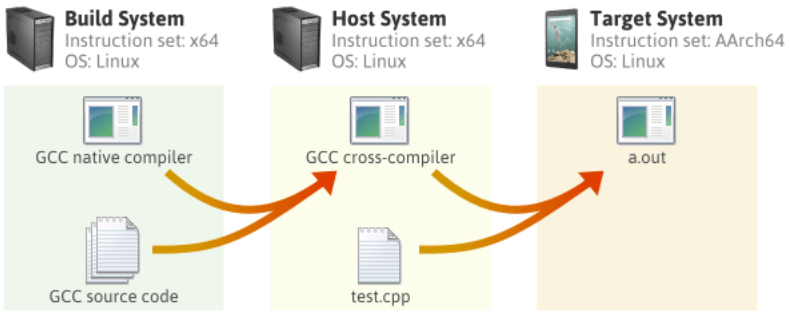

Figura 4.2: Sistemas envolvidos no processo de *cross-compilation* [1]

Primeiramente, identifica-se o sistema *build*, responsável por gerar o *cross-compiler*, a partir do compilador local (gcc/g++) e respetivo código fonte. Uma vez criado o compilador, torna-se possível desenvolver um qualquer programa ou então usar o código fonte de bibliotecas, com vista à sua compilação para a plataforma pretendida. Realça-se o facto de esta etapa se realizar no mesmo computador, no entanto, este designa-se agora sistema *host*, no qual é criado o código executável para o sistema *target*.

Em suma, é usado um compilador local para criar um *cross-compiler* que, por sua vez, compila programas para uma arquitetura ARM, neste caso.

Sendo o sistema *target* uma arquitetura ARM, encontram-se disponíveis diversas versões de *toolchains* para gerar código para este tipo de processador. Tendo em vista a compilação de código em linguagem C e C++, recorreu-se aos respetivos compiladores, gcc-arm-linux-gnueabihf e g++-arm-linux-gnueabihf [2].

No âmbito desta dissertação pretendia-se compilar as bibliotecas num computador convencional, de forma a serem executadas no Raspberry Pi 3. Apesar de o *cross-compiler* usado ser universal, no sentido em que se aplica a qualquer dispositivo ARM, baseado em Linux, é sabido que este tipo de arquitetura contempla um vasto leque de tipos de processadores, pelo que cada microcomputador possui caraterísticas distintas. Neste sentido, há um conjunto de variáveis que

[1] http://preshing.com/20141119/how-to-build-a-gcc-cross-compiler/

[2] "gcc" ou "g++" indentifica linguagem a ser compilada; "arm" identifica a arquitetura do sistema target; "linux-gnueabihf" identifica o sistema operativo Linux (Unix), o tipo de interface binária de aplicação e o tipo de arquitetura, *ARM Hardware Floating point*

permitem especificar o sistema *target* em estudo, o Raspberry Pi 3. Estas especificações, fundamentadas no anexo A.1.2, foram tidas em conta para a configuração da *toolchain*.

A tabela 4.1, apresenta algumas dessas variáveis, bem como a configuração usada em função do sistema *target*.

Tabela 4.1: Variáveis definidas na *toolchain* para processo de *cross-compilation*

Variável	Configuração
CMAKE_SYSTEM_NAME	Linux
CMAKE_SYSTEM_VERSION	3.6.1
CMAKE_SYSTEM_PROCESSOR	ARMv7
CMAKE_C_COMPILER	/.../arm-linux-gnueabihf-gcc
CMAKE_CXX_COMPILER	/.../arm-linux-gnueabihf-g++
CMAKE_FIND_ROOT_PATH	/usr/bin/arm-linux-gnueabihf
FLOAT-ABI	hard
FPU	vfpv3
TUNE	cortex-a53
BYTE-ORDER	little-endian

Após a definição destas *flags* (variáveis), procedeu-se à compilação do código fonte das referidas bibliotecas, que foram, posteriormente, transferidas para o Raspberry Pi. O tempo associado à compilação segundo este método, resultou na duração apresentada na tabela 4.2.

Tabela 4.2: Tempo de duração do processo de *cross-compilation* para cada biblioteca

Biblioteca	Duração *Cross-Compilation* (mm.ss)
Point Cloud Library	53m.17s
Open Natural Interaction	9m.3s

Posto isto, estavam reunidas as condições para o uso destas bibliotecas, que, para o efeito, foram transferidas para o Raspberry, no qual a sua utilização não foi bem sucedida. Apesar de todas as bibliotecas requisitadas se encontrarem presentes na plataforma *target*, o sistema operativo reclamava a ausência de certas dependências subjacentes às mesmas. (Do conhecimento de outros relatos,) a razão que poderá estar na origem desta falha, está associada ao facto de cada plataforma ter características muito específicas, pelo que a configuração da *toolchain*, se feita impropriamente, origina casos como este.

Para além disso, o resultado do processo de *cross-compilation*, quando transferido para o Raspberry, pode causar conflito com as bibliotecas já presentes neste. Este conflito deve-se a incompatibilidade entre versões das mesmas.

4.1.2 Compilação nativa

A compilação nativa carateriza-se por efetuar a compilação de código, na plataforma para a qual este vai ser usado, isto é, o sistema *build* e o sistema *target* dizem respeito ao mesmo computador.

Ainda que, tal como foi dito anteriormente, o código fonte associado a uma biblioteca seja transversal a qualquer plataforma, este facto não dispensa um conjunto de configurações, associada ao tipo de arquitetura a usar. Um compilador nem sempre tem a capacidade de detetar automaticamente as caraterísticas da plataforma na qual corre.

Deste modo, definiu-se um conjunto de variáveis com vista à compilação do código associado a cada biblioteca, enumeradas na tabela 4.3.

Tabela 4.3: Variáveis definidas para processo de compilação nativa

Variável	Configuração
ARCH	armv7-a
TUNE	cortex-a53
FLOAT-ABI	hard
FPU	vfpv3
BYTE-ORDER	little-endian

Como referido anteriormente, arquiteturas como a do Raspberry possuem uma quantidade de memória RAM de 1GB que pode, muitas vezes, ser insuficiente para correr certos programas. Neste caso concreto, dada a complexidade da PCL, associada ao enorme leque de módulos que inclui, faz com que, aquando da compilação de código fonte, a memória RAM se esgote.

A solução adotada para proceder à compilação de código, passou pela aplicação do método de *swap*. Este conceito baseia-se no uso de uma quantidade específica de memória adicional como memória virtual, permitindo assim uma expansão temporária da RAM, de modo a garantir a quantidade de recursos necessário para a compilação de código. Esta memória adicional poderá corresponder a espaço do cartão microSD ou de um disco rígido USB, pelo que esta seleção, bem como a quantidade de memória a adicionar, são caraterísticas a ser definidas no *swap-file*. A quantidade máxima de memória rígida passível de ser usada como memória virtual, corresponde ao dobro da RAM do dispositivo em causa, neste caso, 2GB.

É de salientar que a memória RAM que resulta do método de *swap* tem associado um maior tempo de acesso, no entanto permite contornar o problema de falta de RAM. Seguindo esta metodologia, procedeu-se à expansão da memória virtual de 1GB para 3GB, recorrendo a memória disponível no cartão microSD.

Uma vez concluída a definição das *flags* de compilação e expansão da memória RAM, foi possível compilar o código fonte de ambas as bibliotecas. Numa primeira abordagem, De modo a assegurar o sucesso deste processo, optou-se por recorrer a apenas um *core* do processador pois, caso contrário, isto é, se se tentasse acelerar o processo usando os quatro *cores* que este possui, o microcomputador não teria capacidade de completar este processo.

A duração deste processo para cada biblioteca encontra-se referido na tabela 4.4. Salienta-se o facto de se ter procedido à instalação de ambas as versões da OpenNI, pelo que na tabela se encontra apenas a duração associada à versão usada.

Tabela 4.4: Tempo de duração do processo de compilação nativa para cada biblioteca

Biblioteca	Duração Compilação Nativa (hh.mm.ss)
Point Cloud Library	4h.10m.7s
Open Natural Interaction	19m3s

Finalizada a compilação da PCL e da OpenNI, começou-se por testar um simples bloco de código, que permitia a aquisição de dados do sensor e a sua conversão para o formato de nuvem de pontos, usando, para o efeito, ambas as bibliotecas. Este procedimento foi implementado para o Kinect 1 (OpenNI1) e Asus Xtion Pro Live (OpenNI2).

Conectando o Kinect 1 ao Raspberry, não foi possível a receção de dados, ainda que o microcomputador reconheça a presença do sensor. O problema que está na origem desta falha não se trata de um défice de energia fornecida ao sensor, uma vez que esta não é fornecida pelo Raspberry, mas sim externamente. Assim, o facto de o microcomputador não conseguir aceder ao sensor para aquisição de dados, pode ser explicado pela biblioteca que o suporta, OpenNI1, que, uma vez descontinuada, o código associado a esta encontra-se desatualizado, o que cria incompatibilidades com sistemas operativos mais recentes como é o caso do Raspbian.

Por outro lado, usando o Asus Xtion Pro Live, o acesso ao sensor foi inicialmente negado, o que se resolveu com recurso a um *powered* USB *hub*. Neste caso, a versão da OpenNI utilizada ainda se encontra disponível, o que remete para a garantia da sua operacionalidade em dispositivos como o microcomputador selecionado. Estes resultados obtidos no Raspberry, justificam a seleção do Asus Xtion Pro Live, referida em 3.1.1.

4.2 Conclusões

Este capítulo apresentou duas formas distintas de instalar bibliotecas, a partir do seu código fonte, no Raspberry Pi e respetivos resultados.

De um modo geral, ambas as técnicas possuem um processo de compilação idêntico, sendo o que as distingue a plataforma na qual se compila o código fonte.

A escolha do método de compilação nativa assentou na simplicidade deste tipo de abordagem. Pela definição de um conjunto simples de variáveis, foi possível garantir a criação de código executável no Raspberry. Dada a eficácia e confiabilidade deste método, a duração a si associada não traduziu um obstáculo.

A rapidez normalmente associada ao processo de *cross-compilation*, acaba por não compensar, dada a complexidade desta metodologia. Para além disso, nem sempre é garantida a criação de código executável compatível com a plataforma *target*. Isto é, as bibliotecas ao serem compiladas

externamente, quando transferidas para o Raspberry podem não funcionar, devido a incompatibilidades entre estas e dependências previamente instaladas.

Assim, optando pelo método de compilação nativa foi possível obter o código executável das mesmas, sem conflitos inerentes. A única incompatibilidade encontrada, diz respeito à instalação de uma versão desatualizada da biblioteca OpenNI1, a qual não foi possível usar no Raspberry Pi.

Capítulo 5

Software de Aplicação

Como referido na secção 1.3, esta dissertação pretende replicar tarefas tipicamente realizadas por manipuladores na indústria, mais especificamente, operações *pick-and-place*. Para tal, é necessário reconhecer o tipo de peça a manipular, pelo que este processo compreende um conjunto de etapas, por sua vez, descritas no presente capítulo. Esta descrição será acompanhada dos resultados obtidos para cada fase.

Nesta abordagem, optou-se por efetuar o reconhecimento de um cilindro e atuação do manipulador para se deslocar até este. Neste âmbito, e uma vez que se trata de um estudo, pode afirmar-se que objetivo inicial desta aplicação foi o de criar um programa capaz de reconhecer o objeto em questão, fazendo-o de forma simples, de modo a conferir um baixo peso computacional. À medida que se foram obtendo resultados, e em função destes, a complexidade do programa foi aumentada.

Assim, tendo em conta a aplicação pretendida, foi possível dividir este processo em três etapas distintas: aquisição de dados, presente na secção 5.1, o processamento destes dados com vista à deteção de um cilindro e a sua localização, em 5.2, e, por fim, o processo de atuação do manipulador e respetiva garra, no subcapítulo 5.3. Estas etapas encontram-se representadas no fluxograma da figura 5.1.

De uma forma geral, a fase de aquisição compreende o acesso ao sensor, receção de dados e conversão dos mesmos para formato nuvem de pontos (PCD). Adquirida a nuvem, segue-se o seu processamento que inicia com uma fase de filtragem, feita segundo os eixos do referencial do sensor, e de redução de resolução. Seguidamente, segmenta-se e extrai-se da nuvem a mesa, na qual se encontram os objetos, que, por sua vez, são divididos em diferentes conjuntos de pontos, denominados de *clusters*. Cada *cluster* é processado com o intuito de encontrar um cilindro que, após segmentado permite o cálculo das suas coordenadas, segundo o ponto de vista do sensor. Já na fase de manipulação, estas coordenadas são convertidas para o referencial do robô, e enviadas para o controlador de modo a mover o manipulador e garra até ao cilindro.

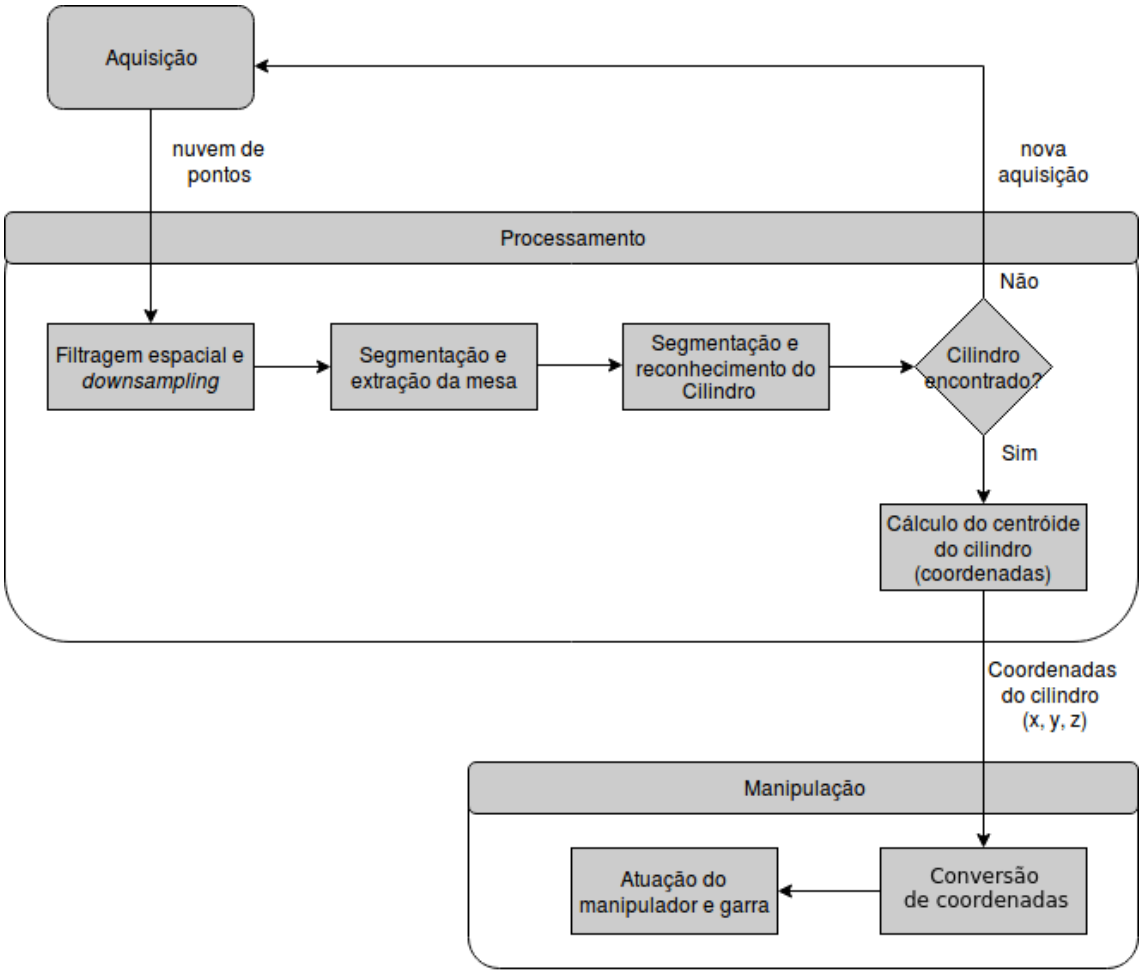

Figura 5.1: Fluxograma representativo do *software* de aplicação

5.1 Aquisição

O sensor selecionado foi o Asus Xtion Pro Live, o qual se baseia no princípio de luz estruturada, referido em 2.1.1.2. O seu método de funcionamento compreende a emissão de um padrão *laser*, pelo que a sua deformação permite apurar a profundidade de cada ponto no cenário abrangido pelo sensor.

Em primeiro lugar, surge a necessidade de calibrar ambas as câmaras do sensor, RGB e Infravermelha, sendo que este processo passa pelo uso de um padrão xadrez (*chessboard*). A calibração dos parâmetros intrínsecos de cada câmara é feita pela deteção desse *chessboard* e serve para garantir a correta leitura dos pontos face à sua posição real.

Neste contexto, foi usado um padrão xadrez de 8x5 vértices de interseção de quadrados pretos, cada qual possuindo lados de comprimento de 10.8cm. Recorreu-se a uma aplicação ROS (*Robot Operating System*), sendo que o processo de calibração para a câmara RGB e câmara infravermelha se encontra ilustrado nas figuras 5.2 e 5.3, respetivamente.

