Universidade de Aveiro Departamento de Engenharia Mecânica
2014

Nuno Joel Marques dos Santos

Bin Picking de Objetos Polimórficos Convexos usando Perceção 3D

Nuno Joel Marques dos Santos

Bin Picking de Objetos Polimórficos Convexos usando Perceção 3D

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Licenciado Abílio Manuel Ribeiro Borges, Assistente convidado do Departamento de Engenharia Mecânica da Universidade de Aveiro e do Doutor Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro.

O Júri

Presidente

Prof. Doutor Jorge Augusto Fernandes Ferreira
Professor auxiliar da Universidade de Aveiro

Vogais

Doutor Luís André Freitas da Rocha
Investigador do Instituto de Engenharia de Sistemas e Computadores do Porto

Engenheiro Abílio Manuel Ribeiro Borges
Assistente convidado da Universidade de Aveiro

Agradecimentos

À minha família, em especial à minha mãe, que sempre me apoiaram e tornaram possível esta etapa na minha vida.
Um muito obrigado ao Professor Abílio Borges por ter aceite ser meu orientador, pela grande disponibilidade que sempre teve e por toda a ajuda dada.
Ao Professor Vítor Santos, pelas importantes linhas orientadoras que deu nas reuniões. Foi uma ajuda fundamental na fase mais difícil do trabalho.
Um agradecimento à MOTOFIL pelos manipuladores Fanuc disponibilizados, sem eles não era possível realizar este trabalho.
Aos meus amigos, com quem partilhei muitas horas de estudo, noites de trabalho e bons momentos de lazer.
Por último, mas nunca menos importante, à Nélia, minha namorada, por todo o apoio e dedicação. A sua presença no dia a dia tem sido fundamental no meu sucesso.
A todos um muito obrigado.

palavras-chave Bin picking, Perceção 3D, Kinect, Nuvem de pontos

resumo O *bin picking* é um processo de grande interesse na indústria, uma vez que permite maior automatização, aumento da capacidade de produção e redução dos custos. Este tem vindo a evoluir bastante ao longo dos anos e essa evolução fez com que sistemas de perceção 3D começassem a ser implementados.

Este trabalho tem como principal objetivo desenvolver um sistema de *bin picking* usando apenas perceção 3D. O sistema deve ser capaz de determinar a posição e orientação de objetos com diferentes formas e tamanhos, posicionados aleatoriamente numa superfície de trabalho. Os objetos utilizados para fazer os testes experimentais, são esferas, cilindros e prismas, uma vez que abrangem as formas geométricas existentes em muitos produtos submetidos a *bin picking*.

Após a identificação e seleção do objeto a apanhar, o manipulador deve autonomamente posicionar-se para fazer a aproximação e recolha do mesmo.

A aquisição de dados é feita através de uma câmara Kinect. Dos dados recebidos apenas são trabalhados os referentes à profundidade, centrando-se assim este trabalho na análise e tratamento de nuvem de pontos.

O sistema desenvolvido cumpre com os objetivos estabelecidos. Consegue localizar e apanhar objetos em várias posições e orientações. Além disso apresenta uma velocidade de processamento compatível com a aplicação em causa.

keywords Bin picking, 3D Perception, Kinect, Point Cloud

abstract

Bin picking is a process with major interest in industry, since it allows greater automation, increased production capacity and costs reduction. It has evolved greatly over the years and was that evolution that made possible that these 3D perception systems began to be implemented.
This study has as main objective to develop a bin picking system using just 3D perception. The system must be able to determine the position and orientation of objects with different forms and sizes, positioned randomly in a work area. The objects used to make experimental tests, are spheres, cylinders and prisms, since these cover the geometric forms existing in many products subjected to bin picking.
After the identification and selection of the object to pick, the manipulator must autonomously position itself to make the approximation and collect that same object.
The data acquisition is made through a Kinect camera. From the collected data, only the ones related to the depth are studied, focusing this study in point cloud handling and analysis.
The developed system fulfills the established objectives. It manages to locate and grab objects in several positions and orientations. Furthermore it presents a velocity of processing compatible with the concerned application.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Enquadramento

A venda de robôs industriais está continuamente em crescimento. Segundo as projeções da IFR[1], as vendas até ao final de 2014 superam as 179.000 unidades vendidas no ano de 2013 (Figura 1.1).

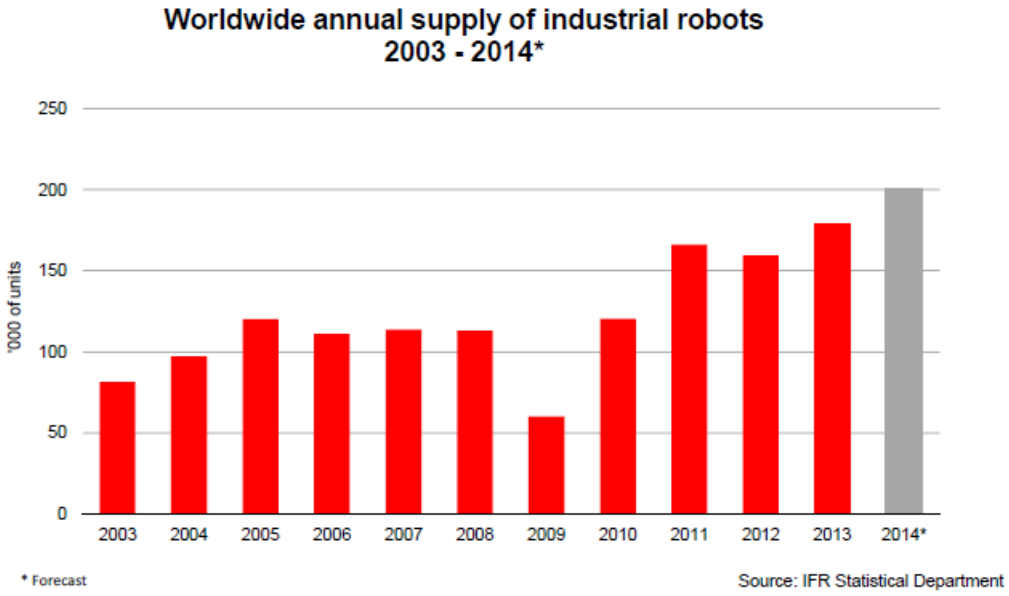


Figura 1.1: Projeção de vendas para 2014 face aos anos anteriores [1]

Estes números demonstram o aumento da automatização implementada nas empresas em todo o globo.

[1] IFR - International Federation of Robotics

Contribuem para este crescimento fatores como: a competitividade que exige um aumento da produtividade; o crescimento do consumo, que requer uma capacidade de produção cada vez maior; e o baixo ciclo de vida, que necessita de sistemas flexíveis. [1].

Muitos destes robôs são aplicados em processos que necessitam de decisões para poderem atuar. No entanto, estas não são possíveis de acontecer sem a existência de equipamentos auxiliares, nomeadamente sensores, para controlar as ações a realizar [10].

A utilização de câmaras de visão 2D permite que os sistemas sejam capazes de analisar o espaço de trabalho, para dar ordens aos robôs, mediante as informações recebidas. Desta forma, a atuação dos robôs deixa de ser um conjunto de movimentos previamente programados, passando esses movimentos a depender das decisões tomadas pelo programa.

Contudo, esta medida por si só, não é suficiente para tornar os robôs capazes de realizar outro tipo de tarefas, como é o caso do *bin picking*.