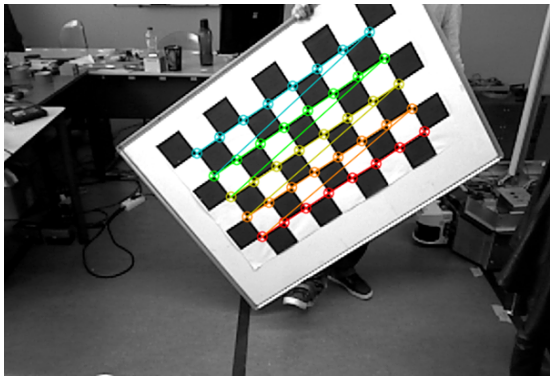

Figura 5.2: Processo de calibração da câmara RGB por meio de um padrão xadrez

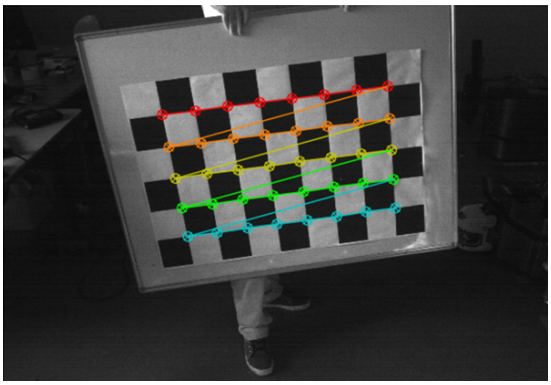

Figura 5.3: Processo de calibração da câmara infravermelha por meio de um padrão xadrez

A fase de aquisição, ilustrada na figura 5.4, começa por criar uma interface com o sensor, recorrendo à função *OpenNI2Grabber*. Uma vez estabelecida a conexão com o Asus Xtion, dá-se a aquisição de informação 3D. Os dados do sensor são convertidos para o formato de nuvem de pontos, recorrendo à biblioteca *Open Natural Interaction* (OpenNI) em conjunto com a *Point Cloud Library* (PCL). A resolução da nuvem de pontos corresponde ao valor definido por omissão nas configurações do sensor, mais especificamente, 640×480 *pixels*.

Tendo em conta que se pretende fazer uma gestão otimizada de recursos no Raspberry Pi, o sensor apenas é ligado quando se pretende obter uma nuvem de pontos para processamento.

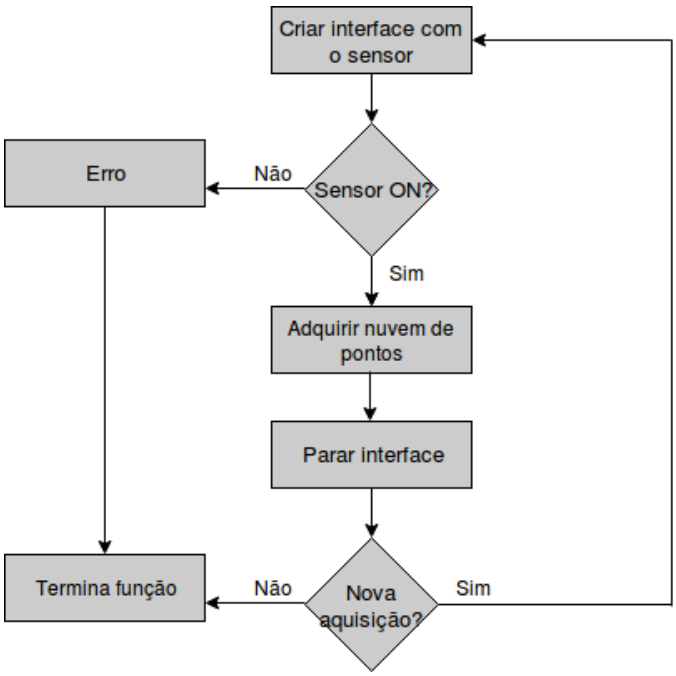

Figura 5.4: Fluxograma representativo da fase de aquisição

Existem diferentes tipos de nuvens de pontos, por exemplo, *PointXYZ*, na figura 5.5, que possui apenas os dados relativos à profundidade de cada *pixel* da imagem, e *PointXYZRGB*, ilustrada

na figura 5.6 que, para além da distância, fornece a informação da cor. As figuras 5.5 e 5.6 representam uma parte do campo de visão do sensor, perante a bancada de testes, representando os dois tipos de nuvens referidos. Para a aplicação inicial, optou-se por adquirir nuvens do tipo *PointXYZ*, pelo que se trabalhou somente com a informação da profundidade.

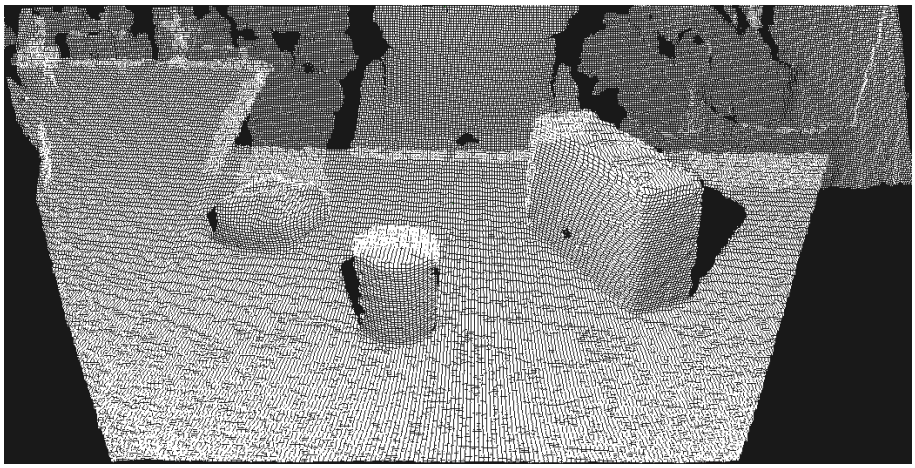

Figura 5.5: Nuvem de pontos *depth*, do tipo *PointXYZ*

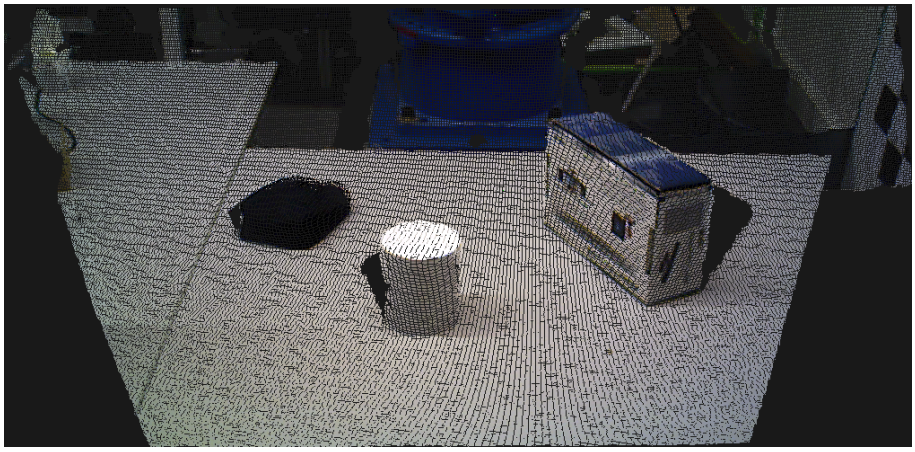

Figura 5.6: Nuvem de pontos RGB, do tipo *PointXYZRGB*

5.2 Processamento

Após a aquisição da nuvem de pontos, segue-se o seu processamento de forma a alcançar o objetivo pretendido, o de reconhecer um cilindro. Esta etapa compreende quatro fases sequenciais, mais propriamente, a filtragem da nuvem adquirida, em 5.2.1, bem como a extração dos elementos que a compõem e o reconhecimento do cilindro, em 5.2.2 e 5.2.3, respetivamente, e, por último, cálculo das coordenadas do mesmo, em 5.2.4.

5.2.1 Filtragem

Usando um sensor como o Asus ou o Kinect é possível produzir uma nuvem de 30720 pontos (640×480), o que corresponde a uma grande quantidade de informação. O número de pontos presentes numa nuvem influencia diretamente a velocidade de processamento associada a esta, tendo em conta que as operações são realizadas em cada ponto. Assim, existe um conjunto de técnicas de otimização que podem ser aplicadas, de forma a acelerar o processo seguinte.

Para otimizar o processo, selecionou-se uma região de interesse, uma vez que o sensor possui uma localização fixa em relação à mesa, na qual estão depositados os objetos, e o seu referencial é conhecido. Uma região de interesse traduz um volume de espaço, segundo o qual se pretende aplicar o processamento, pelo que neste caso corresponde à mesa e objetos sobrepostos, eliminando da nuvem os restantes pontos. Assim, recorreu-se à função *PassThrough*, que corresponde a um filtro passa-banda capaz de remover qualquer ponto cujos valores não pertençam a um determinado intervalo. O filtro em questão foi aplicado aos pontos da nuvem segundo os três eixos, respetivamente, x, y e z , usando os seguintes intervalos: [-0.2 , 1.1]m, [-0.3 , 0.5]m e [0.3 , 1.0]m, respetivamente. Estes valores traduzem a região no espaço 3D definida pela mesa e objetos.

Apesar de esta filtragem reduzir o número de pontos numa percentagem considerável de aproximadamente 50%, a quantidade de pontos é ainda elevada. Assim, após a filtragem segundo os eixos, optou-se por aplicar a técnica de *downsampling*, que corresponde a uma redução do número de pontos da nuvem, usando uma abordagem de grelha *voxel*.

A classe *VoxelGrid* cria uma grelha *voxel* 3D a partir da nuvem de entrada, sendo que cada *voxel*, ou caixa, corresponde a um conjunto de pontos que, após a aplicação deste método, serão aproximados aos seu centróide[1]. Poder-se-ia ter aproximado cada *voxel* ao seu centro de gravidade, o que tornaria o processo mais rápido, no entanto, a abordagem adotada representa a superfície de forma mais precisa. Neste âmbito, o número de pontos foi reduzido usando um tamanho de folha, *leaf size*, para a grelha de *voxel* de 1cm. Este valor foi selecionado de forma a garantir uma diminuição do peso computacional da nuvem, sem que haja perda de informação.

A aplicação desta estrutura de dados à nuvem de pontos, para além de diminuir a resolução através da partição do espaço 3D, permite a remoção de ruído associado a medições erróneas que advêm da aquisição.

A figura 5.7 representa a nuvem de pontos que resultou do processo de filtragem, contendo apenas a mesa e objetos sobrepostos. Para além disso, é visível a redução do número de pontos, resultantes do método de *downsampling*.

[1] Ponto cujas coordenadas correspondem à média de todos os pontos que pertencem ao *voxel*.

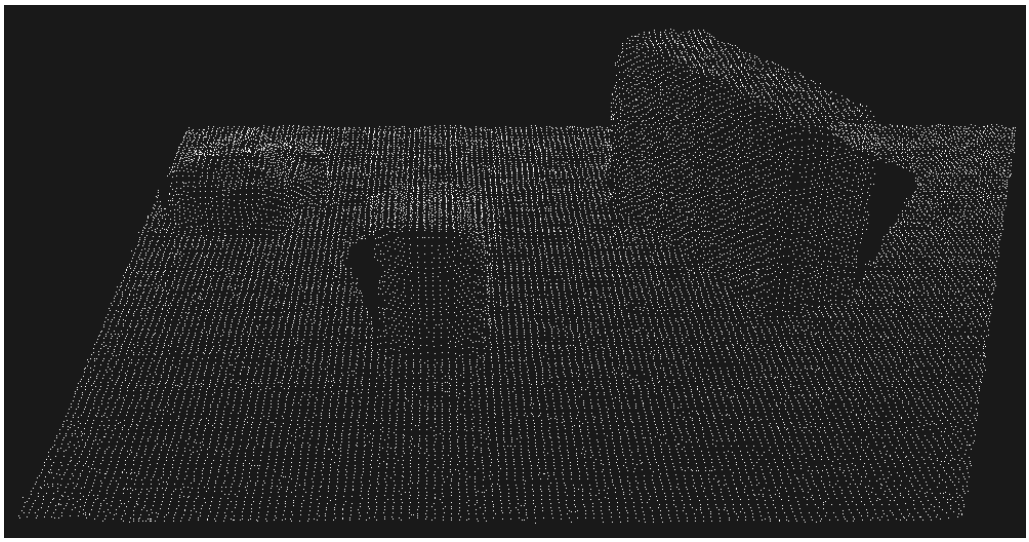

Figura 5.7: Nuvem de pontos após filtragem segundo os eixos (x,y,z) e *downsampling*

Assim, de uma forma geral, a fase de filtragem encontra-se representada na figura 5.8.

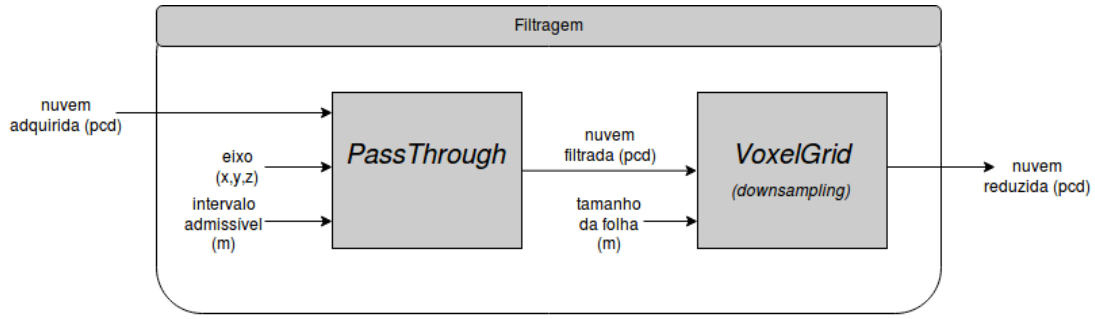

Figura 5.8: Diagrama representativo da fase de filtragem

5.2.2 Extração

Após a fase de filtragem seguem-se as etapas que permitiram o reconhecimento do cilindro.

Para a segmentação de um determinado objeto, é pertinente a subdivisão da nuvem de pontos em diferentes aglomerados, isto é, *clusters*. O processo de *clustering* é frequentemente usado para dividir a nuvem em conjuntos consistentes, suscetíveis de serem processados independentemente.

Neste âmbito, começou-se por segmentar a mesa, com vista à sua extração, de modo a isolar os objetos. Para o efeito, recorreu-se à função *SACSegmentation* (*SAmple Consensus Segmentation*), na qual, a partir de um modelo conhecido e segundo um método definido, é possível encontrar na nuvem os pontos que pertencem a esse mesmo modelo. No código desenvolvido, usou-se o método RANSAC, referido em 2.2.1.1, em conjunto com um modelo planar, de forma a encontrar a mesa.

Esta função recebe também como parâmetros de entrada a nuvem de pontos e um valor de *threshold*, que define o máximo desvio aceite para dados com ruído. Aleatoriamente, o algoritmo

probabilístico RANSAC seleciona um subconjunto de pontos da nuvem recebida. De seguida, dá-se o processamento do modelo planar estimado em função dos parâmetros fornecidos, usando apenas os dados desse subconjunto. Este método calcula os elementos de todo o conjunto de dados que pertencem ao modelo definido, fornecendo os coeficientes obtidos no modelo estimado. Assim, este cálculo retorna os *inliers* escritos no formato de *PointIndices*, bem como um conjunto de quatro coeficientes apurados, definidos no formato *ModelCoefficients*.

Os índices dizem respeito a uma lista de alguns pontos que constituem uma nuvem, não sendo estes os pontos, propriamente ditos, mas sim do seu índice na nuvem. Este formato de dados é usado por muitos algoritmos e será usado para o processo de extração. Por outro lado, os coeficientes apresentam, em função do conjunto de *inliers*, os parâmetros do modelo estimado para a mesa, na forma da equação 5.1.

$$ax + by + cz + d = 0 \tag{5.1}$$

Assim, um ponto será considerado um *outlier* se não pertencer ao modelo indicado tendo em conta as condições iniciais, dentro de um *threshold*, neste caso de 1cm. Este processo é repetido de forma aleatória e iterativa, num máximo estipulado de cem iterações, até que sejam encontrados pontos suficientes, que possam ser classificados como parte do modelo indicado.

Encontrados os *inliers* (índices) e coeficientes capazes de se aproximar do modelo indicado, segue-se a sua extração, através da função *ExtractIndices*. Este método permite extrair de uma nuvem um conjuntos de índices, neste caso, os que definem o plano da mesa, resultando deste processo a nuvem que contém a mesa, representada na figura 5.9. A imagem 5.10, por sua vez, representa de outra forma o resultado desta operação, demonstrando que este processo foi bem sucedido pela clara distinção que há entre a mesa e o conjunto de objetos.