Bin picking é o termo utilizado para definir o processo de apanhar um objeto especifico entre um conjunto de objetos colocados de forma aleatória numa zona de trabalho.

Este trabalho monótono, repetitivo e pouco ergonómico é normalmente realizado por pessoas, existindo portanto uma necessidade de as substituir por um sistema que seja capaz de o executar continuamente e sem falhas [11].

A continua evolução da tecnologia permitiu que a aquisição de dados para este fim começasse a ser feita com dispositivos 3D, em alternativa às câmaras 2D conjugadas com sensores de distância. É neste ponto que este trabalho se enquadra, na identificação de objetos sujeitos a *bin picking* apenas utilizando dados provenientes de um dispositivo 3D.

1.2 Objetivos

O objetivo final deste trabalho é o desenvolvimento de um sistema que seja capaz de identificar e manipular um determinado tipo de objetos colocados aleatoriamente numa superfície. Este objetivo pode dividir-se em dois mais específicos, correspondendo a duas partes deste trabalho.

O primeiro é a identificação dos objetos que estão posicionados e orientados de forma aleatória. Nesta primeira parte, o sistema, além de identificar os objetos dispostos na área de trabalho, tem de escolher o melhor candidato a ser manipulado, e obter a sua posição e orientação.

O segundo é a correta manipulação dos objetos por parte do robô, que os irá agrupar consoante a sua geometria e dimensão.

Os dados devem ser obtidos por meio de um dispositivo 3D em tempo real, sendo este o único meio de informação e o único tipo de dados a tratar.

É fundamental que o sistema desenvolvido tenha um processamento rápido, dado o tipo de aplicação a que se destina.

Uma vez que esta tecnologia já está disponível no mercado, pretende-se com este trabalho mostrar a viabilidade de uma solução alternativa e mais económica.

É também importante que o trabalho desenvolvido seja partilhado para poder ser utilizado em outras aplicações ou projetos. Para isso, é fundamental a utilização de um ambiente de desenvolvimento comum e a utilização de bibliotecas e software *open-source*.

1.3 Estrutura do trabalho

Para poder cumprir os objetivos acima referidos esta dissertação encontra-se organizada nos seguintes capítulos:

- Estado de arte - São descritos os sistemas de aquisição de dados 3D, apresentados exemplos de aplicações industriais e feita uma breve descrição de trabalhos académicos relacionados.
- Hardware e Software - Aqui é descriminado todo o equipamento e ferramentas utilizados. Também é explicado o funcionamento geral do programa, ou seja, de que forma se desenrola a comunicação entre todos os equipamentos.
- Identificação e Recolha dos Objetos - É descrita de forma detalhada como é realizada a identificação e seleção dos objetos a recolher, bem como a sua recolha.
- Testes e Análise de Resultados - Este capítulo contém as tabelas e gráficos com os dados referentes aos ensaios realizados, bem como a sua análise.
- Conclusões - Este último capítulo contém as ilações obtidas com a realização deste trabalho.

Capítulo 2

Estado da Arte

Sendo o *bin-picking* um processo associado à robótica, existe uma diversidade de tecnologia, em constante desenvolvimento, que vai sendo integrada neste processo de forma a desenvolvê-lo. Neste capitulo serão apresentados alguns sistemas de aquisição de dados 3D, alguns exemplos práticos das tecnologias utilizadas em *bin picking* e ainda alguns trabalhos académicos relacionados.

2.1 Sistemas de aquisição de dados 3D

2.1.1 Estéreo

A tecnologia estéreo utiliza duas câmaras de maneira a obter duas perspetivas diferentes da mesma vista, à semelhança da visão humana. É necessária uma calibração das câmaras para alinhar a informação dos píxeis e extrair os valores de profundidade. Geralmente, as câmaras estão espaçadas de uma curta distância e são montadas quase em paralelo como se pode observar no esquema simplificado da Figura 2.1.

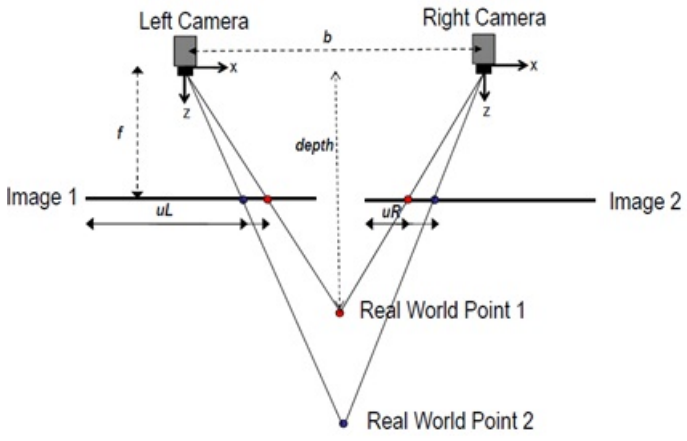


Figura 2.1: Esquema simplificado de visão estéreo [2]

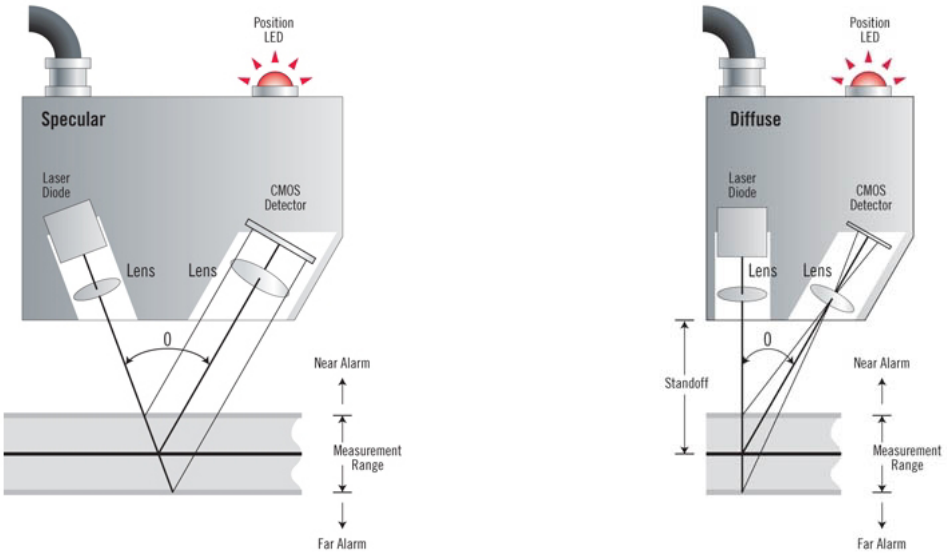
Estes sistemas são mais adequados a aplicações onde as câmaras estejam fixas, para evitar distúrbios. Algumas das aplicações em que estes sistemas se podem integrar são a navegação, onde podem medir o tamanho e distância de obstáculos, a robótica industrial, como é o caso do *bin picking*, e a inspeção automatizada, para a deteção de defeitos que apenas com visão 2D seriam impossíveis de detetar[12]. Uma das vantagens em utilizar este sistema é o facto de ser não invasivo, contudo, o seu bom funcionamento depende das condições de iluminação.

2.1.2 Triangulação laser

Os sistemas de triangulação laser são constituídos por uma fonte de luz laser e um detetor PSD[1] ou CMOS[2]/CCD[3].

O feixe laser é projetado sobre o objeto que se pretende medir e parte desse feixe é refletido para a ótica do detetor. O sinal do detetor é usado para determinar a distância ao objeto.

Na Figura 2.2 estão apresentadas duas configurações em que estes sistemas são usados. Estas configurações dependem do tipo de superfície que se pretende medir. Para superfícies espelhadas é necessário utilizar a configuração visível na Figura 2.2a, em outro tipo de superfícies a configuração utilizada é visível na Figura 2.2b. [3]



(a) *Specular* (b) *Difuse*

Figura 2.2: Principio da triangulação laser [3]

[1]PSD - Position sensitive device
[2]CMOS - Complementary metal-oxide semiconductor
[3]CCD - Change-couple device

2.1.3 Tempo de voo

As câmaras com tecnologia de tempo de voo (TOF[4]) produzem uma imagem de profundidade, em que cada píxel tem guardado um valor de distância. São usadas em muitas aplicações como navegação, reconstrução 3D e interação entre humanos e máquinas.

O funcionamento base desta tecnologia, como é ilustrado na Figura 2.3, é a emissão de uma onda de luz infravermelha sobre um objeto e deteção da mesma, após reflexão, por parte do sensor. A medição da diferença de fase entre a luz emitida e a refletida permite calcular a distância a que o objeto se encontra.[4]

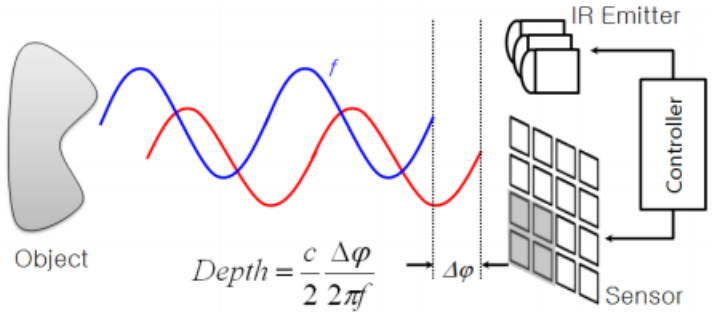


Figura 2.3: Principio de funcionamento da tecnologia TOF [4]

2.1.4 Projeção de padrões de luz

O modo de funcionamento destas câmaras baseia-se na projeção e observação de padrões como está ilustrado na Figura 2.4 . O projetor de luz infravermelha emite um padrão conhecido sobre os objetos que é observado pela câmara de luz infravermelha. Como o padrão observado está deformado em relação ao projetado, é possível obter o valor da distância entre o objeto e a câmara com base nessa deformação.

2.2 Exemplos de *Bin Picking* usando perceção 3D

2.2.1 *Liebherr Automation*

A *Liebherr Automation* oferece um sistema de manipulação robótico com software 3D capaz de remover itens dispostos aleatoriamente num recipiente e colocá-los com precisão numa máquina ou linha de montagem (Figura 2.5).

O sistema de laser 3D não é afetado por luz exterior e consegue reconhecer peças pretas, castanhas ou com ferrugem.

O braço robótico possui garras especialmente desenvolvidas com eixos adicionais para permitir um posicionamento preciso, evitando desta forma possíveis colisões.

[4]TOF - Time-of-flight

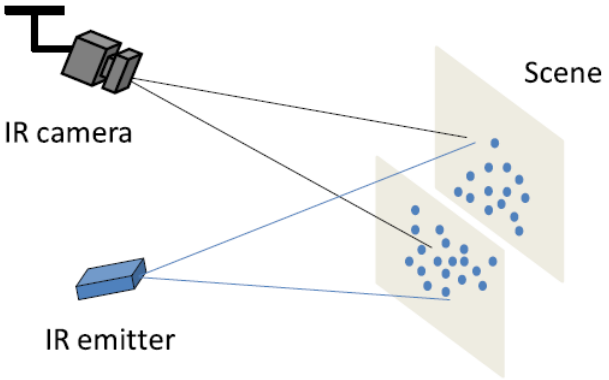


Figura 2.4: Principio de funcionamento da tecnologia de padrões projetados [5]

O produto apresentado aumenta a produtividade e eficiência. É adequado para manipular objetos com massa entre 1 e 50 kg. Está principalmente direcionado para as indústrias automóvel, aeroespacial e de maquinagem em geral. [6]

Figura 2.5: Sistema de *bin picking* da *Liebherr Automation* [6]

2.2.2 *Universal Robotics*

A *Universal Robotics* desenvolve software, denominado *Neocortex*, que permite às máquinas aprender com a experiência. Este capacita-as de reagir e se adaptar ao meio envolvente, bem como realizar tarefas dispendiosas, perigosas ou difíceis para as pessoas.

A aplicação *Neocortex Random Bin Picking* pode ser integrada em novos robôs com 6 eixos ou robôs delta com 3 ou 4 eixos.

Dependendo da sensibilidade requerida, podem ser adicionados sensores de proximidade, aceleração, ou outros. A informação obtida corresponde à posição (x,y,z) e à orientação (Rx,Ry,Rz) das peças que se encontram na área de trabalho.

A aplicação processa os dados 3D recebidos, seleciona a peça que deve ser apanhada, envia a posição e orientação em que esta se encontra para o controlador do robô e monitoriza os seus movimentos. Consegue ainda ser treinada para reconhecer uma larga variedade de objetos (Figura 2.6). [7]

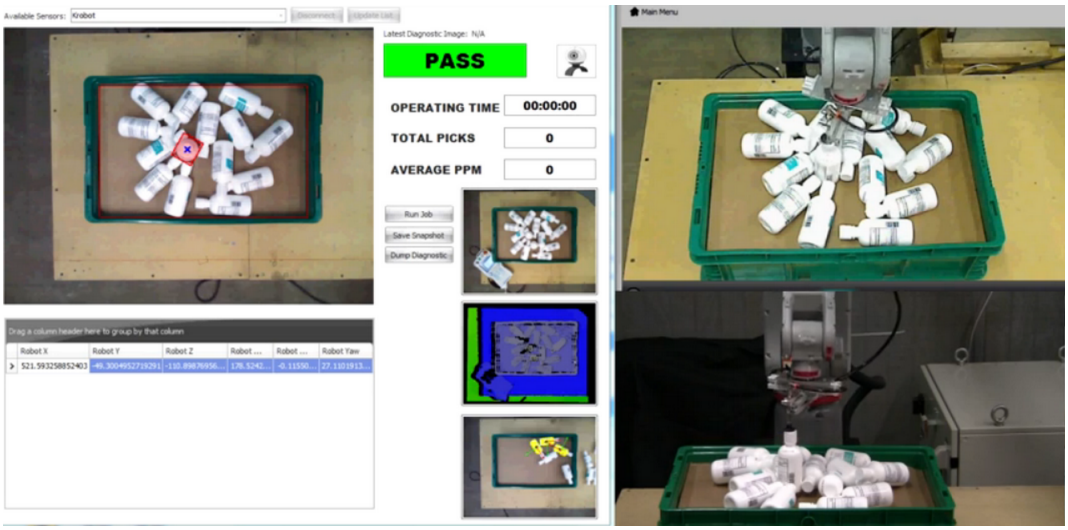


Figura 2.6: Sistema de *bin picking* da *Universal Robotics* [7]

2.3 Trabalhos académicos relacionados

2.3.1 *Development and Evaluation of a Kinect based Bin-Picking System*

Neste trabalho é estudada a possibilidade de utilizar a câmara Kinect para o reconhecimento de objetos num sistema de *bin picking*.

O método de reconhecimento utilizado pelo autor baseia-se obtenção de correspondência entre o objeto observado e um objeto de referência guardado em disco.