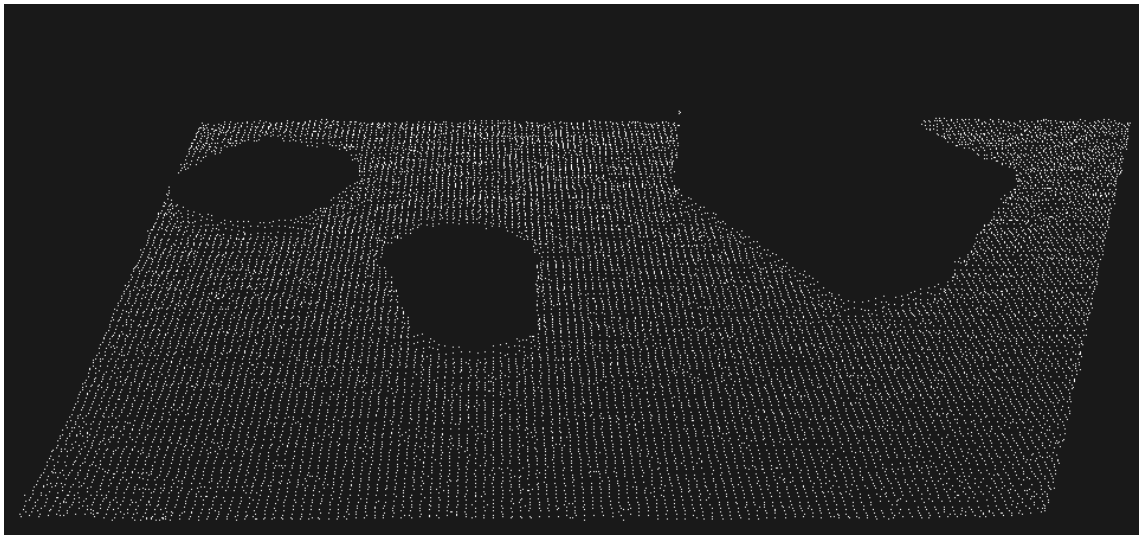

Figura 5.9: Nuvem de pontos resultante da segmentação da mesa

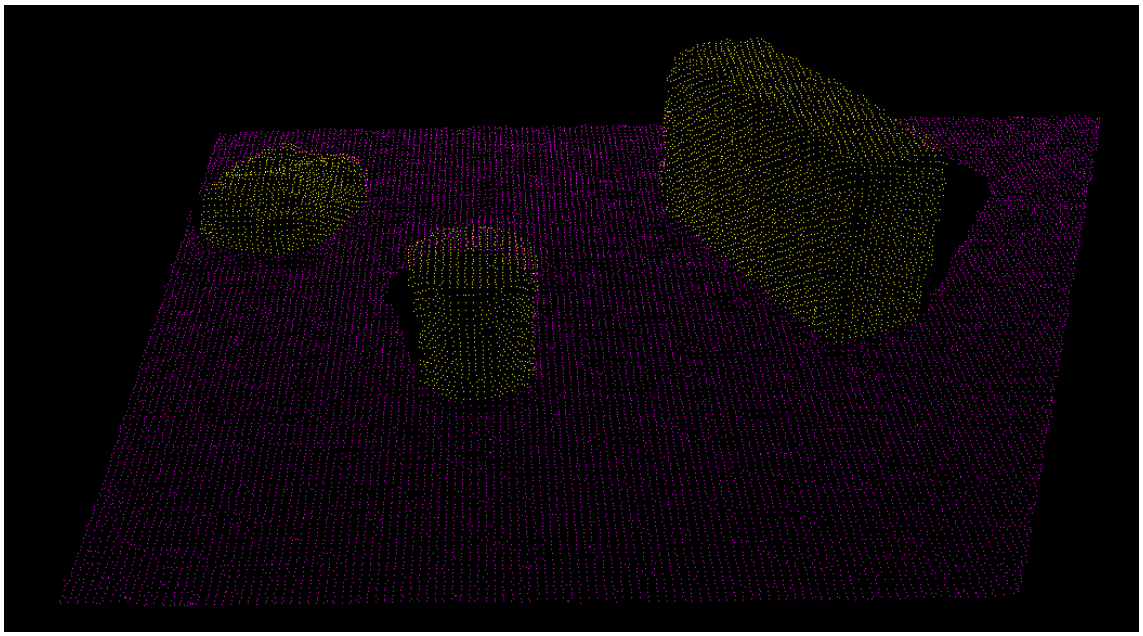

Figura 5.10: Nuvem de pontos que concatena o resultado da segmentação da mesa com os restantes pontos que constituem a nuvem (objetos)

De forma a obter uma nuvem somente com os objetos, a nuvem resultante da segmentação da mesa, 5.9, foi subtraída à nuvem filtrada, 5.7, originando a nuvem presente na figura 5.11. Para esta etapa recorreu-se novamente à rotina *ExtractIndices*.

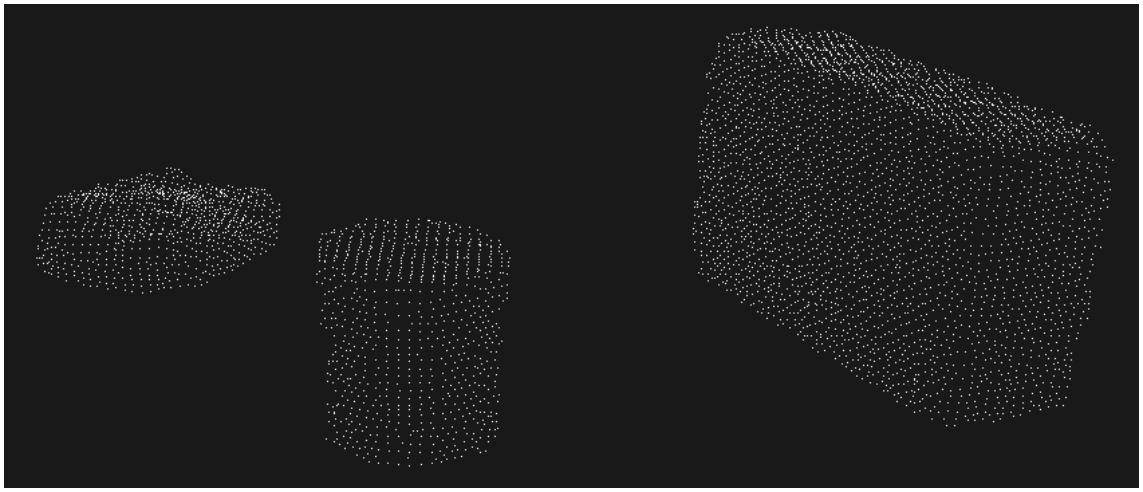

Figura 5.11: Nuvem de pontos apenas com os objetos

Posto isto, torna-se agora necessário separar os objetos entre si. Neste âmbito, insere-se o conceito de *clustering* associado à função *EuclideanClusterExtraction*. Este método permite dividir uma nuvem não organizada, em partes distintas, através da pesquisa de pontos por meio de uma estrutura de dados.

Nos algoritmos de pesquisa de pontos vizinhos é comum o uso de estruturas de dados, de

modo a tornarem o processo de pesquisa mais eficiente. Para esta técnica de *clustering* selecionou-se o tipo *KdTree*. Esta estrutura de dados armazena um conjunto de pontos de k dimensões numa estrutura em árvore que, em cada nível, se divide ao longo de uma dimensão específica, recorrendo a um hiperplano perpendicular a um dos eixos. Tendo em conta o ponto que se pretende pesquisar, e os respetivos vizinhos, a raiz da árvore de pesquisa corresponde ao ponto mais distante do ponto que está a ser pesquisado, numa esfera de raio definido por este comprimento, contendo, por isso, todos os pontos candidatos a vizinhos. Para a sub-divisão da árvore, são apurados dois pontos, respetivamente, o mais próximo à esquerda e à direita da raiz, segundo um determinado eixo. Terminada a primeira ramificação, a seguinte é feita com hiperplanos perpendiculares ao eixo considerado, aplicando a mesma regra. A ramificação da árvore termina quando se esgotam os pontos. Posteriormente, a pesquisa é feita ao longo dos ramos da árvore de pontos, calculando a distância do ponto pretendido a cada ramo. Se o círculo gerado não contiver nenhum outro ponto, significa que está encontrado o vizinho mais próximo. [33]

Concluindo, o procedimento explicado acima, intersetado com um conjunto de parâmetros de entrada, permite dividir uma nuvem em *clusters*. Para o efeito, é necessário definir um intervalo de dimensões no qual cada *cluster* terá de estar contido, neste caso estipulou-se um tamanho mínimo e máximo. Para além disso, selecionou-se um raio de 2 cm para a definição de esfera no processo de pesquisa de pontos, isto quer dizer que, para dois objetos serem distinguidos em dois *clusters* têm de estar a uma distância superior a 2cm. Esta função retorna um vetor de dimensão variável conforme o número de objetos na mesa, pelo que neste estão alocados, separadamente, os índices de cada *cluster*.

Resumidamente, a fase de extração encontra-se ilustrada no diagrama da figura 5.12.

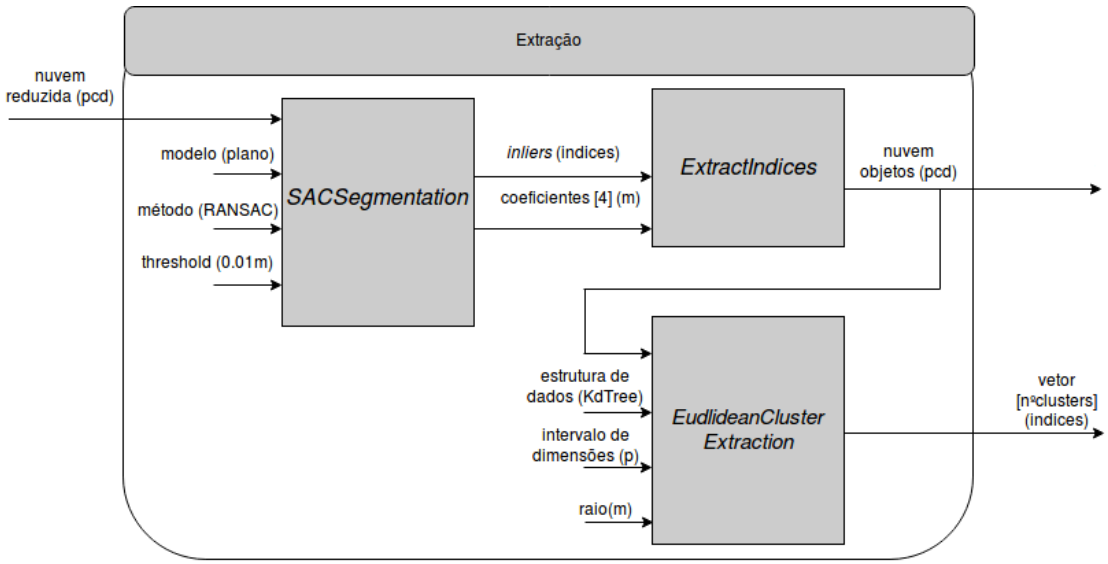

Figura 5.12: Diagrama representativo da fase de extração

5.2.3 Reconhecimento

Calculados os índices, a etapa seguinte consiste em percorrer cada *cluster*, objetivando-se a conversão do conjunto de índices que o define, para o formato de nuvem de pontos representativa do objeto. O resultado desta operação encontra-se representado na figura 5.13 que concatena três nuvens.

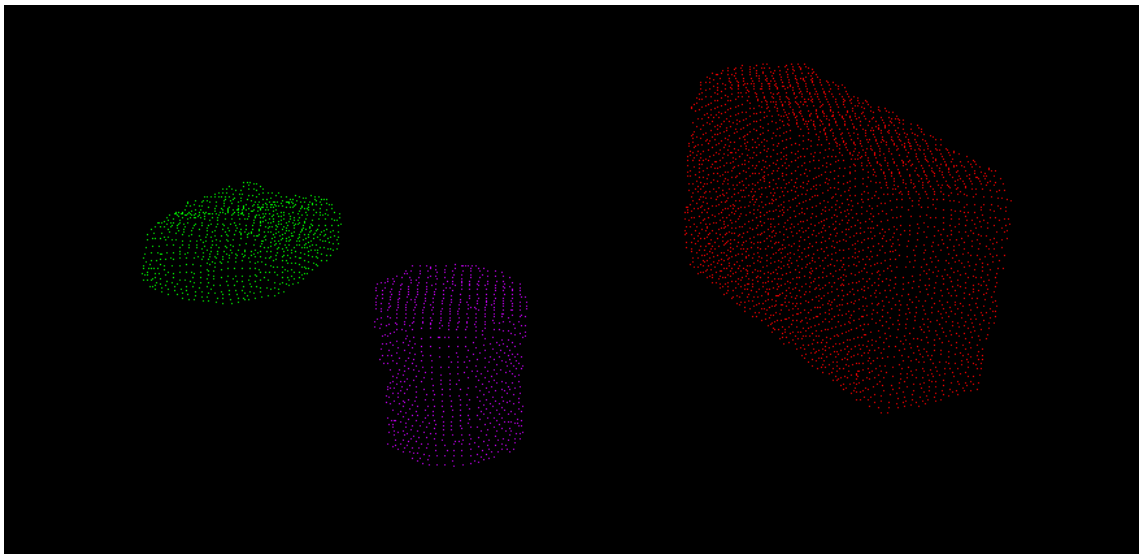

Figura 5.13: Extração dos objetos sobrepostos na mesa

Concluída a extração dos objetos, torna-se possível o processo de segmentação para o reconhecimento de um cilindro, caso exista algum sobre a mesa. Para esta fase, optou-se por recorrer a características específicas do objeto, mais propriamente, as normais à superfície.

As normais à superfície, traduzem uma propriedade relevante, também designada *feature*, de uma superfície geométrica, pelo que correspondem ao vetor perpendicular à superfície num dado ponto. Neste sentido, para cada *cluster*, aplicou-se a função *NormalEstimation* para calcular as normais dos pontos do objeto.

Para cada ponto de uma nuvem, este método encontra os pontos mais próximos, e em função dessa pesquisa calcula a normal desse ponto. Por fim, verifica se a normal se encontra orientada de forma consistente, tendo em conta o ponto de vista do sensor, caso contrário, é mudada a sua orientação.

Para estimar a normal à superfície de um determinado ponto é necessário considerar a sua vizinhança de pontos, numa proporção definida. Esta quantidade de pontos pode ter por base a definição de k pontos vizinhos ou pela seleção de um raio, que terá em conta todos os vizinhos dentro dessa mesma distância. Neste caso, optou-se por efetuar a pesquisa através de um raio previamente selecionado. A escolha do valor deste parâmetro é de extrema importância, uma vez que influenciará diretamente os resultados da segmentação que se segue.

Assim, procedeu-se à análise do melhor valor a selecionar, pelo estudo de um conjunto de raios e avaliação de resultados do respetivo cálculo das normais para um cilindro. Assim avaliaram-se

os seguintes raios de pesquisa: 0.5cm, 1.5cm, 5cm e 10cm, respetivamente, figura 5.14, 5.15, 5.16 e 5.17.

Para uma melhor visualização das normais, o cilindro encontra-se posicionado lateralmente de forma a serem percetíveis os seus limites e representação das normais na fronteira que separa a base superior do cilindro com a sua lateral.

Neste caso, um bom cálculo das normais será aquele capaz de distinguir duas superfícies distintas, a base superior do cilindro e a lateral do mesmo. Mais concretamente, o ângulo compreendido entre as normais destas superfícies terá de ser o mais próximo possível de 90^{o}.

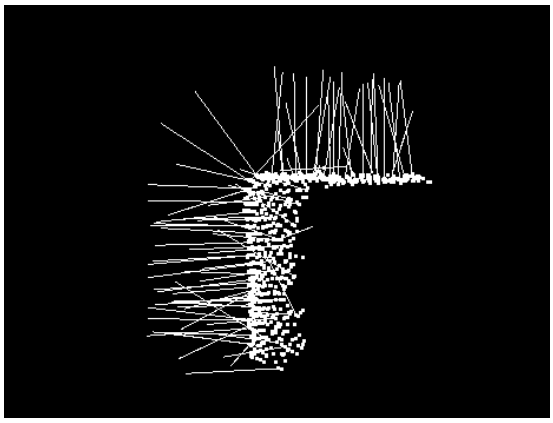

Figura 5.14: Representação do resultado cálculo das normais usando um raio de pesquisa de 0.5cm

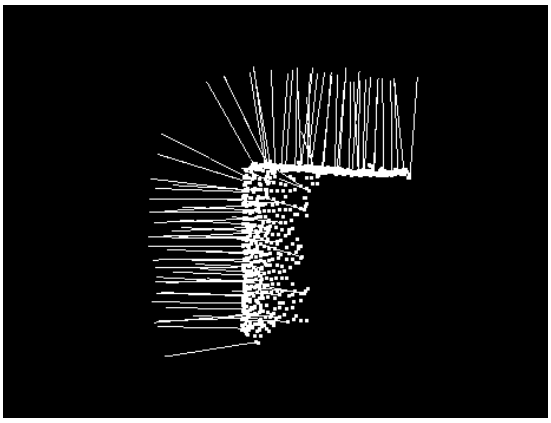

Figura 5.15: Representação do resultado cálculo das normais usando um raio de pesquisa de 1.5cm

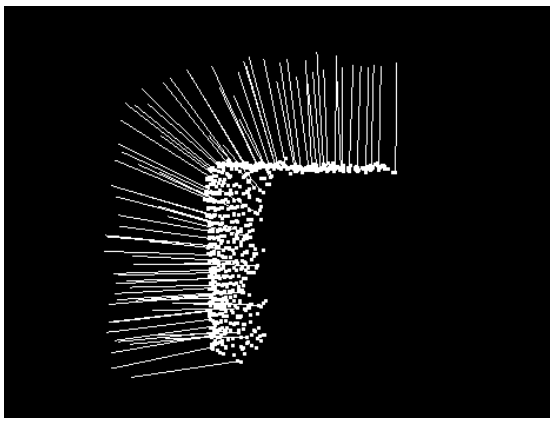

Figura 5.16: Representação do resultado cálculo das normais usando um raio de pesquisa de 5cm

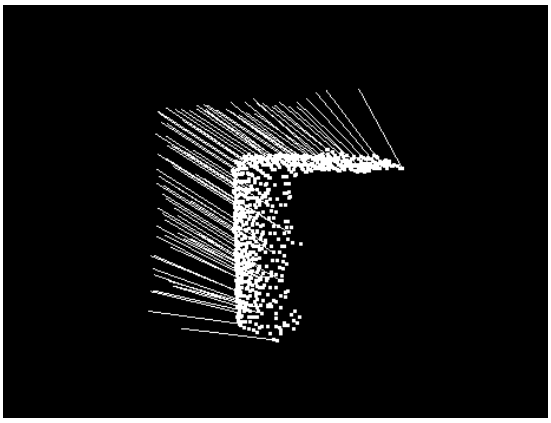

Figura 5.17: Representação do resultado cálculo das normais usando um raio de pesquisa de 10cm

Pela observação das figuras, o raio selecionado que mais se aproxima da representação explicada acima, é o de 1.5cm, imagem 5.15, uma vez que as normais são aproximadamente perpendiculares à superfície na qual se insere o respetivo ponto, mesmo nas zonas de transição entre superfícies.

Assim, conclui-se que para um melhor resultado, o raio selecionado deverá ser reduzido. No entanto, se este for demasiado pequeno, como é o caso do raio de 0.5cm, tendo em conta que este é de ordem bastante reduzida, consequentemente, o número de vizinhos tidos em conta é também menor. Este facto pode originar cálculos errados, o que se verifica pela visualização da figura 5.14, na qual muitas das normais não são perpendiculares à superfície onde o ponto está alocado.

Por outro lado, a escolha de um raio elevado permite abranger um maior número de pontos vizinhos, o que, em zonas de transição de superfícies, desencadeia o cálculo de falsas normais. Em superfícies adjacentes, as normais calculadas corresponderão a uma representação distorcida da realidade, como é possível verificar nas figuras 5.16 e 5.17. Verificando-se que, à medida que o raio é aumentado, esta distorção aumenta significativamente.