Para criar um objeto de referencia foi primeiramente utilizado um modelo CAD convertido para nuvem de pontos. Esta opção verificou-se inviável uma vez que a escala do modelo era diferente da escala dos objetos observados e as resoluções de ambos eram muito distintas. Como alternativa, os modelos de referência foram obtidos com a Kinect.

O algoritmo utilizado para fazer a identificação foi o ICP[5] , que faz um alinhamento do objeto com os vários modelos guardados e atribui um valor a cada. Como resultado, o valor obtido mais próximo de zero corresponde ao melhor alinhamento encontrado.

Os resultados obtidos demonstram que este método de reconhecimento é demasiado lento para ser utilizado numa aplicação de *bin-picking*. [13]

[5]ICP - *Iterative Closest Point*

2.3.2 *Construction of a 3D Object Recognition and Manipulation Database from Grasp Demonstrations*

Neste trabalho o autor apresenta uma solução para construção de modelos de objetos para reconhecimento 3D e manipulação de objetos. É utilizado o recurso a uma base de dados, que contém nuvens de pontos e informações sobre manipulação e caraterísticas dos objetos. Sendo as nuvens da base de dados geradas através de um algoritmo iterativo de registo de nuvens de pontos.

Um dos vários contributos deste trabalho é o sistema de reconhecimento de objetos através de nuvens de pontos, com recurso a modelos 3D. Estes modelos são obtido através da junção de nuvens, recorrendo à caraterística SIFT[6], que permite unir várias nuvens, referentes ao mesmo objeto, recolhidas de pontos de vista distintos.

Antes da criação do modelo são cálculadas um conjunto de métricas para caraterizar a potencial correspondência entre as nuvens. Com base nestas métricas e num conjunto de registos de nuvens manual, com sucessos e insucessos, foi criada uma árvore de decisão que prevê quais as nuvens que são correspondentes.

A utilização deste método apresenta 81% de classificações corretas com 10% de falsos positivos.

O processo de reconhecimento é semelhante ao utilizado na construção de modelos, pois é utilizada a mesma sequência de operações. São cálculadas as métricas e medido o erro entre a nuvem testada e os modelos da base de dados. O modelo que obtiver o menor valor de erro é o selecionado.

Apesar da potencialidade da solução proposta, o autor refere que não é suficientemente robusta para uma utilização prática, pois uma incorreta identificação de objetos leva a uma perigosa propagação de erros. [14]

[6]SIFT - Scale Invariant Feature Transform

Capítulo 3

Hardware e Software

Para realizar este trabalho é necessária a utilização de alguns equipamentos, nomeadamente um computador, um manipulador robótico e um sistema de visão 3D . A par destes, é também necessária a instalação de software para ser possível desenvolver e executar a aplicação.

Ao longo deste capítulo será apresentado e detalhado todo o hardware e software utilizado.

3.1 ROS (*Robot Operating System*)

Robot Operating System é uma infraestrutura flexível que ajuda ao desenvolvimento de software para robôs. É um conjunto de bibliotecas, ferramentas e convenções que pretendem simplificar a criação de comportamentos complexos e robustos nos robôs.

A sua criação foi resultado da necessidade de interligar desenvolvimentos feitos por diferentes laboratórios, encorajando assim, a colaboração entre eles.[15]

A versão mais recente do ROS é a Indigo Igloo, contudo a utilizada neste trabalho foi a Hydro Medusa.

3.1.1 Modo de Funcionamento

O funcionamento do ROS baseia-se na publicação e subscrição de mensagens em tópicos por parte dos nodos, como ilustra a Figura 3.1.

Os nodos são programas executáveis que fazem parte de um pacote ROS. Estes podem subscrever e publicar mensagens num ou mais tópicos, bem como fornecer ou utilizar um serviço.

Os tópicos são canais de comunicação entre nodos, podendo ter múltiplos publicadores e subscritores. Neles são trocadas mensagens que podem ser standard ou personalizadas, consoante a aplicação.

Quando se inicia o ROS além da criação dos nodos, são também iniciadas várias dependências, bibliotecas e sistema de comunicação entre eles. Durante o seu funcionamento, sempre que um nodo publica uma mensagem num tópico, todos os nodos que estejam subscritos a esse tópico recebem a mensagem.

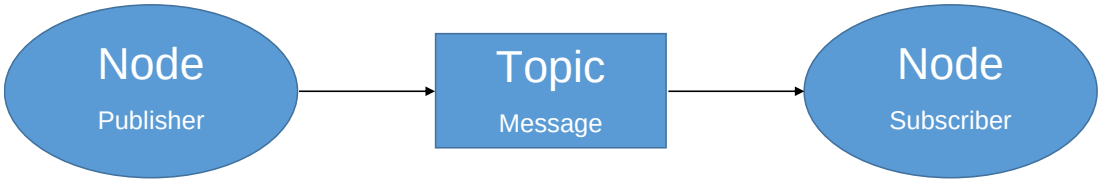


Figura 3.1: Esquema de comunicação do ROS

3.2 PCL (*Point Cloud Library*)

Point Cloud Library é uma biblioteca de código aberto que incorpora um extenso número de algoritmos para processar nuvens de pontos nD e geometrias 3D. A organização em módulos facilita o seu contínuo desenvolvimento e utilização. Cada um contém inúmeras classes e funções estruturadas para que a sua utilização seja simples. Dos vários módulos existentes, os utilizados para a aplicação final foram:

- common - contém as estruturas de dados e métodos comuns utilizados por grande parte da biblioteca PCL. Estas estruturas incluem a classe *PointCloud* e diversos tipos de pontos que são usados para representar pontos, normais às superfícies, valores RGB, descritores, entre outros. Nele também estão incluídas várias funções para calcular distâncias, médias e transformações geométricas entre muitas outras.
- filters - contém mecanismos para remoção de ruído e pontos deslocados. É neste módulo que estão por exemplo os filtros *PassThrough* e *VoxelGrid* utilizados para redução do número de pontos de uma nuvem.
- kdtree - fornece a estrutura de dados denominada kd-tree, que permite fazer partição de espaço, ou seja, armazenar um conjunto de pontos k-dimensionais numa estrutura em árvore que torna eficaz tanto a procura em intervalos como a procura de vizinhos mais próximos.
- segmentation - contém algoritmos para segmentar uma nuvem de pontos em grupos distintos. Estes algoritmos são adequados para processamento de nuvens de pontos composta por regiões espacialmente isoladas. Como resultado, a nuvem é dividida por essas regiões podendo cada parte ser processada de forma independente.
- visualization - este módulo foi criado para permitir uma visualização rápida dos resultados dos algoritmos aplicados às nuvens de pontos.

A PCL está disponível para os sistemas Windows, MacOS, Linux e Android. Uma vez que está integrada no ROS para poder ser utilizada em projetos de robótica não foi necessária a sua instalação. [16]

3.3 O Dispositivo Kinect

A câmara Kinect é um sensor desenvolvido pela *Microsoft* para a consola XBox 360. Está equipada com um motor, quatro microfones, uma câmara RGB e um sensor de profundidade, composto por um projetor e câmara IR (Figura 3.2).

Figura 3.2: Dispositivo Kinect [8]

Este dispositivo tem capacidade para capturar imagens a cores com resolução de 1280×960 a 12 FPS[1], 640×480 a 15 FPS ou 640×480 a 30 FPS. No caso das imagens de profundidade é possível obter a resolução de 640×480, 320×240 ou 80×60 com um *frame rate* de 30 FPS.