Finalizado este estudo, selecionou-se um raio de 1.5cm para o cálculo das normais à superfície de cada *cluster*, cujo resultado será o *input* para o procedimento que se segue.

Calculadas as *features*, neste caso, os vetores normais que caraterizam um determinado objeto, recorreu-se ao método *SACSegmentationFromNormals* para apurar se o respetivo se trata de um cilindro ou não. Para tal, foi usado, novamente, o método RANSAC, porém, desta vez, o modelo a comparar corresponde a um cilindro.

Neste contexto, conhecendo de antemão o raio do cilindro (4.5cm), definiu-se um desvio aceitável de 0.5cm para o raio do modelo estimado. Para além disso, estipulou-se uma distância *threshold* de 0.3cm, que define o quão próximo um *inlier* tem de estar do modelo para ser aceite como tal. O mesmo valor foi atribuído ao peso das normais, uma vez que traduziu melhores resultados na rejeição de objetos. Para além disso, definiu-se um máximo de 100 iterações para a execução do método RANSAC.

Assim, segundo este conjunto de parâmetros de entrada, o algoritmo calculará os índices do *cluster* que se aproximam do modelo definido. Tendo em conta que o método RANSAC assume à partida que a nuvem em estudo possui *inliers*, através de sucessivas iterações, o seu resultado será sempre um conjunto de coeficientes e *inliers*. Assim, a distinção entre um objeto e um cilindro é feita pela análise da razão entre o número de pontos inicial do *cluster* e o número de *inliers* do modelo estimado. A equação do modelo do cilindro usada por método, carateriza-se por definir apenas a zona tubular do cilindro, isto é, corresponde a um cilindro desprovido de bases.

Deste modo, o modelo estimado da segmentação do *cluster* que corresponde a um cilindro, apenas corresponderá à sua estrutura tubular, sendo ignorada a sua base superior. Esta explicação encontra-se concretizada na figura 5.18, na qual é possível observar, segundo um ponto de vista frontal e lateral, o resultado da segmentação da face tubular do cilindro, representada a amarelo.

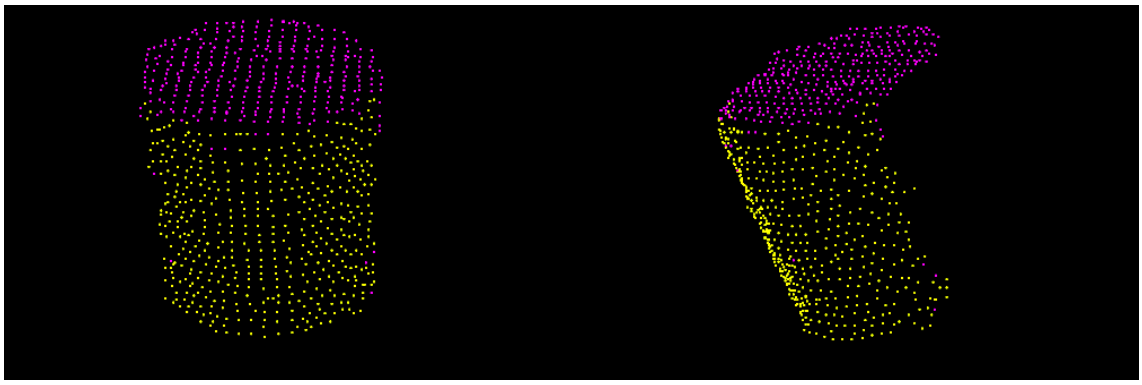

Figura 5.18: Resultado da segmentação do cilindro do ponto de vista frontal, à esquerda, e lateral, à direita

Os resultados obtidos da segmentação do cilindro foram bastante satisfatórios, no sentido em que praticamente todos os pontos, que constituem a zona tubular do cilindro, foram aceites como *inliers* do modelo, o que vem provar a correta seleção do raio de pesquisa, para o cálculo das normais à superfície. Existe um conjunto muito reduzido de pontos do tubo que não foram aceites, o que se deve ao facto de estes se localizarem em zonas de fronteira, nas quais o cálculo das normais é mais propício a originar vetores erróneos.

De igual forma, este algoritmo foi aplicado aos restantes *clusters*, isto é, um estojo e uma caixa, sendo que o seu resultado se encontra ilustrado nas figuras 5.19 e 5.20, representado a amarelo. Como é possível verificar, em ambos os *clusters* verifica-se uma reduzida quantidade de pontos passíveis de pertencer ao modelo do cilindro.

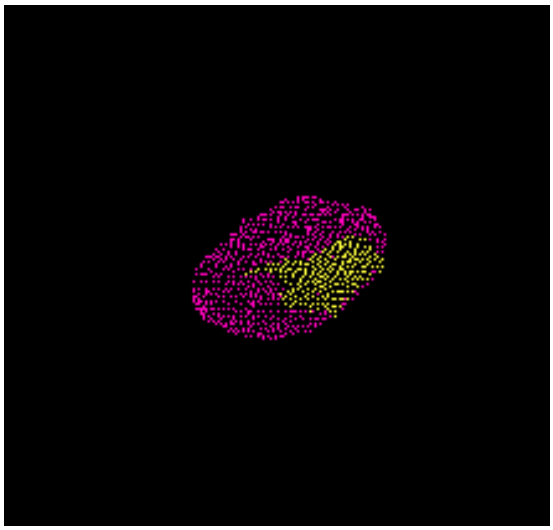

Figura 5.19: Resultado da segmentação do modelo de um cilindro num estojo

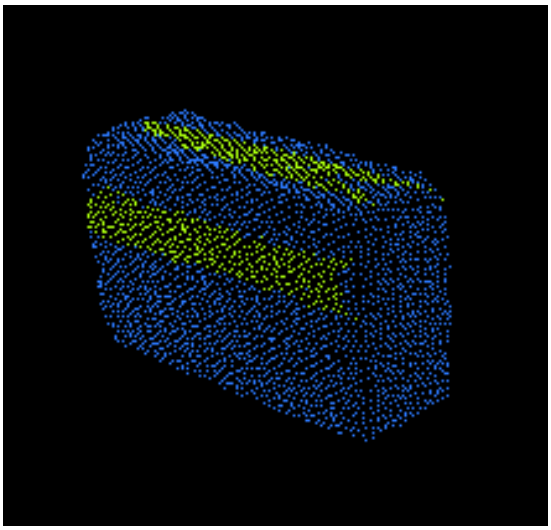

Figura 5.20: Resultado da segmentação do modelo de um cilindro numa caixa

O caso de estudo usado para explicar o princípio de funcionamento do *software* de aplicação, começou por usar três objetos para provar a eficácia deste, respetivamente, uma caixa, um estojo de aspeto retangular, sem arestas definidas, e um cilindro branco. Assim, no seguimento do que

aqui foi apresentado, a tabela 5.1 apresenta para cada objeto o nível de confiança, do resultado do segmentação, em função do número de pontos do respetivo *cluster*.

Como seria de esperar, o cilindro corresponde ao objeto com maior percentagem de confiança. No entanto, tendo em conta os resultados da segmentação para o cilindro, este valor não traduz efetivamente a eficácia do método, tendo sido esta bastante mais elevada. A percentagem de confiança na segmentação de um cilindro, foi definida pela razão entre o número de *inliers* resultantes do processo de segmentação e o número total de pontos do *cluster* inicial. Uma vez que a segmentação do cilindro tem apenas em conta a zona tubular, da segmentação resultam apenas os índices associados aos pontos desta zona, pelo que os restantes, isto é, os que constituem a base superior, são considerados *outliers*, o que explica a percentagem de confiança associada ao cilindro ser relativamente baixa.

Porém, apesar do algoritmo RANSAC se tratar de um método probabilístico, após testes com um diferente número de objetos, chegou-se à conclusão que a percentagem de confiança associada ao cilindro excede sempre os 50%, sendo que esta percentagem relativa a outros objetos nunca ultrapassa os 30%. Desta forma, o critério definido, para aceitar um cilindro, assenta na condição de a sua percentagem de confiança ser superior a 50%. Este *threshold* mostrou-se o valor mais adequado para os testes efetuados, de modo a aceitar o cilindro, quando presente na mesa.

Tabela 5.1: Percentagem de confiança associada a cada objeto, tendo em conta o número de *inliers* resultantes da segmentação do respetivo *cluster*

	Nº de pontos (nuvem *cluster*)	Nº de *inliers* (nuvem segmentada)	Percentagem de confiança(%)
Caixa	3208	397	12%
Estojo	913	242	26%
Cilindro	832	481	57%

É ainda importante referir que o algoritmo de segmentação do cilindro, para além dos *inliers*, retorna um conjunto de sete coeficientes que definem o modelo estimado do cilindro encontrado. A tabela 5.2 identifica estes valores apurados para o cilindro aceite. Deste modo, as três primeiras posições correspondem às coordenadas (x,y,z) de um qualquer ponto situado ao longo do eixo do cilindro, as seguintes três posições dizem respeito às coordenadas do vetor de orientação do eixo do cilindro, com base no ponto anterior, e, por fim, a sétima posição aloca o raio do cilindro.

Tabela 5.2: Identificação dos coeficientes apurados da segmentação do cilindro

Coeficiente	Posição	Valor (cm)
Coordenada X de um ponto no eixo do cilindro	0	-5.5
Coordenada Y de um ponto no eixo do cilindro	1	4.2
Coordenada Z de um ponto no eixo do cilindro	2	70.7
Coordenada X do vetor direção do eixo do cilindro	3	3.2
Coordenada Y do vetor direção do eixo do cilindro	4	-74.1
Coordenada Z do vetor direção do eixo do cilindro	5	-67.0
Raio do cilindro	6	4.5

O conjunto de passos que contempla a fase de reconhecimento de um cilindro encontram-se esquematizados na figura 5.21.

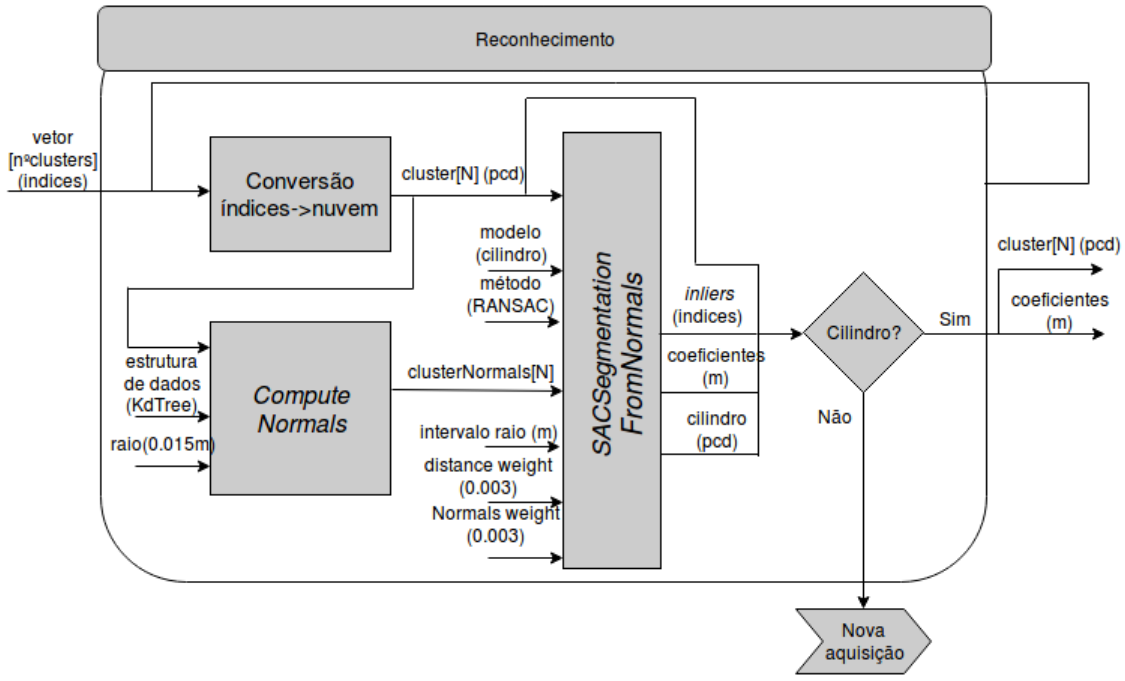

Figura 5.21: Diagrama representativo da fase de reconhecimento do cilindro

5.2.4 Cálculo das coordenadas

Realizada a segmentação, e uma vez aceite o cilindro, a próxima etapa compreende o cálculo das coordenadas do ponto central do eixo que o define, o centróide. Para tal, recorreu-se à função *compute3DCentroid*, capaz de alocar num vetor as coordenadas do centróide de uma nuvem.

Esta função recebe como entrada uma nuvem, pelo que, neste caso, se optou por usar duas nuvens e comparar valores, de forma a avaliar qual a mais conveniente para este cálculo. Deste modo, apurou-se o centróide da nuvem resultante da segmentação, bem como o do correspondente *cluster*, uma vez que o primeiro extrai a base superior do cilindro.

As figuras 5.22 e 5.23 representam, respetivamente, a nuvem *cluster* e a segmentada, segundo dois pontos de vista distintos, para que seja possível verificar que ambas as nuvens não representam um visão geral do objeto. Este facto leva a que o cálculo do seu centróide nunca corresponda ao ponto médio do eixo central do cilindro.

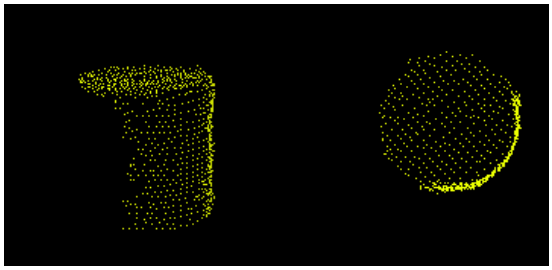

Figura 5.22: Nuvem de pontos representativa do *cluster*, segundo um ponto de vista lateral, à esquerda, e superior, à direita

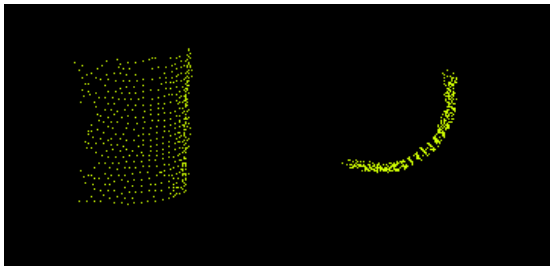

Figura 5.23: Nuvem de pontos representativa da nuvem segmentada, segundo um ponto de vista lateral, à esquerda, e superior, à direita

Assim, para obter melhores resultados, recorreu-se às três primeiras posições do conjunto de coeficientes que resultam da segmentação do cilindro, pelo que os seus valores definem um ponto aleatório no eixo central do mesmo. A tabela 5.3 apresenta o conjunto de valores que será tido em conta para o cálculo das coordenadas do objeto.

Tabela 5.3: Possíveis valores para o centróide do cilindro

	Nuvem *cluster* (cm)	Nuvem segmentada (cm)	Coeficientes do cilindro (cm)
X	-5.2	-5.1	-5.5
Y	2.7	5.0	4.2
Z	66.3	66.4	70.7

Sabendo que os coeficientes apurados do modelo traduzem o eixo central do objeto e conhecendo a sua orientação, é possível extrair destes coeficientes o valor exato das coordenadas X e Z da localização do centro do cilindro, que neste caso são, respetivamente, -5.5cm e 70.7cm. Resta a seleção do valor para a coordenada segundo Y, pelo que agora não se pode recorrer a este parâmetro nos coeficientes, uma vez que não traduz o ponto médio do eixo, mas sim um valor aleatório ao longo deste. Neste sentido, foram testados os valores de Y do centróide de cada uma das nuvens (*cluster* e segmentada), assumindo que correspondem ao ponto médio do eixo central do cilindro. Concluiu-se que a melhor aproximação para este valor pertence à nuvem que representa o *cluster*, isto é, 2.7cm. Desta forma, e neste caso específico, o centróide do cilindro pode ser definido em X, Y e Z por -5.5cm, 2.7cm, 70.7cm, respetivamente.

O diagrama da figura 5.24 apresenta o processo de cálculo da coordenadas do cilindro, explicado na presente subsecção.

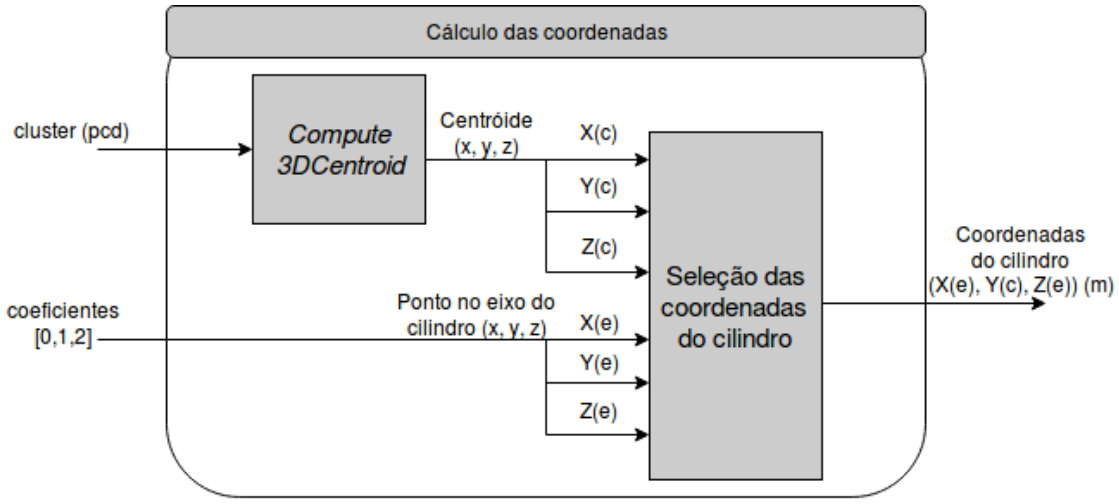

Figura 5.24: Diagrama representativo da fase de cálculo das coordenadas do cilindro

5.3 Manipulação

Após o processamento da nuvem de pontos adquirida, e caso exista um cilindro na bancada de teste, surge a fase de manipulação. Esta etapa compreende a conversão das coordenadas, obtidas e expressas no referencial do sensor, para o referencial do manipulador, explicada no subcapítulo 5.3.1, seguida da atuação propriamente dita, em 5.3.2.