As suas limitações estão relacionadas com o sensor de profundidade. Em materiais com superfícies espelhadas é impossível obter a sua distância, uma vez que a luz emitida pelo projetor IR não é refletida para a câmara (reflexão especular). Por outro lado, materiais demasiado difusos não conseguem refletir luz suficiente para a câmara e, dessa forma, também não é possível determinar a sua distância. A sua utilização em ambiente exterior também está condicionada, pois a luz solar provoca interferência com o sensor IR.

Uma vez que esta câmara foi desenvolvida para se conectar a uma consola de jogos, não é possível fazer a sua ligação direta a um computador. É necessário utilizar um adaptador que tem como função fornecer energia aos vários componentes da câmara e permitir ligá-la ao computador através de porta USB.

A utilização deste dispositivo deve-se ao seu baixo custo. Além disso, os drivers necessários para a conexão ao computador já estão integrados no ROS, não sendo necessário adquirir software adicional.

Apesar da vasta quantidade de dados que é possível obter através deste dispositivo, apenas foram processados os dados referentes ao sensor de profundidade.

[1]FPS-Frames por segundo

3.3.1 Calibração da câmara

Apesar de não ser obrigatoriamente necessário calibrar a câmara, uma vez que o driver *openni_ camera* fornece por defeito modelos com valores de distância focal precisos, foi realizada a calibração (Figura 3.3), para garantir que os valores obtidos eram corretos.

Para efetuar a calibração recorreu-se à ferramenta que está disponível no ROS executando o nodo cameracalibrator.py e seguiram-se as instruções disponibilizadas em [17].

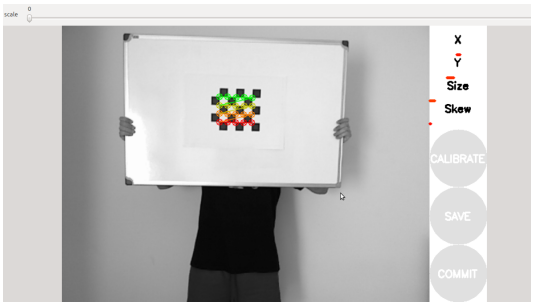


Figura 3.3: Calibração da câmara

3.3.2 Suporte

A colocação da câmara no espaço é um fator importante para a correta identificação dos objetos. Esta podia ser colocada no braço do robô ou num plano superior à mesa. A primeira opção verificou-se menos vantajosa uma vez a câmara ia limitar a capacidade de manobra do robô. Por outro lado, dado que o sensor 3D necessita de estar a uma distância superior a 500mm dos objetos e as dimensões do robô são reduzidas, não haveria campo de visão suficiente para trabalhar. Optou-se então por colocar a câmara num plano superior à mesa, garantindo assim a visualização dos objetos.

Para poder fixar a câmara na posição pretendida adquiriu-se um suporte universal e fez-se uma base de ligação à câmara. Na Figura 3.4 é possível visualizar a disposição da câmara, bem como toda a estrutura de suporte.

Figura 3.4: Estrutura de suporte da câmara

3.4 Robô FANUC

O robô é responsável pela manipulação dos vários objetos colocados sobre a mesa. Pretende-se que este tenha flexibilidade para se posicionar nas mais diversas configurações de modo a recolher corretamente os objetos.

Neste trabalho utilizou-se um Robô FANUC LR 200 iD/30 iB Mate (Figura 3.5). O robô é composto por 6 juntas, tem um alcance de 717mm, capacidade para suportar uma carga máxima de 7kg e atingir uma velocidade superior a 4 $m.s^{-1}$ [9]. Na Figura 3.6 estão descriminadas todas as dimensões do robô bem como o seu espaço de trabalho. O robô está equipado com uma pinça de vácuo que permite apanhar objetos com diferentes formas, nomeadamente cilindros, esferas e prismas.

Figura 3.5: Robô Fanuc utilizado

3.5 Computador

O computador é o equipamento central deste trabalho. A ele estão ligados todos os dispositivos e é nele que toda a informação é processada. A velocidade com que os dados são analisados e extraído um resultado é influenciada pela máquina utilizada.

Neste trabalho foi utilizado para o desenvolvimento e execução do programa um SONY VAIO VPCEB4Z1E com as caraterísticas da Tabela 3.1

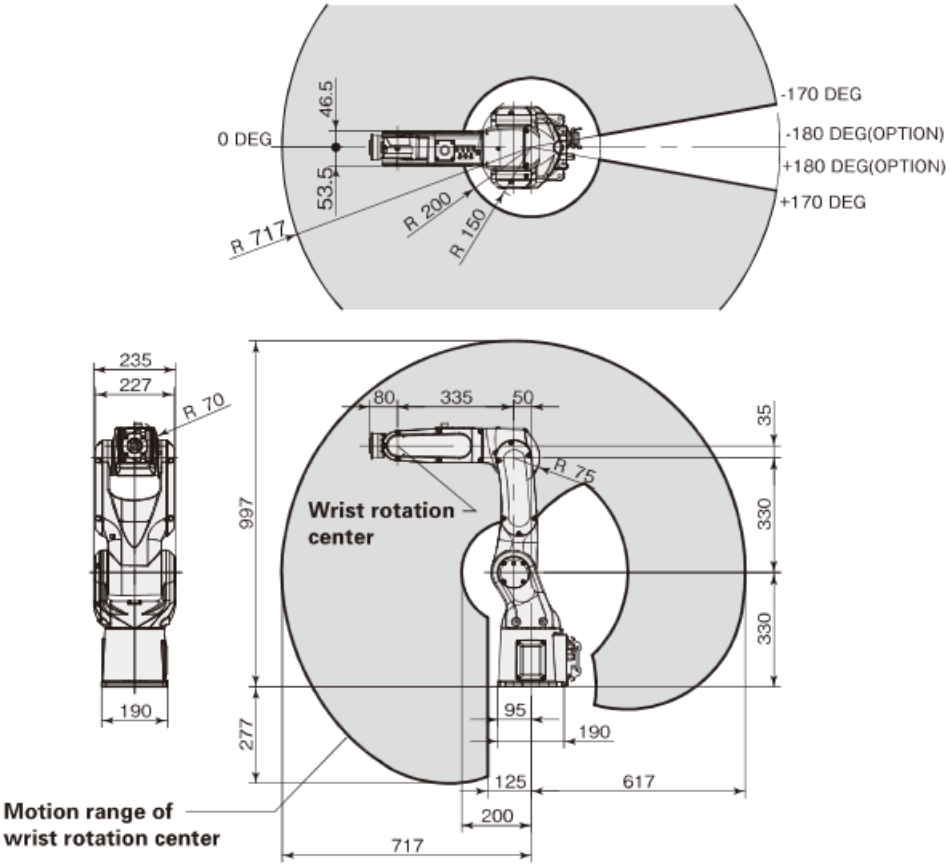


Figura 3.6: Dimensões do robô [9]

Tabela 3.1: Caraterísticas do computador

Processador	Intel Core i5-480M Dual Core
Frequência	2.66 GHz
Memória RAM	4 GB
Tipo de Memória	DDR3-1066
Disco Rígido	HDD 500 GB
Placa Gráfica	ATI Mobility Radeon HD 5650
Memória Gráfica	2743 MB (1024 MB dedicados)
Sistema Operativo	Ubuntu 12.04.4 Desktop (64-bit)

3.6 Arquitetura do Software

Para reconhecimento dos objetos, determinação da sua posição e orientação, e posterior manipulação é necessário criar uma estrutura de software. Esta estrutura deve ser capaz de estabelecer ligação com os diversos aparelhos, sendo necessário analisar os dispositivos e o tipo de comunicação em que estes operam, para ser possível comunicar com todos eles de uma forma funcional.