5.3.1 Conversão de coordenadas

Esta etapa inclui a conversão das coordenadas do objeto a partir do referencial do sensor, para o referencial do manipulador. Para tal, recorreu-se a uma matriz de transformação, incluindo rotação e translação, explicada a seguir.

Neste sentido, o referencial *0* traduz o referencial de base do manipulador e o referencial *s* correspondente ao referencial do sensor, ambos ilustrados na figura 5.25.

Assim, a posição do centróide do objeto, expressa no referencial do manipulador pode ser dada pela matriz de transformação H_s^0,

$$H_s^0 = \begin{bmatrix} R_s^0 & d_s^0 \\ 0 & 1 \end{bmatrix} \tag{5.2}$$

na qual $d_s^0 = \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}$ corresponde à posição da origem do referencial do sensor, *s*, em relação ao referencial do manipulador, *0*.

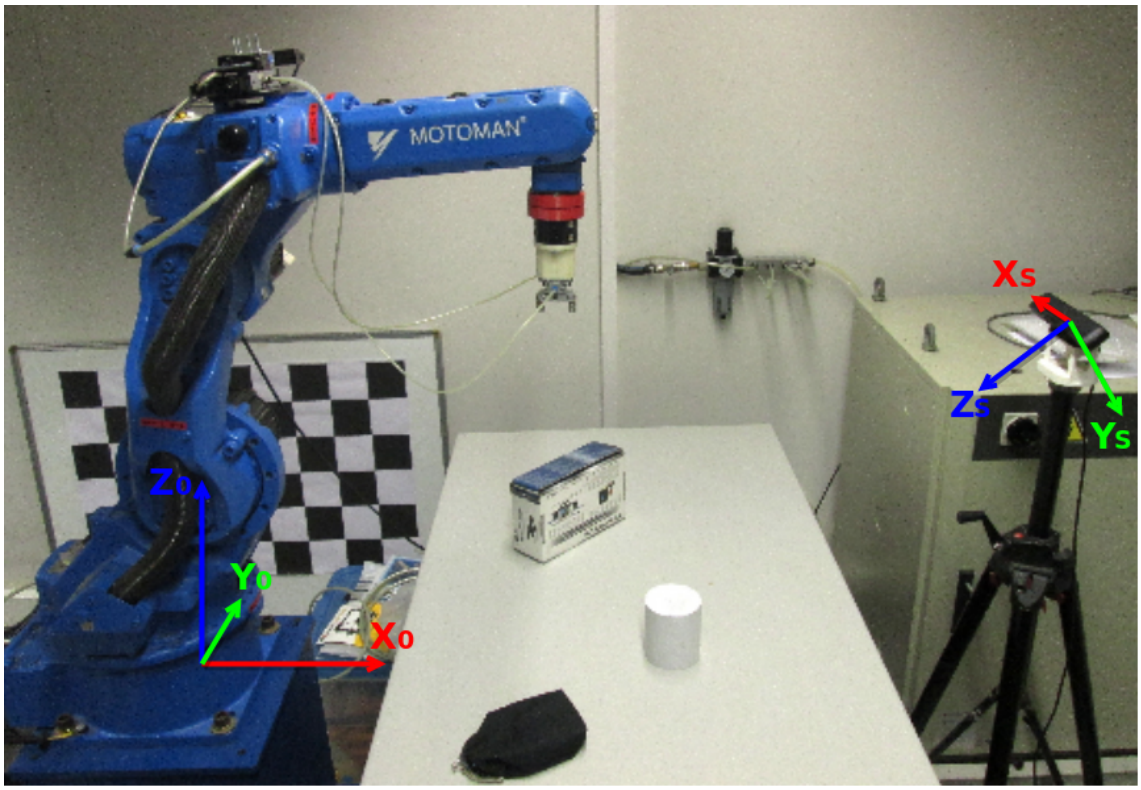

Figura 5.25: Representação do sistema de coordenadas do referencial do manipulador, *0*, e referencial do sensor, *s*

É possível obter-se a representação da rotação do referencial *s* em relação ao referencial *0*, a partir de uma sequência de rotações definidas na equação abaixo,

$$R_s^0 = R_{z,\phi} \times R_{x,\psi} \tag{5.3}$$

e sabendo que,

$$R_{z,\phi} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

$$R_{x,\psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{5.5}$$

é possível obter a matriz de rotação R_s^0, dada por

$$R_s^0 = \begin{bmatrix} \cos\phi & -\sin\phi\cos\psi & \sin\phi\sin\psi \\ \sin\phi & \cos\phi\cos\psi & \cos\phi\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{5.6}$$

.

Posto isto, dada a receção dos valores do centróide do cilindro encontrado, como P_s, as coordenadas do objeto dadas pelo referencial do manipulador, P_0, são dadas pela equação

$$P_0 = H_s^0 \times P_s \tag{5.7}$$

Tendo em conta que as coordenadas do cilindro, P_0, traduzem o seu centroíde, e conhecendo à partida a altura do mesmo, uma vez que se trata de um objeto conhecido, é possível conhecer o ponto para o qual se tem de enviar o manipulador, nos casos em que o cilindro não está deitado. Assim, ao valor de Z das coordenadas do objeto calculadas, é somada metade da altura do cilindro, com uma pequena margem, por razões de segurança, o que faz com que se obtenha o ponto no qual a ferramenta (garra) será atuada.

5.3.2 Atuação

Findada a conversão de referenciais, que culmina na obtenção do ponto para o qual o sistema de atuação se tem de mover, torna-se possível a atuação do manipulador e garra para a concretização da operação *pick-and-place*.

O manipulador Motoman possui um controlador, para o qual são enviados comandos, via *ethernet*, quando este se encontra configurado em modo remoto. Estes comandos incorporam os parâmetros necessários para definir o movimento do manipulador, tais como o tipo de movimento, a pose a atingir pela garra e respetiva orientação desta, a velocidade, entre outros. Realça-se o facto de o movimento se iniciar com a garra aberta, pelo que esta apenas é fechada quando atingida a posição parametrizada, de modo a agarrar objeto. Posto isto, o cilindro é transportado para uma posição estipulada.

Estabelecida a conexão com o manipulador, segue-se um conjunto de etapas sequenciais, que ficam a cargo do controlador NX100. Este começa por identificar os comandos recebidos, atuando em função disso, pelo que neste caso em concreto (operações *pick-and-place*), e de forma sucinta, as tarefas do controlador encontram-se apresentadas na figura 5.26.

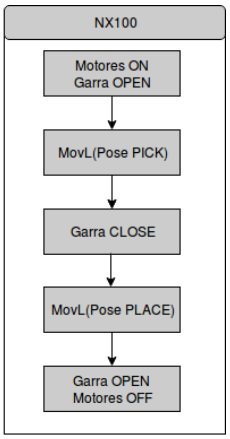

Figura 5.26: Conjunto de tarefas a cargo do controlador do manipulador, NX100

5.4 Conclusões

Inicialmente, a abordagem tomada para o desenvolvimento do *software* de aplicação aqui apresentado, tinha como objetivo a criação de um programa que não traduzisse um peso computacional significativo para o Raspberry Pi. O facto de o objeto a encontrar ser conhecido, previamente, em termos de forma e dimensões, permitiu tornar o seu reconhecimento mais simples. Esta motivação justifica a escolha de um cilindro como peça a detetar. A escolha da cor do objeto foi, neste caso, um fator de relevância, uma vez que favorece a aquisição de dados menos ruidosos. Sabendo que a técnica de luz estruturada calcula a profundidade de um ponto, baseando-se na interpretação da forma do padrão de luz infravermelha emitido e, tendo em conta que, a cor branca apresenta um maior índice de reflexão de luz, selecionou-se um objeto branco.

As condições de iluminação da bancada de testes não se aproximam de uma aplicação industrial, sendo que o sistema de iluminação usado foi luz natural e/ou lâmpadas da sala. As variações na iluminação podem afetar diretamente os resultados do processamento. No entanto, neste caso, acabaram por não influenciar de forma significativa.

Por outro lado, durante a implementação do código e familiarização com a PCL, surgiu o interesse no uso de algoritmos mais pesados, como é o caso do RANSAC, o que contraria a ideologia que serviu de base para o desenvolvimento do programa. O RANSAC é um algoritmo iterativo e probabilístico, que sendo robusto, traduz um peso computacional elevado. No entanto, o processo de filtragem prévia, ao reduzir o número de pontos numa grande escala, diminuiu a quantidade de recursos gastos na execução do algoritmo.

De uma forma geral, o programa permite a deteção do cilindro numa mesa, independentemente da orientação do seu eixo central, rejeitando outros objetos e ordenando ao manipulador que se desloque até este e o transporte até um ponto conhecido, usando para o efeito uma garra. A aplicação desenvolvida fornece ainda ao utilizador a informação do número de objetos presentes na mesa.

Capítulo 6

Testes e Resultados

O estudo no qual assenta esta dissertação, passa por avaliar o uso do Raspberry Pi no processamento de informação 3D de um determinado cenário, para o reconhecimento de objetos. Deste modo, o presente capítulo apresenta as vertentes nas quais este estudo incidiu, bem como os resultados que daí se apuraram, no subcapítulo 6.1. Por fim, em 6.2 surge uma conclusão comparativa relativamente aos casos de estudo apresentados.

6.1 Casos de Estudo

Tendo em conta que o foco do trabalho desenvolvido foi a criação de uma solução para reconhecimento de um cilindro, passível de ser executada num Raspberry Pi, os testes efetuados incidiram maioritariamente sobre as fases de aquisição, 2.1.1, e processamento da nuvem de pontos, 2.1.2. Esta escolha deve-se ao facto de, na fase de manipulação, a calibração compreender um processo de duração fixa, e a atuação apenas exigir do microcomputador o envio de comandos, por *ethernet*, para o controlador NX100 que, por sua vez, assegurará o movimento do manipulador e a atuação da garra.

Primeiramente, efetuou-se uma análise comparativa da execução do *software* de aplicação, no computador de desenvolvimento e no microcomputador de teste, apresentada em 6.1.1. Os testes realizados envolveram o estudo da influência de diferentes variáveis nas fases de aquisição e processamento da nuvem, entre as quais, o número de objetos presentes na mesa, 6.1.3 e o tipo de nuvem de pontos a processar, 6.1.4.

6.1.1 Teste geral de desempenho

Concluída a implementação do *software* de aplicação que traduz a versão original do mesmo, descrito em 5, procedeu-se à sua compilação e teste em ambas as ferramentas de processamento (Toshiba Satellite e Raspberry Pi). Tal como para a instalação de bibliotecas no Raspberry se recorreu a memória externa (*swap*), também para a compilação da solução proposta foi usada esta técnica. A duração do processo de compilação, bem como os recursos utilizados, no que toca a percentagem de CPU e quantidade de memória RAM, apresentam-se na tabela 6.1. Salienta-se

o facto de os processadores de ambas as ferramentas serem *quad-core*, sendo que o computador convencional apresenta uma memória RAM de 6GB, enquanto que o microcomputador possui uma RAM de apenas 1GB, com a possibilidade de ser expandida para 3GB, através de técnica de *swap*. Uma vez que a quantidade de código a ser compilado era bastante menor, comparativamente ao código associado às bibliotecas, neste caso optou-se por não restringir o número de *cores* no processo de compilação do *software*.

Tabela 6.1: Resultados do desempenho de cada plataforma de processamento durante a compilação do *software* de aplicação

	Toshiba Satellite	Raspberry Pi 3 Modelo B
Tempo	2 minutos e 3 segundos	8 minutos e 50 segundos
RAM	1536 MB	1606[1]MB
CPU	123%	105%

A gestão da distribuição de instruções pelos diferentes *cores* em ambas as plataformas é feita internamente. Assim, para efeitos de interpretação de resultados, salienta-se o facto de a percentagem do CPU apresentada ao longo deste capítulo poder assumir valores do intervalo [0,400]%. Uma vez que o processador possui quatro *cores*, um valor de 100% significa que a percentagem de CPU a ser usada nesse momento equivale a um *core*.

As variáveis acima tabeladas, foram apuradas, sob as mesmas circunstâncias, para a fase de execução do programa, mais propriamente, durante as etapas de aquisição e processamento de uma nuvem de pontos com três objetos colocados na mesa, pelo que os resultados podem observar-se na tabela 6.2. Para uma melhor análise de resultados, cada uma variável aqui avaliada para ambas as plataformas, isto é, tempo, memória RAM e CPU, encontra-se representado no gráfico da figura 6.1, 6.2 e 6.3, respetivamente.

Tabela 6.2: Resultados do desempenho de cada plataforma de processamento durante as fases de aquisição e processamento

Processo	Variável	Toshiba Satellite	Raspberry Pi 3 Modelo B
Aquisição	Tempo	0.108 segundos	0.320 segundos
	RAM	92.2 MB	81.92 MB
	CPU	6.3%	32.1%
Processamento	Tempo	0.06 segundos	0.450 segundos
	RAM	104.45 MB	101.376 MB
	CPU	10.3%	27%

[1] 826MB(RAM)+780MB(*Swap*)

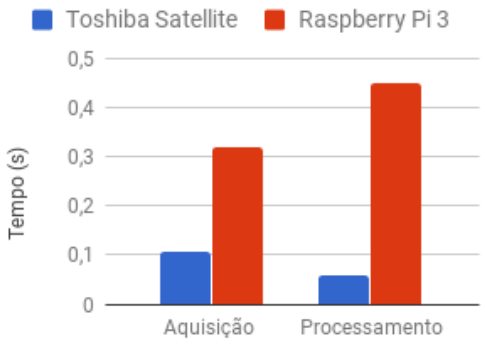

Figura 6.1: Duração dos processos de aquisição e processamento em ambas as plataformas

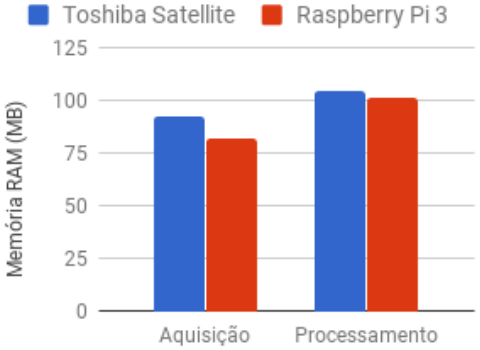

Figura 6.2: Memória RAM usada nos processos de aquisição e processamento em ambas as plataformas

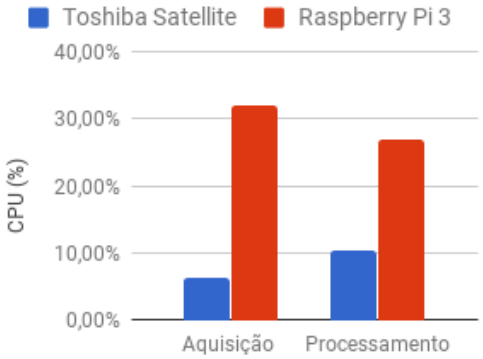

Figura 6.3: CPU usado nos processos de aquisição e processamento em ambas as plataformas

6.1.2 *Downsampling*

Na solução apresentada, a resolução da nuvem a ser processada é dada pela função *VoxelGrid*, abordada em 2.1.1, que aproxima ao seu centróide os pontos pertencentes a um cubo de aresta definida. Assim, quanto maior este valor, menor será a quantidade de pontos da nuvem resultante, o que, por sua vez, equivale a um tempo de processamento mais reduzido. Porém, há que garantir o sucesso no reconhecimento do cilindro, pelo que este valor, quando demasiado elevado, pode levar à perda de informação na nuvem que impede a correta identificação do objeto. É importante salientar que a nuvem na qual é aplicada a redução de resolução corresponde à nuvem resultante da filtragem ao longo dos eixos, feita anteriormente.

O teste aqui apresentado tem como intuito analisar o peso que o número de pontos, isto é, a resolução da nuvem que resulta do método *downsampling*, na duração da fase de processamento do *software* de aplicação. Para o efeito, admitiu-se um ambiente de teste com três objetos na mesa.

Neste sentido, foram aplicados diferentes valores à aresta do cubo, tendo-se avaliado o tempo de duração da fase, bem como o resultado da segmentação do cilindro, isto é, se foi bem ou mal

sucedida, representada, respetivamente, por um visto ou uma cruz, sendo que os resultados obtidos no microcomputador podem ser observados na tabela 6.3.

Tabela 6.3: Resultados obtidos no processamento da nuvem para diferentes resoluções no Raspberry Pi

Aresta voxel (cm)	Nº pontos	Tempo (segundos)	Reconhecimento
0 cm	170602	6.8	✓
0.5cm	21997	0.75	✓
1 cm	6108	0.45	✓
5 cm	294	0.11	✗
10 cm	82	0.10	✗

O primeiro teste traduz a não redução do número de pontos, daí o valor da aresta indicado para o *voxel* ser 0 cm. Os restantes valores foram selecionados de forma a avaliar a resposta do Raspberry ao processamento da nuvem associada.

6.1.3 Número de objetos

O teste aqui apresentado teve como objetivo estudar de que forma o número de objetos presentes na mesa influencia o tempo de processamento da nuvem no Raspberry, fazendo o mesmo teste para o computador convencional. Entenda-se por processamento, o conjunto de passos que permitem o reconhecimento de um cilindro numa determinada nuvem de pontos.

Foi considerado um número máximo de cinco objetos para este estudo, os mesmos encontram-se ilustrados na figura 6.4.

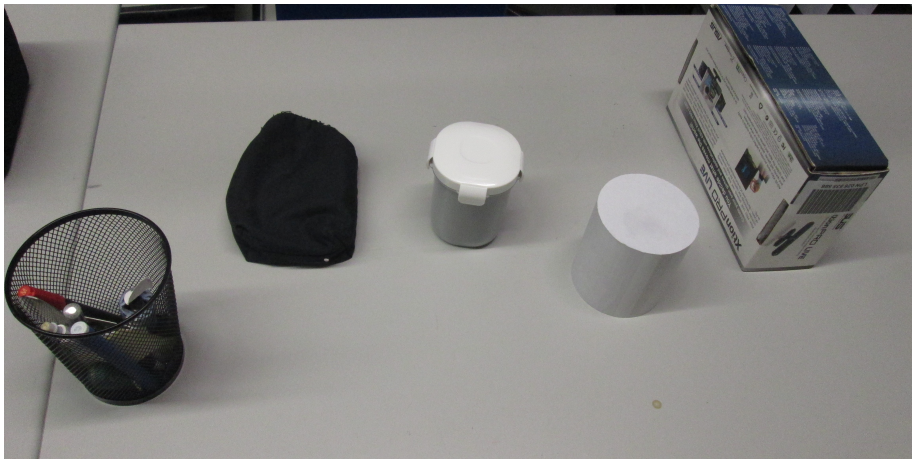

Figura 6.4: Conjunto de objetos usados para teste

Tabela 6.4: Duração da fase de processamento em função do número de objetos presentes na mesa

Número de objetos	Toshiba Satellite (segundos)	Raspberry Pi 3 Modelo B (segundos)
0	0.03	0.2
1	0.033	0.34
2	0.04	0.39
3	0.06	0.45
4	0.064	0.47
5	0.08	0.48

6.1.4 Tipo de nuvem de pontos

A *Point Cloud Library* disponibiliza diferentes tipos de nuvens nas quais é possível trabalhar. Aqui, pretende-se verificar se o uso de uma nuvem apenas com informação da profundidade ou uma nuvem que contenha também o valor da componente RGB para cada ponto, influencia o seu tempo de aquisição e processamento. Mais concretamente, foram usados dois tipos de nuvens diferentes para este estudo, *PointXYZ* e *PointXYZRGB*, sendo que a mesa se encontrava com três objetos, novamente.

O uso da componente RGB é, sem dúvida, uma mais-valia para o processamento de uma nuvem, pois fornece mais informação sobre o cenário, isto é, imprime a cor de cada ponto da nuvem. Esta informação pode permitir acelerar o reconhecimento de um objeto. No entanto, é necessário ter em conta que este tipo de nuvem confere um maior peso computacional do que uma simples nuvem de profundidade.

A tabela 6.5 mostra os resultados obtidos neste estudo, em ambas as plataformas.

Tabela 6.5: Resultados para as fases de aquisição e processamento usando dois tipos distintos de nuvens de pontos

		Toshiba Satellite		Raspberry Pi 3 Modelo B	
Processo	Variável	Depth	RGB-Depth	Depth	RGB-Depth
Aquisição	Tempo	0.108 seg	0.55 seg	0.320 seg	1.45 seg
	RAM	92.2MB	100,1MB	81.92 MB	362MB
	CPU	6.3%	12.6%	32.1%	71.45%
Processamento	Tempo	0.060 seg	0.10 seg	0.450 seg	0.65 seg
	RAM	104.448 MB	176.64 MB	101.376 MB	184.32 MB
	CPU	10.3%	16.1 %	27%	29.7%

6.1.5 Interface de aquisição

Uma abordagem possível para garantir a aquisição de dados do sensor compreende a criação da interface com o mesmo, seguida da aquisição de dados e posterior paragem desta interface,

pelo que a nuvem resultante desta aquisição, é encaminhada para a fase de processamento. Outra hipótese inclui a criação da interface, deixando-a ligada durante a etapa de processamento da nuvem.

O caso de estudo aqui abordado pretende apurar qual a melhor metodologia a adotar para o microcomputador em causa. Mais propriamente, pretende-se estudar a influência da interface com o sensor durante a fase de processamento. Por isso, obtiveram-se os valores temporais, de percentagem de CPU usado, bem como de RAM para a etapa de tratamento da nuvem, no Raspberry para ambas as abordagens, isto é, "Interface ON-OFF"e "Interface ON-ON". Os resultados encontram-se representados na tabela 6.6.

Tabela 6.6: Resultados do desempenho do Raspberry Pi na fase de processamento, para diferentes abordagens de interface com o sensor

	Variável	Interface ON-OFF	Interface ON-ON
Processamento	Tempo	0.45 segundos	1.3 segundos
	RAM	101.376 MB	124.92 MB
	CPU	27%	208.2%

6.1.6 *Headless*

Em contexto industrial, a monitorização de processos é uma necessidade recorrente, sendo realizada tipicamente por meio de uma consola. Esta monitorização pode passar pelo controlo visual de um determinado processo, realizando, para o efeito, *streaming* de vídeo.

O teste que aqui se apresenta pretende avaliar o uso do Raspberry para este tipo de aplicação, isto é, como suporte de uma interface HMI. Para o efeito, conectou-se um monitor, rato e teclado ao microcomputador, para garantir o acesso nativo ao mesmo. Posteriormente, o sensor foi ligado objetivando-se o *streaming* de dados em formato de vídeo.

Este teste avaliou diferentes variáveis, entre elas a taxa de aquisição do sensor, a temperatura do processador, bem como a percentagem de CPU usada pelo processo. Os resultados associados a este estudo encontram-se na tabela 6.7, sendo feito o mesmo estudo para o computador de desenvolvimento, o Toshiba. O cenário em causa incluía, novamente, a presença de três objetos na mesa.

Para o Raspberry Pi, o intervalo de temperatura admissível que assegura a operacionalidade do *System-on-a-Chip* (SoC) que possui, compreende um intervalo de -40° C a 85°C. Salienta-se ainda o facto de que quando o processador não se encontra a desempenhar uma tarefa em específico, ou seja, quando o Raspberry apenas está ligado, a temperatura do SoC ronda os 50°C.

Como referido em 2.1.1.4, a taxa de aquisição do sensor Asus Xtion Pro Live apresenta um valor de 30 *frames* por segundo (FPS). Este parâmetro foi possível de verificar no computador de desenvolvimento, porém, no Raspberry Pi esta frequência não ultrapassa os 15FPS. Para além disso, a taxa de atualização de imagem no *streaming* realizado revelou-se bastante menor do que a taxa de aquisição do sensor.

Tabela 6.7: Índice de desempenho do computador de desenvolvimento e microcomputador de teste para *streaming* de dados do sensor

	Toshiba Satellite	Raspberry Pi 3 Modelo B
Frame Rate	30FPS	15FPS
Temperatura	62°C	82.7°C
CPU	175.4%	323.7%

Adicionalmente, estudou-se a influência do tipo de acesso ao microcomputador na execução do *software* de aplicação desenvolvido. Para o efeito, foram comparados parâmetros como o tempo, a memória RAM usada e a percentagem de CPU exigida pelas fases de aquisição e processamento da solução proposta, recorrendo a um acesso via SSH e um acesso nativo.

Tabela 6.8: Resultados do desempenho do Raspberry Pi na execução do *software* de aplicação, via SSH e nativamente

Processo	Variável	Acesso Nativo	Acesso Remoto
Aquisição	Tempo	0.27 segundos	0.320 segundos
	RAM	96.27 MB	81.92 MB
	CPU	41.3%	32.1%
Processamento	Tempo	0.48 segundos	0.45 segundos
	RAM	100.8 MB	101.376 MB
	CPU	30%	27%

6.2 Conclusões

Começando pelos testes de performance efetuados tanto no computador de desenvolvimento como no microcomputador de teste, destaca-se a clara discrepância entre tempos de compilação do *software* de aplicação. Este facto não se revela negativo, pois o processo de compilação apenas ocorre uma vez. A necessidade do uso de memória adicional contribuiu para o aumento deste tempo.

A percentagem de CPU usada na compilação, em ambas as plataformas, ultrapassa os 100%, o que significa que o processador recorre a mais do que um *core* para este processo. No Raspberry Pi, a distribuição de instruções pelos quatro *cores* fica ao cargo do próprio microcomputador, que neste processo em concreto utilizou 95% de um *core* e os restantes 10% de outro, o que remete para um uso não balanceado dos *cores*.

Na avaliação do desempenho do Raspberry Pi na execução do *software* de aplicação, verifica-se que a razão entre a memória RAM gasta e a disponível é superior em comparação ao computador convencional. Para além disso, como seria de esperar, o tempo de aquisição e processamento da nuvem no microcomputador é significativamente maior do que no computador Toshiba. Este

resultado pode ser explicado pela velocidade do processador do Raspberry Pi ser de 1.2GHz, enquanto que a velocidade do processador do computador Toshiba é de 1.6GHz.

Nesta análise geral de desempenho verificam-se percentagens de CPU usado distintas para as duas plataformas. Para uma melhor leitura dos dados obtidos, o *software* de aplicação foi executado de forma cíclica no Raspberry, medindo-se os valores deste parâmetro de avaliação em cada ciclo. Verificou-se que o valor da percentagem de CPU usado para a etapa de processamento, em ciclos distintos para uma mesma nuvem de pontos, apresentava grandes discrepâncias. Este resultado deve-se ao facto de, durante a fase de processamento, se recorrer a um algoritmo probabilístico.

O RANSAC seleciona um conjunto de pontos aleatório, realizando um determinado número de iterações de modo a encontrar o conjuntos de pontos que melhor se aproximam do modelo definido. Deste modo, e para o melhor caso, este pode encontrar os *inliers* do cilindro nas primeiras iterações, o que traduz um baixo peso computacional no CPU. Porém, este método pode apenas apurar este conjunto de pontos, após um número considerável de iterações. Neste caso, a percentagem de CPU usada apresenta um valor significativamente mais elevado, em função do número de iterações que realizou. Desta forma, conclui-se que a percentagem de CPU utilizado na fase de processamento pode assumir valores muito distintos entre testes.

Pela análise do teste realizado no processo de *downsampling*, foi possível apurar que o número de pontos de entrada para um algoritmo, neste caso de segmentação de um cilindro, é um fator crucial capaz de determinar o tempo de execução. Numa abordagem desprovida de redução de resolução, o tempo de processamento da nuvem no microcomputador ultrapassa os 6 segundos, o que numa aplicação de *pick-and-place* traduz um tempo elevado.

Deste modo, reconhece-se a necessidade de aplicar *downsampling* à nuvem de pontos, no entanto é necessário atribuir um tamanho para a aresta do cubo, que não comprometa o reconhecimento do objeto pretendido. Como se verifica para uma aresta de 5 ou 10 cm que, apesar de reduzir drasticamente o tempo de processamento, em ambos os casos, os valores selecionados levam à perda de informação relevante que impede a deteção de um cilindro na nuvem.

No que toca ao estudo que tem em conta o número de objetos na mesa, como seria de esperar, a duração da etapa de processamento aumenta com o número de objetos. É sabido que a cada objeto será aplicado o algoritmo RANSAC, de modo a identificar o cilindro. Sendo o RANSAC um método que confere algum peso computacional, este facto contribui para uma maior duração do processamento. No Raspberry Pi verifica-se que este aumento é mais acentuado do que no computador Toshiba. Este resultado poderá ser indicador de uma limitação para o uso do Raspberry numa aplicação que imponha tempos de processamento mais exigentes, por exemplo, na ordem dos 0.2 segundos.

Conclui-se da tabela 6.5 que o tipo de nuvem pontos influencia diretamente as fases de aquisição e processamento. Desta influência, a mais evidente, sem dúvida, é o tempo que uma nuvem com componente RGB confere a este processo.

Na aquisição de dados, quando se trata de uma nuvem do tipo *PointXYZRGB*, para além da informação da profundidade, é necessário obter o valor de três componentes adicionais, *red* (R),

green (G) e *blue* (B), o que aumenta significativamente a duração deste processo. Uma maior quantidade de informação a adquirir, neste caso não equivale a uma maior quantidade de dados a processar, uma vez que o processamento trata somente a informação das variáveis que ditam a profundidade. No entanto, é notório o aumento da memória RAM utilizada, que pode ser explicado pelo facto de se operar um estrutura de dados de maiores dimensões.

Pela análise de resultados, no que toca ao desempenho do Raspberry na fase de processamento da nuvem, usando diferentes abordagens para a interface com o sensor, pode-se concluir que mantendo a interface ao longo desta etapa traduz-se no pior caso. A abordagem que mantém a interface com o sensor durante a fase de processamento, requer mais poder do CPU, o que se reflete no valor de 208.2%, equivalendo ao uso de mais de dois *cores*. Assim, conclui-se que esta metodologia não é a mais apropriada, pois não faz um bom aproveitamento de recursos.

Na análise do uso de um sistema operativo gráfico, numa aplicação de interface com o utilizador, na qual é feito *streaming* de dados adquiridos do sensor, pode-se concluir que esta tarefa requer bastante poder de um processador. A percentagem de CPU exigida para assegurar o processo em causa, inviabiliza o uso do Raspberry para este tipo de aplicação, isto é, como unidade de processamento central de uma consola.

Pode ainda verificar-se que a reprodução de dados obtidos do sensor, em tempo real no sistema operativo do Raspberry, traduz no SoC uma temperatura que se aproxima da temperatura máxima admissível. Este resultado contribui também para concluir que o microcomputador em estudo não traduz a solução mais adequada para aplicações de interface HMI.

Apesar de a taxa de aquisição de dados do sensor, quando conectado ao Raspberry, ser de 15FPS, a taxa de atualização do vídeo produzido não traduz essa *frame rate*. Este facto pode ser explicado pela quantidade de CPU usada pela tarefa em questão, que ocupa uma percentagem equivalente a mais do que três *cores* do processador.

Para o *software* de aplicação desenvolvido nesta dissertação, a redução da *frame rate* de aquisição de dados para 15FPS, não tem qualquer impacto uma vez que a duração da fase de processamento da nuvem de pontos, para o melhor caso, é de 0.2 segundos. Porém, esta redução poderá inviabilizar o uso do Raspberry em aplicações de tempo real, que exijam uma taxa de aquisição maior.

Por outro lado, pela comparação de resultados obtidos na execução do programa de forma nativa bem como remotamente, conclui-se que não existe uma diferença notória nas variáveis avaliadas. Contudo, o uso de um sistema operativo gráfico faz uso de outros recursos, por isso, e para garantir a menor sobrecarga ao Raspberry, manteve-se a abordagem *headless* inicialmente definida.

Em suma, pelos testes efetuados, é possível definir quais as variáveis que mais influenciam o tempo de processamento de uma nuvem no *software* de aplicação apresentado.

Um dos requisitos mais importantes de qualquer aplicação industrial corresponde ao seu tempo de resposta. Os testes efetuados demonstram que a solução proposta, quando executada no Raspberry, apresenta uma capacidade de resposta na ordem das décimas de segundo, o que para a tarefa

em causa (*pick-and-place*) não traduz uma duração muito longa, no entanto poderá ser um fator impeditivo do uso deste microcomputador, em tarefas mais exigentes.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusões

Hoje em dia, os sistemas robóticos instalados na indústria são suportados por um computador industrial de elevado custo. A constante procura pela minimização de custos, potencia o uso de outras soluções mais baratas, para o mesmo efeito. Deste modo, a dissertação aqui apresentada focou-se no estudo do uso de uma plataforma *low-cost*, nomeadamente, o Raspberry Pi, em detrimento de um PC industrial para o controlo de uma determinada tarefa. Assim, este estudo incluiu o desenvolvimento de uma solução capaz de replicar uma operação tipicamente industrial, mais propriamente, tarefas *pick-and-place*. Normalmente, este tipo de operação tem a si associado um sistema de sensorização responsável por captar informação do cenário em causa, bem como um programa responsável pelo tratamento de dados, por forma a localizar o objeto pretendido. Para além disso, estas operações são realizadas por manipuladores industriais, que tem a seu cargo a tarefa *pick-and-place* propriamente dita.

Desta forma, a análise do microcomputador compreendeu as fases que vão desde o seu *setup*, até ao desenvolvimento e teste de uma aplicação que contribuiu para a avaliação da sua aplicabilidade na indústria. Relativamente à fase de *setup*, foi possível reconhecer várias complicações. Concretamente, foi exigida a compreensão da arquitetura na qual este assenta, a arquitetura ARM, a qual foi estudada e documentada. Nesta fase foram ainda bordados dois métodos de compilação de bibliotecas, mais propriamente, a *cross-compilation* e a compilação nativa. Desta análise, pôde concluir-se que a primeira técnica não se revelou uma abordagem eficaz, uma vez que não foi possível garantir a operacionalidade das bibliotecas. Por outro lado, a compilação nativa constitui um método de elevada eficácia, que assegurou a utilização das bibliotecas compiladas.

Já na fase de desenvolvimento do *software*, apresentou-se um programa capaz de reconhecer a presença, ou não, de um cilindro numa mesa, pelo que, em caso afirmativo, são apuradas as coordenadas do mesmo, com vista à sua manipulação. Pela análise de resultados pode afirmar-se que, para a aplicação em estudo, o Raspberry Pi apresentou um comportamento aceitável. No entanto, apesar da tentativa de se aproximar esta solução de uma aplicação industrial, reconhece-se que está ainda longe de o ser, uma vez que não inclui, por exemplo, o contorno de obstáculos.

Por isso, a aplicabilidade deste microcomputador no âmbito industrial dependerá essencialmente do tipo de tarefa a que se destina.

Os testes efetuados permitiram apurar as variáveis que mais influenciam o desempenho deste microcomputador durante o processo de aquisição e processamento de dados. Dos resultados obtidos, é ainda possível concluir que o Raspberry, apesar de responder num período de tempo na ordem das décimas de segundo, não consegue fazer face ao desempenho temporal do computador convencional utilizado.

Quanto à sua utilização para aplicações de processamento de nuvens de pontos, observou-se que a sua capacidade é suficiente para uma aplicação mais simples, que não inclui requisitos específicos e exigentes de tempo real. Esta caraterística habilita esta ferramenta emergente para ser usada em aplicações de processamento de imagem ou informação 3D de média/baixa exigência computacional.