Nesta secção são apresentados os nodos que foram criados, descrita a sua função no programa e de que forma comunicam entre si. É demonstrada a forma como são adquiridos os dados provenientes da câmara e qual o seu conteúdo. São também apresentados os programas criados na consola e de que forma são enviadas as ordens para o controlador do robô. Por último é apresentada de que forma o utilizador pode dar ordens e visualizar a troca de informação.

A câmara Kinect é um dispositivo de perceção, portanto a sua única função é o fornecimento de dados (*input*). Por outro lado, o robô Fanuc é um atuador, ou seja, recebe informações do PC para executar uma tarefa (*output*). No entanto, existe a necessidade de conhecer as coordenadas em que este se encontra a cada momento. Esta necessidade faz com que o robô seja também um dispositivo de entrada (*input*). Na Figura 3.7 está representado o esquema de comunicação entre a câmara, PC e robô.

Figura 3.7: Comunicação entre dispositivos

Analisado o fluxo de dados entre dispositivos, é necessário estruturar o programa para processar os dados recebidos e enviar as ordens necessárias.

O programa é composto por três nodos que comunicam entre eles através de três tópicos.

O nodo */Kinect* é o responsável pelo tratamento de dados. Este subscreve um tópico que contém os dados da câmara e processa-os. Subscreve ainda mais dois tópicos: o tópico */keyboard*, onde são publicadas letras que se traduzem em ordens, como iniciar ou parar o processamento dos dados; e o tópico */request* onde são feitos os pedidos de coordenas. Este nodo publica as suas mensagens, que contêm várias informações como coordenadas de um objeto ou a indicação da inexistência de objetos, no tópico */reply*.

O nodo */Fanuc* é o que comunica com o controlador do robô, enviando e recebendo informação. É o responsável por enviar coordenadas e ler a posição em que o manipulador se encontra. Também subscreve dois tópicos: o tópico */keyboard* já referido anteriormente, e o tópico */reply* onde é publicada informação por parte do nodo */Kinect*. Este último

tópico é atualizado sempre que o nodo */Fanuc* faz um pedido, ou seja, cada vez que publica uma mensagem no tópico */request*.

Na Figura 3.8 está representado um esquema que resume todas as ligações entre os vários nodos que constituem o programa.

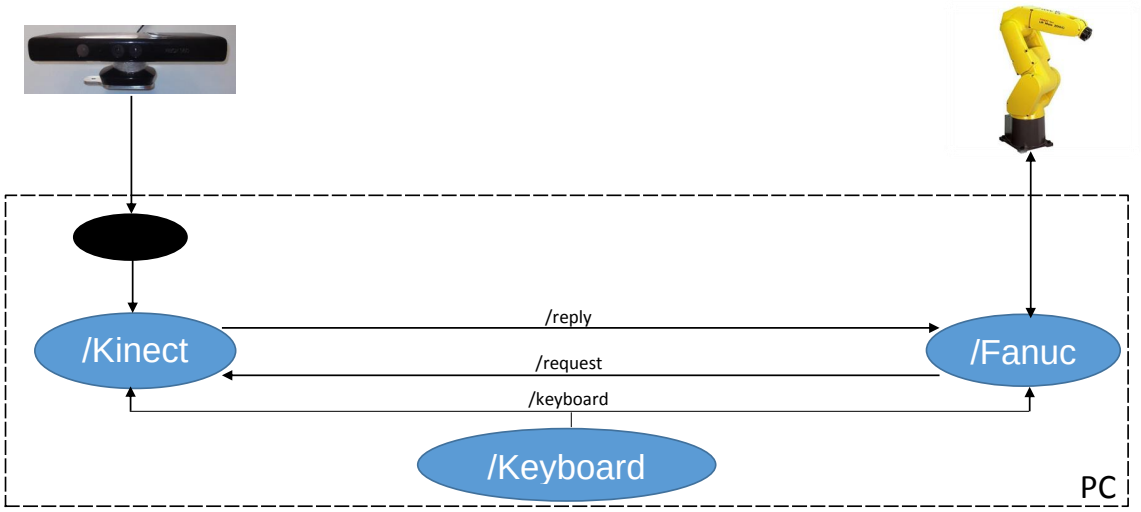


Figura 3.8: Comunicação entre nós

O nodo */Keyboard* tem a função de publicar no tópico */keyboard* as letras que são pressionadas no teclado do computador. Cada letra tem associada uma determinada ordem como se pode ver na Tabela 3.2.

Tabela 3.2: Lista de comandos

Letra	**Ordem**	
	/Kinect	**/Fanuc**
k	Iniciar/Parar processamento de dados	
f		Iniciar/Parar robô
s		Começar ciclo
c		Realizar calibração

3.6.1 Aquisição e tratamento de dados

Como já foi referido, o dispositivo utilizado para aquisição de dados é a câmara Kinect.

Uma vez que o ROS já tem incluídos os drivers da câmara, só é necessário incluir no ficheiro *launch* um conjunto de argumentos que vão permitir ligar a câmara e ter acesso aos seus dados. Este ficheiro de formato XML permite lançar vários nodos em simultâneo. No Anexo C encontra-se o ficheiro *launch* criado.

Na Figura 3.9 é possível visualizar os principais nodos, representados com ovais, que são criados quando a câmara é ligada.

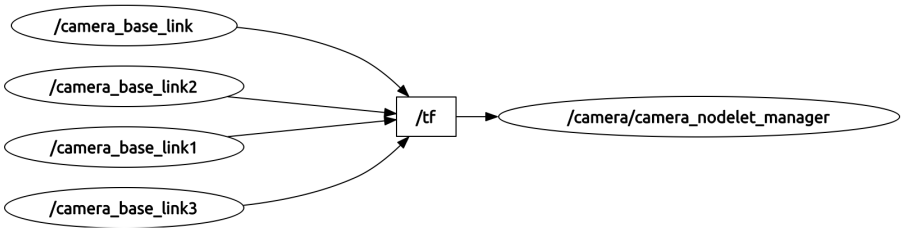


Figura 3.9: Nós referentes ao funcionamento da câmara

Estando a câmara em funcionamento é necessário fazer a leitura dos dados que ela fornece. Como existem vários sensores neste dispositivo, é possível obter diferentes tipos de dados.

Para este trabalho pretende-se obter a informação relativa à profundidade. Existem dois tópicos possíveis de subscrever cujas mensagens contêm esta informação, são eles: */camera/depth/points* e */camera/depth_ registered/points*. Em ambos os tópicos o tipo de mensagem tem a designação *sensor_ msg/PointCloud2*, no entanto, o primeiro contém apenas dados de posição enquanto que o segundo, além destes, contém também dados relativos à cor de cada píxel.

Como o objetivo do trabalho incide apenas sobre o tratamento de dados 3D, o tópico a usar poderia ser o */camera/depth/points*. Contudo, para facilitar a visualização dos objetos no ecrã recorrendo à cor, utilizou-se o tópico */camera/depth_ registered/points*.

Com a subscrição do tópico referido por parte do nó */Kinect*, responsável pelo tratamento de dados, um conjunto adicional de tópicos é criado como se pode observar na Figura 3.10. É possível verificar ainda a sequência de mensagens desde os sensores da câmara até ao nó */Kinect*.