Por fim, acredita-se que a próxima versão do Raspberry apresente melhores caraterísticas como, por exemplo, uma maior quantidade de memória RAM, que permitirá melhorar a capacidade de resposta a uma aplicação industrial e que continue a apresentar um preço competitivo.

7.2 Trabalho Futuro

Como trabalho futuro sugere-se primeiramente o estudo de outras versões de *software* de aplicação. Numa instância, poder-se-ia incluir uma abordagem de planeamento de trajetórias para controlo do movimento do manipulador. Numa outra instância, alargar a análise da avaliação do seu uso para outras aplicações de domínio industrial.

Em segundo lugar, para um melhor aproveitamento das potencialidades do microcomputador estudado, sugere-se o estudo do uso da sua unidade de processamento gráfico. Isto, com o intuito de acelerar a execução do *software* desenvolvido.

Por fim, poder-se-ia estudar a aplicabilidade de outros microcomputadores existentes no mercado, que apresentem um preço semelhante. Do mesmo modo, também identificar e avaliar o desempenho da plataforma de baixo custo acoplada a outras ferramentas de aquisição, se prevê como um ponto a considerar futuramente.

Anexo A

Fundamentos sobre Arquitetura de Computadores

Com o propósito de se avaliar a viabilidade do uso de um Raspberry Pi para controlo de processos industriais, começou-se por estudar o seu princípio de funcionamento. Com especial foco na unidade de processamento central, a secção A.1 apresenta a metodologia de operação de um processador genérico, sendo feita uma descrição mais pormenorizada do funcionamento de processadores de arquitetura *Acorn/Advanced* RISC *Machine*, ARM.

A.1 Raspberry Pi

Primeiramente, é imperativo entender a metodologia de funcionamento do microcomputador selecionado, o Raspberry Pi 3 Modelo B. Este estudo é feito tendo por base as especificações do mesmo, enumeradas na secção 2.1.2.4 e ilustradas no diagrama apresentado na figura A.1.

No contexto deste microcomputador, o conceito System-on-a-Chip (SoC), corresponde a um circuito integrado, que reúne a maioria dos componentes de um computador num só *chip*. Estes componentes incluem a unidade de processamento gráfico (GPU), unidade processamento central (CPU) e a memória RAM. Atualmente, os SoCs são largamente utilizados, pelo que uma das maiores vantagens se reflete no baixo consumo de energia, quando comparado a um computador com *hardware* tradicionalmente distribuído. O Raspberry Pi 3 modelo B é baseado no SoC Broadcom 2837, o qual incorpora o processador *quad-core* Cortex-A53, da fabricante ARM, bem como o processador gráfico *dual-core* VideoCore IV, pelo que nesta versão da placa, a memória RAM não pertence a este chip, estando localizada na face inferior do microcomputador.

O Raspberry Pi trata-se de um computador *single-board*[1], desprovido de disco rígido próprio, sendo que o seu sistema operativo, encontra-se alocado num cartão de memória micro SD. Atualmente, existe um conjunto de distribuições, baseadas em Linux, preparadas para operar neste microcomputador, o lhe que confere versatilidade.

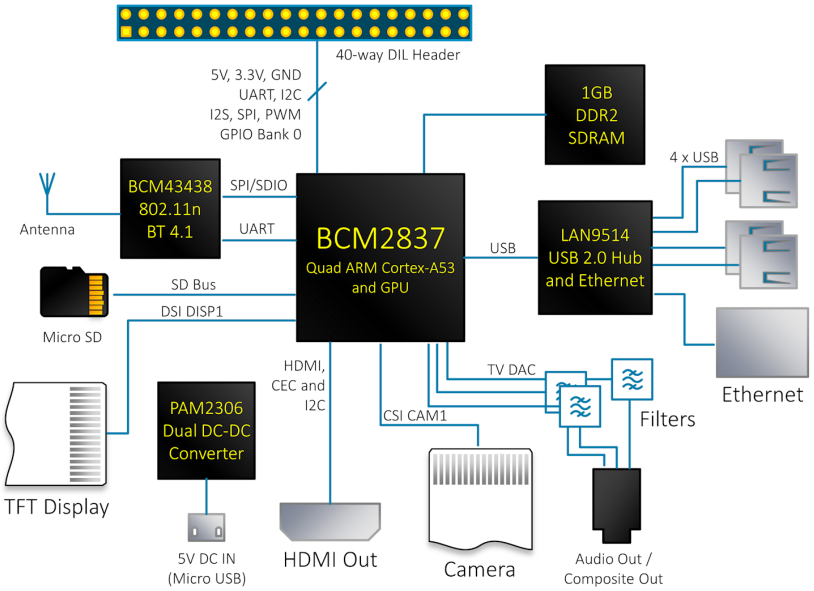

Figura A.1: Diagrama com os constituintes do Raspberry Pi 3 [2]

Relativamente à capacidade do processador do Raspberry Pi, apesar de esta ser de 64 bits, este opera a 32 bits. Tendo em conta que um CPU tem a seu cargo a execução de instruções, mais especificamente, cálculos lógicos e aritméticos, bem como instruções que movem dados entre registos, a grande diferença entre processadores de 32 e 64 bits assenta no número de cálculos que estes conseguem fazer por segundo, o que influencia diretamente a velocidade a que completam processos.

Este valor pode ser eficazmente aumentado pela quantidade de *cores* que o processador apresenta. Em dispositivos *multicore*, com duas ou mais unidades de processamento, o sistema operativo reconhece cada núcleo como um processador diferente, capaz de ler e executar instruções em registos. Isto significa que um único processador é capaz de efetuar diferentes instruções, paralelamente, em *cores* separados, numa abordagem multi-tarefa. Neste sentido, é de ressaltar que o Raspberry Pi apresenta um CPU *quad-core*, isto é, possui 4 unidades de processamento independentes, com um *clock* de 1,2GHz cada uma.

O Raspberry Pi possui ainda uma unidade de processamento gráfico (GPU), que desempenha um papel importante para o aumento da velocidade de processos, por permitir processamento paralelo. A metodologia na qual este processador se baseia não se aplica a qualquer tipo de processo, realizando maioritariamente instruções aritméticas. As instruções que um GPU é capaz de executar são tipicamente mais simples e específicas, quando comparadas às de um CPU, no entanto, o último não explora de forma exaustiva este conceito de *multithread*. Para além de ser usado para o *display* gráfico de um sistema operativo num monitor, é comum recorrer-se a um processador gráfico para aplicações que envolvam visão computacional. Em jeito de conclusão, o

[2] https://www.element14.com/community/community/raspberry-pi/blog/2017/01/16/raspberry-pi-3-block-diagram

[2] Computador de placa única

CPU tem a seu cargo uma vasta panóplia de instruções, que se caracterizam pela sua complexidade, o que limita a sua otimização. Por outro lado, o GPU é responsável por executar um número reduzido de instruções de baixa complexidade, este facto permite acelerar processos.

O subcapítulo A.1.1 compara dois tipos de computadores distintos, focando-se especialmente na arquitetura na qual este microcomputador assenta. Seguidamente, é apresentado o princípio de funcionamento de um processador, em A.1.2.

A.1.1 RISC e CISC

Sustentado por uma arquitetura ARM, o Raspberry Pi baseia-se na tipologia RISC (*Reduced Instruction Set Computer*), enquanto que os computadores convencionais seguem uma metodologia CISC (*Complex Instruction Set Computer*). A grande distinção entre estes foca-se no conjunto de instruções que o seu processador pode realizar, isto é, a versão da *Instruction Set Architecture* (ISA) que possuem. As instruções correspondem às ordens que um processador recebe de modo a realizar instruções, este processo encontra-se explicado à frente em A.1.2

Uma outra distinção entre estas arquiteturas assenta na forma como é feito o acesso à memória durante a execução de instruções. A memória de um computador, mais propriamente, a memória acedida e manipulada pelo CPU, segue uma estrutura hierárquica, representada na figura A.2, podendo ser distinguida em três tipos de memória, mais propriamente, registos, *cache* e memória RAM.

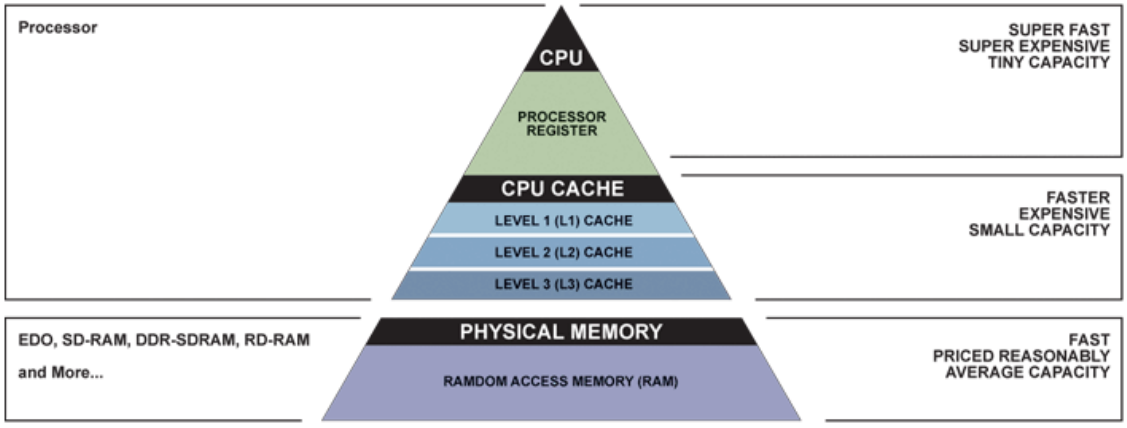

Figura A.2: Estrutura hierárquica da memória acedida pelo CPU [3]

O processo de leitura e escrita em registos é o mais rápido, quando comparado ao acesso a qualquer outro tipo de memória, uma vez que os primeiros se encontram alocados na unidade de processamento central (CPU). Salienta-se, no entanto, a baixa capacidade de armazenamento que possuem. A *cache* é tipicamente usada como alternativa ao acesso da memória principal. Este procedimento deve-se ao facto de a *cache* permitir uma manipulação rápida, tornando possível acelerar o processo de execução de instruções, apesar de esta traduzir um espaço reduzido de

[3] Adaptada de https://www.bit-tech.net/hardware/memory/2007/11/15/the_secrets_of_pc_memory_part_1/3

memória. Por fim, a memória RAM, que se encontra instalada externamente ao CPU, possui um tempo de acesso lento.

Neste âmbito, os processadores do Raspberry, GPU e CPU, efetuam cálculos apenas em registos, pelo que não correm instruções diretamente na memória.

Para além disto, o conjunto de instruções presente num RISC é otimizado, o que permite que estas sejam executadas recorrendo a uma quantidade baixa de transístores, tornando o dispositivo mais barato e de baixo consumo energético. Acrescenta-se que, o facto de serem instruções simples favorece a sua execução imediata. Apesar do baixo custo e simplicidade do *hardware*, o seu desempenho depende diretamente da complexidade da aplicação pretendida.

Por outro lado, os sistemas CISC podem operar com registos ou manipular diretamente dados da memória. Destacam-se por possuir um conjunto de instruções mais completo, o que faz deles processadores poderosos, no entanto, uma vez que realizam instruções na memória RAM, o consequente tempo de acesso pode atrasar o processo. Tendo em conta que são capazes de realizar instruções mais complexas, a execução das mesmas subentende a sua conversão para porções de código mais simples (*microcode*), o que implica um certo número de ciclos de relógio até terminar.

Em suma, para um mesmo resultado, plataformas ARM, isto é, dispositivos que se baseiam numa arquitetura RISC, necessitam de realizar mais instruções do que os CISC.

A.1.2 Princípio de funcionamento do processador

Independentemente do tipo de processador, é importante ter em conta as fases que completam a execução de uma instrução. Assim, de uma forma geral, o princípio de funcionamento de um CPU contempla três passos principais. Primeiramente, uma instrução binária é transferida da memória para a unidade de processamento, segue-se a sua interpretação e, por fim, esta é executada, pelo que o seu resultado é escrito em registos.

Uma instrução encontra-se na memória como um valor binário de comprimento na ordem dos bytes, pelo que no Raspberry Pi esta corresponde a 4 bytes, isto é, 32 bits. Nesta representação está definida a identidade da instrução (*operation code*), assim como um ou mais operandos, que correspondem a valores ou endereços associados à instrução. A *instruction set* de processadores ARM caracteriza-se por conter um reduzido número de instruções para manipulação de dados.

A figura A.3 representa, resumidamente, o conjunto de etapas para a execução de uma instrução. É sabido que o CPU necessita de conhecer a tarefa para esta ser efetuada, para o efeito existe um registo denominado *program counter*, no qual é alocado o endereço da instrução a ser executada, tratando-se por isso de um apontador. Nesta primeira fase, designada *Instruction Fetch* (IF), a instrução é transferida para um registo do CPU. Após a receção da mesma, o *program counter* é incrementado num valor de 4 bytes, de modo a apontar para a próxima posição na memória, que corresponde à instrução seguinte. Posteriormente, dá-se a descodificação da instrução, Decode (DE), a qual interpreta o cálculo a realizar que, por sua vez, fica ao cargo da unidade aritmética (ALU) numa instância final, Execute (EX). Em função da lógica enviada pelo descodificador, a ALU efetua a instrução, terminando com a escrita do resultado num registo.

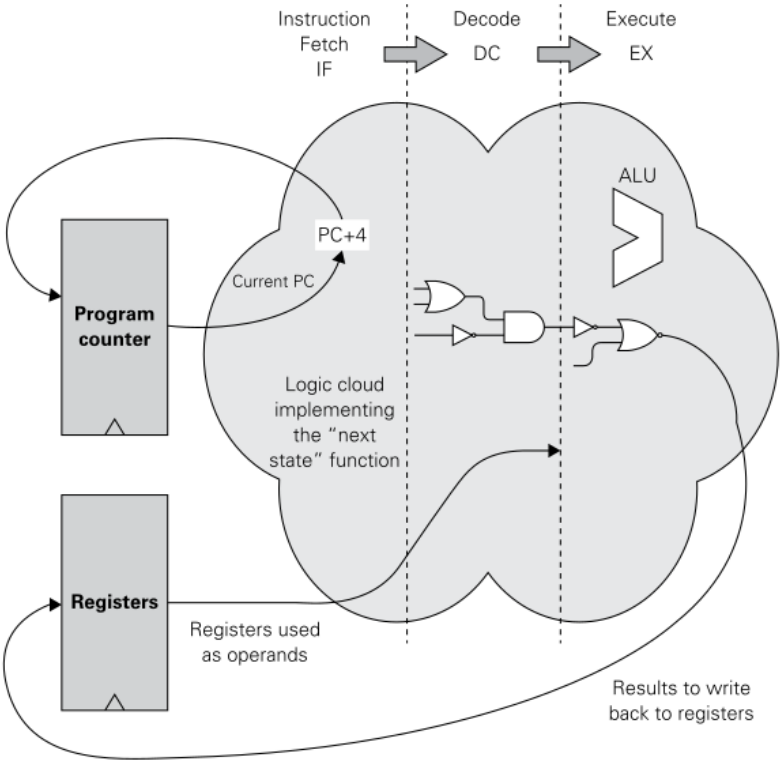

Figura A.3: Processo de execução de uma instrução num processador [12]

Inicialmente, o conjunto de etapas enumerado acima era executado como um todo, isto quer dizer que, a instrução seguinte apenas era iniciada quando findada, por completo, a primeira. Esta metodologia não fazia um bom proveito dos recursos e, neste contexto, surgiu a técnica de *pipeline*, ilustrada na figura A.4, vocacionada maioritariamente para sistemas RISC. Sabendo que a realização de uma instrução envolve, por exemplo, três etapas fixas e sequenciais, o *pipeline* sugere a execução sobreposta de tarefas de forma independente, sem que haja conflito entre as mesmas, uma vez que não partilham recursos. Assim, a metodologia proposta aloca cada uma das fases de execução de uma instrução num ciclo.

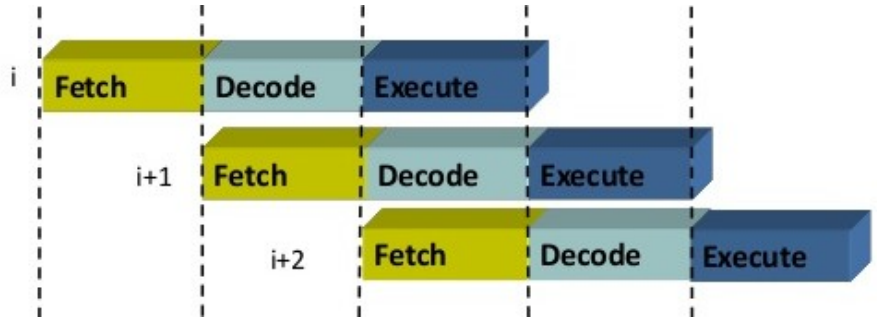

Figura A.4: Princípio de funcionamento da metodologia *pipeline* [4]

[4] Adaptada de https://www.slideshare.net/phzope75/unit-ii-arm-7-introl

As instruções encontram-se no CPU, isto é, listadas em memória, onde aguardam para ser executadas. Quando terminada a etapa inicial da primeira instrução, isto é, finalizado o primeiro ciclo, esta segue para a segunda fase. Em simultâneo, a segunda instrução pode iniciar a primeira etapa, e assim sucessivamente, pelo que no ciclo imediatamente a seguir são executadas paralelamente três tarefas, correspondendo a instruções e fases distintas. Esta abordagem, tem como principal objetivo um melhor aproveitamento da capacidade de processamento de um CPU, sendo utilizada para aumentar a sua velocidade de operação.