3.6.2 Comunicação com o robô

Como referido anteriormente, o nodo */Fanuc* é o responsável pela troca de mensagens com o controlador do robô. Esta ligação, PC/Controlador, é feita por um cabo *Ethernet*, sob os protocolos de comunicação TCP/IP (Figura 3.11), onde o IP do controlador é o 192.168.0.231 e a porta de comunicação é a 4900.

Para que as ordens enviadas para o robô sejam executadas é necessário colocar em funcionamento a aplicação servidora *RobCom* [18] através da consola.

A utilização desta aplicação permite um fácil controlo do robô sem que exista a necessidade de adquirir software para o controlador ou para o computador.

Esta aplicação tem uma linguagem própria designada *robCOMM language*, que define comportamentos do robô e permite controlá-lo a partir de um dispositivo remoto. As instruções do *robCOMM* utilizadas encontram-se na Tabela 3.3

Durante o funcionamento do programa desenvolvido, sempre que se estabelece uma comunicação com o robô, a sequência de mensagens trocadas é a mesma. Inicia-se com o

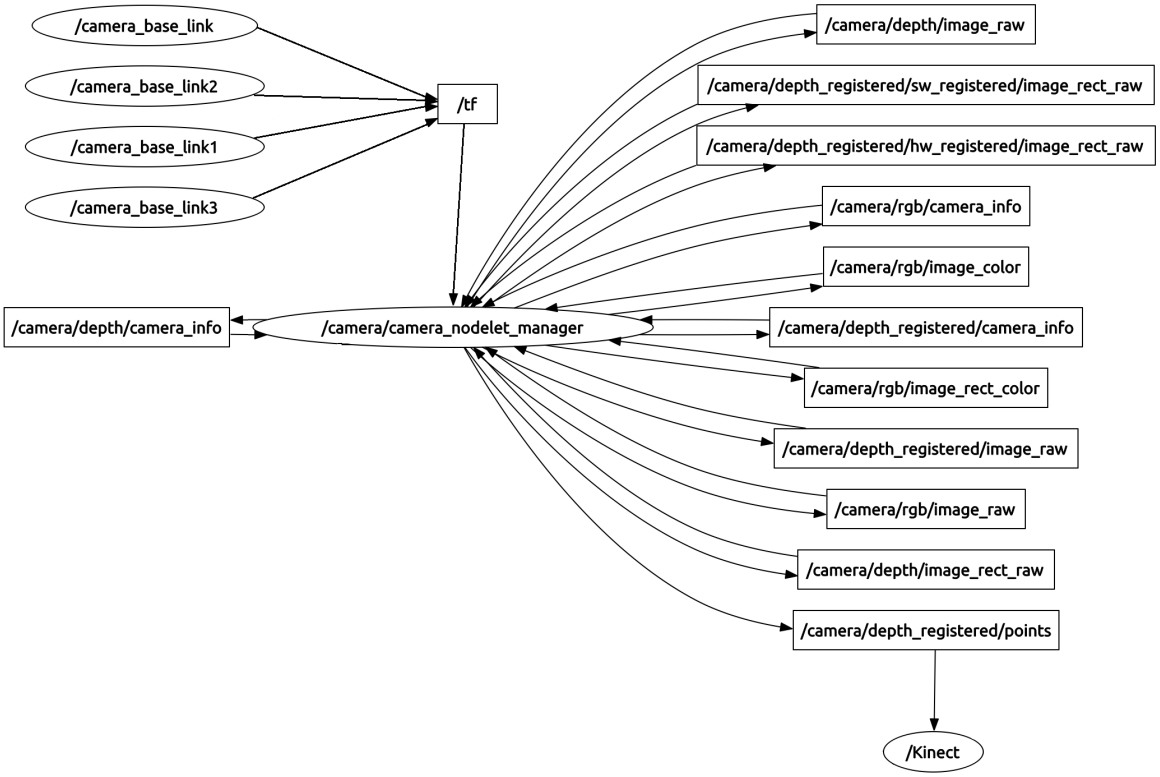


Figura 3.10: Troca de mensagens

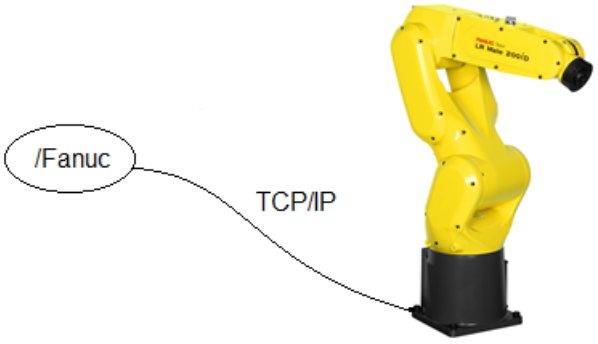


Figura 3.11: Comunicação com robô

envio de coordenadas e configurações de junta para registos do controlador. De seguida é enviada ordem para executar um dos vários programas criados na consola. Por último é controlada a posição em que se encontra o manipulador.

Cada um dos programas criados na consola tem uma sequência de movimentos e ações que depende da tarefa a realizar. Em 3.1 está como exemplo o código correspondente ao programa NUNOC. Os restantes programas descritos na Tabela 3.4 encontram-se no Anexo A.

Tabela 3.3: Lista de instruções do *robCOMM* utilizadas

Instrução	**Função**
SETREG	Escreve em registos
RUNTPP	Executa um programa TP
GETCRCPOS	Obter posição atual do ponto central da ferramenta

$$
\begin{aligned}
1 &: JPR[70 : robCOMM]100\%FINE;\\
2 &: JPR[71 : robCOMM]100\%FINE;\\
3 &: Closehand1;\\
4 &: LPR[72 : robCOMM]4000mm/secFINE;\\
5 &: LPR[71 : robCOMM]4000mm/secFINE;\\
6 &: JPR[73 : robCOMM]100\%FINE;\\
7 &: JPR[74 : robCOMM]100\%FINE;\\
8 &: LPR[75 : robCOMM]4000mm/secFINE;\\
9 &: Openhand1;\\
10 &: LPR[74 : robCOMM]4000mm/secFINE;\\
11 &: JPR[70 : robCOMM]100\%FINE;
\end{aligned}
\tag{3.1}
$$

Tabela 3.4: Programas desenvolvidos para movimentar o robô

Programa	**Descrição**
NUNOPOS	Ir para uma posição
NUNOCAL	Calibração
NUNOCALT	Voltar à posição inicial depois de calibrar
NUNOC	Apanhar e arrumar objeto conhecido
NUNONC	Apanhar objeto não classificado
NUNOJC	Arrumar objeto
NUNOD	Apanhar e largar objeto desconhecido

3.6.3 Interface com utilizador

Depois de iniciado o programa, é necessário introduzir alguns comandos para que comecem a ser processados os dados e possam ser apanhados os objetos. Estas ordens são enviadas através do terminal (Figura 3.12), onde também vão surgindo mensagens para que o utilizador possa acompanhar o processo e atuar, se for necessário.