Com o avanço da tecnologia e popularização do conceito RISC, foram surgindo processadores cada vez mais poderosos, destinados, por exemplo, a aplicações com requisitos de tempo real. O aumento da complexidade despoletou a necessidade de se dividir uma instrução em mais subetapas, de modo a melhorar a capacidade de resposta destes processadores. Assim, atualmente, os tipos mais usuais de arquiteturas RISC, que recorrem à técnica de *pipeline*, incluem instruções com 3, 5 e 8 etapas. O CPU presente no Raspberry Pi possui uma arquitetura capaz de operar a 32 ou 64 bits, pelo que a primeira corresponde à implementada atualmente no mesmo, como referido acima. Isto significa que o processador do microcomputador usado, apesar de possuir uma arquitetura Armv8-A, assenta na versão anterior Armv7-A, baseada numa *instruction set* de 32 bits. [12]

Salienta-se que há um compromisso relacionado com a máxima velocidade de execução de uma tarefa, que é necessário garantir, por isso a ordem do *pipeline* associado a um processador basear-se-á sempre no *instruction set* que o define.

A máxima sobreposição de tarefas definida num CPU depende principalmente do número de etapas em que a execução de um instrução pode ser dividida. Para além disto, este estudo passa também por avaliar se compensa, ou não, subdividir uma fase, por exemplo, para uma determinada etapa, se a sua duração for inferior à duração de um ciclo, não compensa subdividi-la, pois o tempo mínimo de duração de uma etapa corresponde a um ciclo. Acrescenta-se que a duração de cada ciclo corresponde à etapa mais lenta da instrução. Esta divisão duplicaria o tempo de execução da etapa de um ciclo para dois.

O Raspberry Pi 3, microcomputador usado na presente dissertação, apresenta um processador Cortex-A53 que, segundo a fabricante ARM, oferece um meio termo entre eficiência energética e desempenho. Ressalta o facto de se tratar da versão ARMv8-A baseado num CPU da família Cortex-A, sendo que esta foi dimensionada para operar com uma *instruction set* de 64 bits, contemplando um conjunto de 8 etapas às quais pode ser aplicada a técnica de *pipeline*. Por outro lado, este processador possui a versatilidade de funcionar também para conjuntos de instruções de 32 bits, modo de funcionamento aplicado ao Raspberry Pi 3. As principais unidades que sustentam o seu funcionamento encontram-se representadas na figura A.5. [13]

Dos sistemas que compõem esta arquitetura, destaca-se a tecnologia NEON correspondente a uma unidade *Single Instruction, Multiple-Data* (SMID), que manipula vetores para a execução de instruções em múltiplos dados ao mesmo tempo, instruções necessárias para uma boa performance em aplicações de vídeo e som, por exemplo. Também para a execução de instruções, surge a *Floating Point Unit*, FPU, responsável por realizar instruções com valores de vírgula flutuante. Esta

destaca-se por permitir a realização de cálculos via *hardware*, em detrimento de uma abordagem sem esta unidade, que efetua os cálculos recorrendo a *software*, pela emulação de uma FPU, o que tem a si associado um maior tempo de processamento. Este processador possui uma quantidade variável de memória *cache* de dados L1 (D-*cache*) e de instruções L1 (*I-cache*), pelo que a memória L2 é, por sua vez, de uso opcional. A memória *cache*, de carácter temporário, fica localizada no CPU e por isso permite um rápido acesso. Para além deste facto, esta encontra-se dividida em níveis hierárquicos, daí a designação L1 e L2 (*level*), sendo que a de nível 1 (L1) é a primeira a ser acedida e, ao mesmo tempo, a que se encontra mais próxima do CPU.

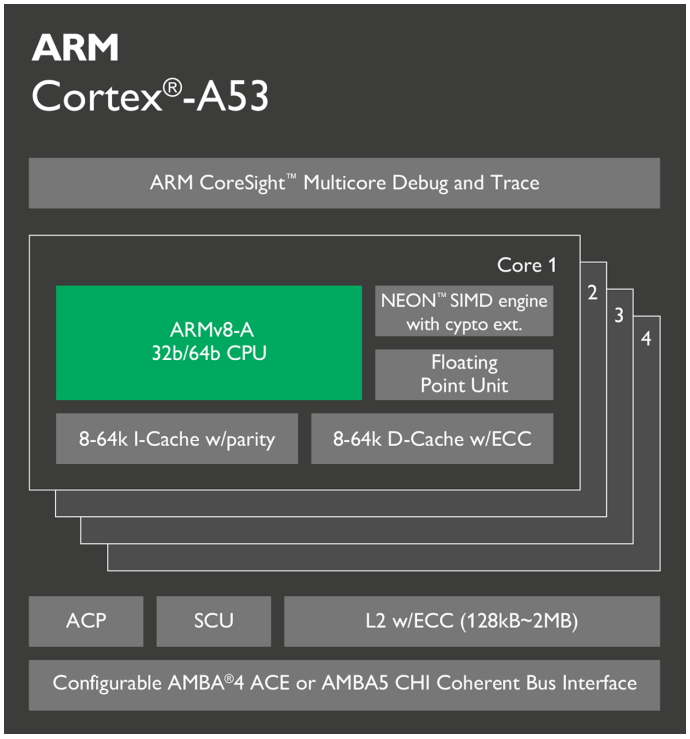

Figura A.5: Principais constituintes do CPU ARM Cortex A-53 [13]

A existência de uma unidade dedicada a instruções que envolvam operandos com vírgula flutuante, bem como a existência de uma *instruction set* de 32 *bits* fazem do Raspberry Pi 3 um dispositivo de arquitetura *Advanced RISC Machine Hardware Floating point* (ARMHF), que assenta numa interface *Embedded Aplication Binary Interface* (EABI). Esta interface define um conjunto de normas, por exemplo, para a interação entre funções e na forma como os dados são representados em memória.

O tipo de processador e respetivas unidades que o complementam, fazem desta versão do Raspberry Pi um dispositivo *ARM Hardware Floating point*.

Anexo B

Guideline de Instalações

B.1 Memória *Swap*

Começar por definir a quantidade memória adicional pretendida, no máximo 2048 MB.

Passo 1 : Definição da quantidade de memória a adicionar
sudo nano /etc/dphys-swapfile
Definir *CONF_SWAPSIZE=2048*

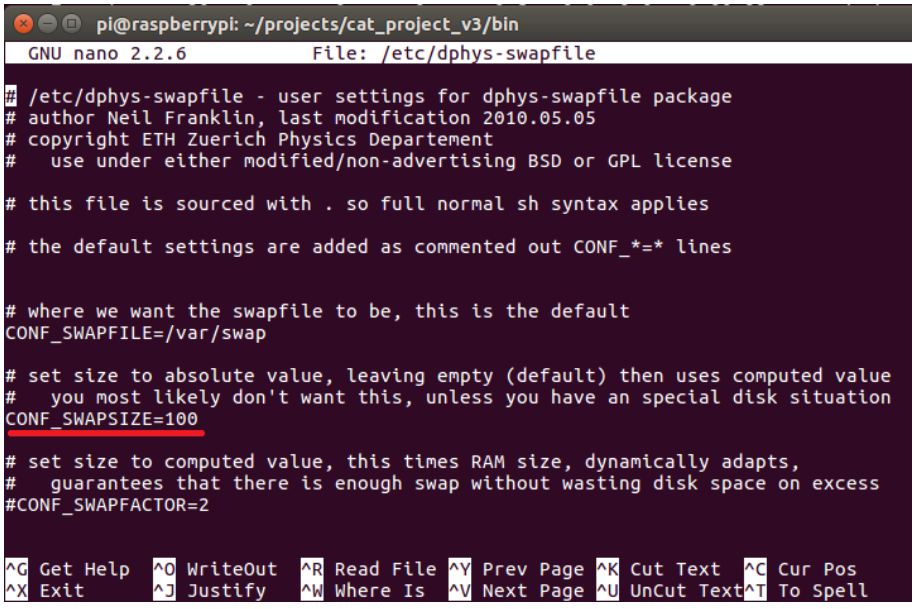

Figura B.1: Ficheiro no qual se define a quantidade de memória *swap* (variável sublinhada a vermelho)

Desativar o *swap* e de seguida, iniciar o *swap*, para ser atualizado o novo valor.

Passo 2: Ativação do swap
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start

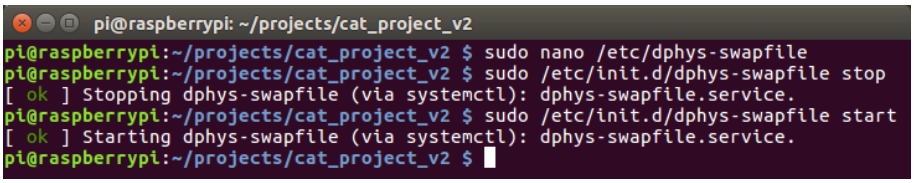

Figura B.2: Ativação da memória RAM adicional *swap*

B.2 *Open Natural Interaction*

Instalação de *packages* necessários.

Passo 1 : Instalação de *packages* necessários
sudo apt-get libusb-1.0-0-dev
sudo apt-get install libudev-dev
sudo apt-get install freeglut3-dev
sudo apt-get install graphviz
sudo apt-get install doxygen

Segue-se aquisição do código fonte associado à OpenNI2 para a diretoria pretendida.

Passo 2 : Aquisição do código fonte da biblioteca
mkdir OpenNI2 && cd OpenNI2
git clone https://github.com/occipital/OpenNI2.git

Posteriormente, as configurações para definir a arquitetura em causa.

Passo 3: Definição de variáveis para a compilação
sudo nano /OpenNI2/ThirdParty/PSCommon/BuildSystem/Platform.Arm
Adicionar Flags
CFLAGS += -march=armv7-a -mtune=cortex-a15 -mfpu=vfpv3 -mfloat-abi=hard

Por fim, a compilação do código fonte da OpenNI2.

Passo 4: Compilação do código fonte
PLATFORM=Arm
sudo make -j1

B.3 *Point Cloud Library*

Para a instalação desta biblioteca recomenda-se o acesso via SSH e o uso de um *swap* de 2GB. Este processo inicia com a instalação de um conjunto de *packages* necessários.

Passo 1 : Instalação de packages necessários
sudo apt-get install libflann-dev
sudo apt-get install libqt4-dev libvtk5-qt4-dev
sudo apt-get install gsl-bin libgsl0-dev

Posto isto, segue-se a aquisição do código fonte associado à PCL para a diretoria pretendida.

Passo 2 : Aquisição do código fonte da biblioteca
mkdir PCL && cd PCL
git clone https://github.com/PointCloudLibrary/pcl.git

De seguida, as configurações para definir a arquitetura em causa.

Passo 3 : Definição de variáveis para a compilação
sudo nano PCL/cmake/pcl_find_sse.cmake
Substituir
-march=native
por
-march=armv7-a
Definir
-mfloat-abi=hard -mfpu=vfpv3 -mtune=cortex-a53
sudo nano PCL/io/include/pcl/io/ply/byte_order.h
sudo nano PCL/common/include/pcl/PCLPointCloud2.h
Definir
#define BOOST_LITTLE_ENDIAN

Por fim, torna-se possível compilar o código fonte da PCL.

Passo 4 : Compilação do código fonte
mkdir build && cd build
cmake ..
make -j1
sudo make install

Referências

[1] Luís Filipe Pinto Cunha e Silva. Utilização de manipuladores em ambientes não estruturados, Julho 2010. Faculdade de Engenharia da Universidade do Porto.

[2] 3d machine vision – technical basics and challenges. Disponível em http://www.stemmer-imaging.co.uk/en/knowledge-base/3d-machine-vision/, acedido a última vez em 26 de Janeiro de 2017.

[3] Troy Hughes e Guillermo Vincentelli. A framework for localization and navigation on a raspberry pi, Abril 2016. Faculty of Worcester Polytechnic Institute.

[4] Miguel Angelo Baggio, Gabriel Kist, Roberta Francine Schmachtenberg, Luciano Porto de Lima, Fábio Lorenzi da Silva, e Jean da Rolt Joaquim. Padevi–protótipo de auxilio a deficientes visuais. *Revista de Empreendedorismo, Inovação e Tecnologia*, 1(1):45–57, 2015.

[5] Raspberry pi 3 model b. Disponível em https://www.raspberrypi.org/, acedido a última vez em 25 de Março 2017.

[6] Shuai Zheng, Jun Hong, Kang Zhang, Baotong Li, e Xin Li. A multi-frame graph matching algorithm for low-bandwidth rgb-d {SLAM}. *Computer-Aided Design*, 78:107 – 117, 2016. {SPM} 2016. doi:http://dx.doi.org/10.1016/j.cad.2016.05.009.

[7] A pi and a kinect autofocus a dslr. Disponível em http://www.i-programmer.info/news/194-kinect/10050-a-pi-and-a-kinect-autofocus-a-dslr.html, acedido a última vez em 15 de Janeiro de 2017.

[8] Angel D Sempere, Arturo Serna-Leon, Pablo Gil, Santiago Puente, e Fernando Torres. Control and guidance of low-cost robots via gesture perception for monitoring activities in the home. *Sensors*, 15(12):31268–31292, 2015.

[9] David G. Lowe. Object recognition from local scale-invariant features. Em *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, páginas 1150–, Washington, DC, USA, 1999. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=850924.851523.

[10] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, e Wolfram Burgard. NARF: 3d range image features for object recognition. *ResearchGate*, 2010. URL: https://www.researchgate.net/publication/260320178_NARF_3D_Range_Image_Features_for_Object_Recognition.

[11] Carsten Steger, Markus Ulrich, e Christian Wiedemann. *Machine Vision Algorithms and Applications*. Wiley, 2008.

[12] Eben Upton. *Learning computer architecture with raspberry PI*. John Wiley and Sons, Indianapolis, IN, 2016.

[13] Cortex-a53 processor. Disponível em https://www.arm.com/products/processors/cortex-a/cortex-a53-processor.php, acedido a última vez em 2 de Junho de 2017.

[14] R.P. Thomson e J.R. Robertson. Industrial manipulator for placing articles in close proximity to adjacent articles, Junho 16 1981. US Patent 4,273,506. URL: https://www.google.com/patents/US4273506.

[15] Z. Vafa e S. Dubowsky. On the dynamics of manipulators in space using the virtual manipulator approach. Em *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, páginas 579–585, Mar 1987. doi:10.1109/ROBOT.1987.1088032.

[16] Pedro Miguel Santos Tavares. Planeamento de trajetórias em manipuladores em ambientes industriais, Julho 2015. Faculdade de Engenharia da Universidade do Porto.

[17] Andry Maykol Pinto, Paulo Costa, Antonio P Moreira, Luís F Rocha, Germano Veiga, e Eduardo Moreira. Evaluation of depth sensors for robotic applications. Em *Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on*, páginas 139–143. IEEE, 2015.

[18] R. Szabó, A. Gontean, e A. Sfiraţ. Robotic arm control in space with color recognition using a raspberry pi. Em *2016 39th International Conference on Telecommunications and Signal Processing (TSP)*, 2016.

[19] Uwe Lauschner, Burkhard Igel, Lukas Krawczyk, e Carsten Wolff. Applying model-based principles on a distributed robotic system application. Em *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference on*, volume 2, páginas 893–897. IEEE, 2015.

[20] Ali Uroidhi, Ronny Mardiyanto, e Djoko Purwanto. Automatic navigation for a mobile robot with real time depth kinect sensor and map database. *JAVA Journal of Electrical and Electronics Engineering*, 14(1), 2016.

[21] Shreyas Shinde, Vaibhav Borle, e Ashwani Longjam. Abhikaha: Aerial collision avoidance in quadcopter using cloud robotics. *International Journal of Research In Science and Engineering*, 2016.

[22] J Sree Madhubala e A Umamakeswari. A vision based fall detection system for elderly people. *Indian Journal of Science and Technology*, 8:167, 2015.

[23] G Ahmed Zeeshan, R Sundar Guru, e Y Sreenivasa Reddy. Feature design scheme for kinect based hand written recognition. *International Journal of Engineering Science*, 2686, 2016.

[24] P. Beňo, F. Duchoň, M. Tölgyessy, P. Hubinský, e M. Kajan. 3d map reconstruction with sensor kinect: Searching for solution applicable to small mobile robots. Em *2014 23rd International Conference on Robotics in Alpe-Adria-Danube Region (RAAD)*, páginas 1–6, Sept 2014. doi:10.1109/RAAD.2014.7002252.

[25] André Filipe Morais Duarte. Reconhecimento de objetos baseado em visão artificial, Julho 2015. Faculdade de Engenharia da Universidade do Porto.

[26] "visual intelligence gives robotic systems spatial sense". Disponível em http://www.embedded-vision.com/platinum-members/embedded-vision-alliance/embedded-vision-training/documents/pages/robotics, acedido a última vez em 2 de Fevereiro de 2017.

[27] Carlos Astua, Ramon Barber, Jonathan Crespo, e Alberto Jardon. Object detection techniques applied on mobile robot semantic navigation. *Sensors (Basel, Switzerland)*, 14, 2014. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4029636/, doi:10.3390/s140406734.

[28] Z. Li, Y. Lou, Z. Li, G. Yang, e J. Gao. T2: A novel two degree-of-freedom translational parallel robot for pick-and-place operation. Em *IEEE ICCA 2010*, páginas 725–730, June 2010. doi:10.1109/ICCA.2010.5524340.

[29] Thorsten Gecks e Dominik Henrich. Human-robot cooperation: Safe pick-and-place operations, 2005.

[30] L Sciavicco e Bruno Siciliano. *Modelling and Control of Robot Manipulators*. Springer London, London, 2000. OCLC: 853261388. URL: http://dx.doi.org/10.1007/978-1-4471-0449-0.

[31] Hamed Sarbolandi, Damien Lefloch, e Andreas Kolb. Kinect range sensing: Structured-light versus time-of-flight kinect. *Computer Vision and Image Understanding*, 139:1 – 20, 2015. URL: http://www.sciencedirect.com/science/article/pii/S1077314215001071, doi:http://dx.doi.org/10.1016/j.cviu.2015.05.006.

[32] J. Schöning e G. Heidemann. Taxonomy of 3d sensors - a survey of state-of-the-art consumer 3d-reconstruction sensors and their field of applications. Em *Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)*, volume 3, páginas 194–199. SCITEPRESS, SCITEPRESS, 2016. doi:http://dx.doi.org/10.5220/0005784801920197.

[33] Documentation *Point Cloud Library*. Disponível em http://pointclouds.org/documentation/, acedido a última vez em 10 de Junho de 2017.