Existe também uma janela de visualização (Figura 3.13) onde se pode observar as nuvens de pontos e saber qual o objeto que será apanhado de seguida.

```
 Classe Y: -19
 Classe Z: -114
 Classe W: 179.721
 Classe P: 0.65261
 Classe R: 0

 Classe: Aproximação
 Classe X: 390
 Classe Y: -18
 Classe Z: -64
 Classe W: 179.721
 Classe P: 0.65261
 Classe R: 0
Novo Objeto Classificação: 3.2
Apanhar objeto Classificação: 3
Objeto Conhecido
Apanhar objeto conhecido
Posição final atingida
 Fanuc: Objeto apanhado, pedir nova coordenada
 Fanuc: Esperar 5 seg

 Kinect: Pedido de Coordenada
 Kinect: Coordenada (134ms)
 Kinect X: -230
 Kinect Y: 22
 Kinect Z: -667
 Kinect W: 2.55844
 Kinect P: 58.7608
 Kinect R: 0

 Kinect: Enviar valores:
 Kinect X: 354
 Kinect Y: 2
 Kinect Z: 216
 Kinect W: 2.55844
 Kinect P: 58.7354
 Kinect R: 0

 Fanuc: Posição impossivel de alcançar
```

Figura 3.12: Terminal com as mensagens trocadas entre nodos

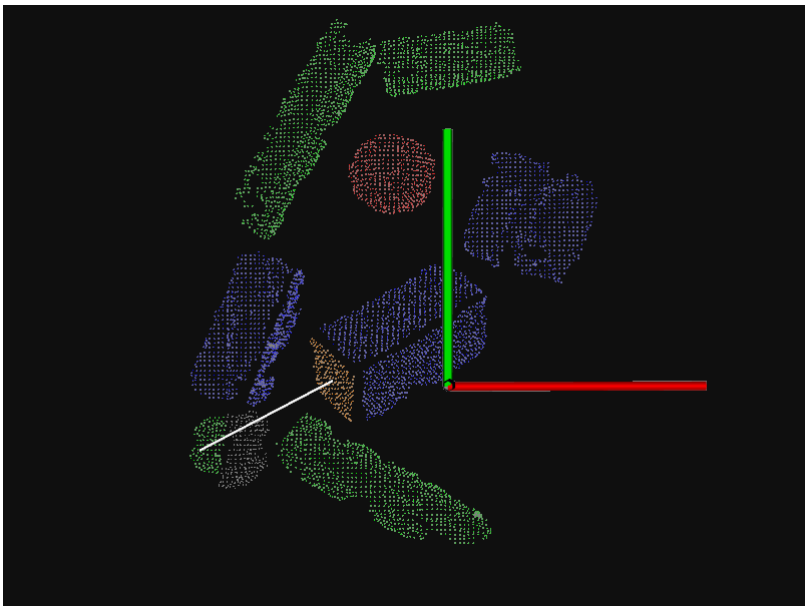


Figura 3.13: Janela de visualização onde surgem os objetos separados por cores

Capítulo 4

Identificação e Recolha dos Objetos

Como referido no Capítulo 1, o primeiro objetivo deste trabalho é a identificação dos objetos e obtenção da sua posição e orientação. Neste Capítulo está apresentada toda a estrutura de código utilizada para alcançar esse objetivo.

Uma vez que apenas é feito o tratamento dos dados 3D provenientes da câmara e os testes foram realizados no laboratório, não é necessário nenhum cuidado especial com a iluminação, uma vez que os dados RGB não são trabalhados e não existe interferência da luz solar ou qualquer outra fonte de luz IR.

A organização do código segue três fases importantes. Na primeira, realiza-se o tratamento da nuvem de pontos, onde são usados alguns filtros e transformações. Numa segunda fase são separadas as superfícies existentes na nuvem e classificadas como sendo esferas, cilindros ou planos. Entre estas é selecionada a candidata principal e identificado o objeto a que pertence. Por fim, a terceira fase consiste em obter as coordenadas do objeto e a orientação em que este se encontra.

Neste capítulo é ainda descrita de que forma é feita a recolha dos objetos.

4.1 Objetos utilizados

A escolha dos objetos para este trabalho teve em conta vários fatores. Em primeiro lugar pretendia-se que fossem objetos com geometrias muito próximas de objetos sujeitos a este tipo de aplicação. Em segundo lugar, as dimensões destes tinham de ser reduzidas, pois o espaço de trabalho era limitado devido às reduzidas dimensões do manipulador. Por último, como foi utilizada uma ventosa de vácuo, era necessário que a superfícies dos objetos não fosse rugosa ou porosa, nem que estes fossem demasiado pesados, para que esta os conseguisse apanhar sem dificuldade.

Dadas estas restrições escolheram-se três categorias de objetos com diferentes dimensões. Na Tabela 4.1 está apresentado cada um dos objetos selecionados.

Tabela 4.1: Propriedades dos objetos

Tipo	**Dimensão (mm)**	**Objeto**
Esfera 1	$\varnothing = 95$	
Cilindro 1	$\varnothing = 70\ L = 250$	
Cilindro 2	$\varnothing = 70\ L = 150$	
Caixa 1	$90 \times 115 \times 120$	
Caixa 2	$75 \times 170 \times 65$	

4.2 Tratamento de dados

4.2.1 Redução do número de pontos

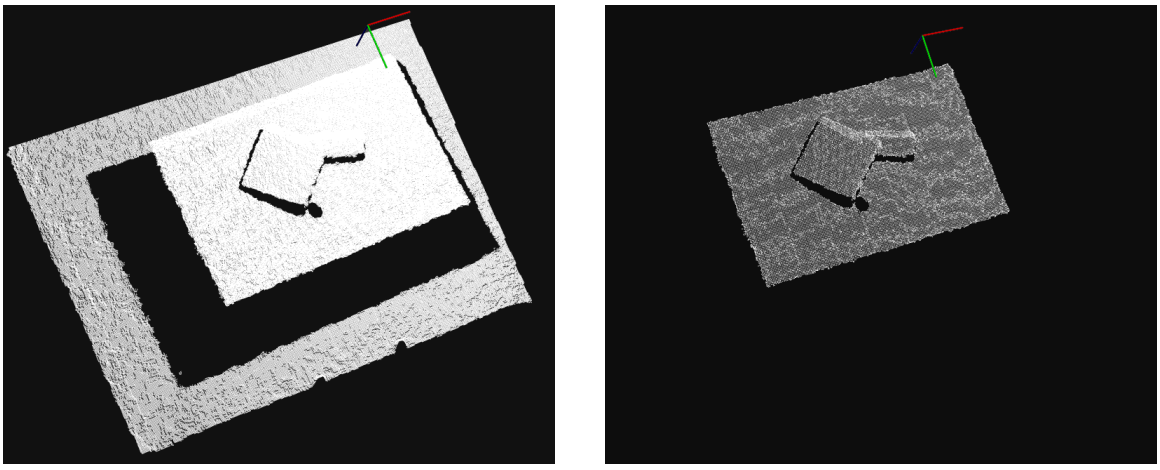
A nuvem de pontos proveniente da Kinect tem uma resolução de 640×480 píxeis com um *frame rate* de 30FPS. Como o tópico subscrito tem informação RGB, o número de bytes ocupado por cada píxel é 4. Isto traduz-se num total de 1200 kB por cada nuvem.

Utilizar uma nuvem de pontos tão densa faz com que o seu processamento seja lento e até mesmo impossível, sendo portanto necessário reduzir o número de pontos.

O primeiro filtro a ser aplicado à nuvem consiste em eliminar todos os pontos que se encontrem fora de um determinado limite. Para isso utilizaram-se as funções da classe *PassThrough*, em que os parâmetros de entrada são valores de distância em relação ao referencial da câmara. Como os objetos estão colocados em cima de uma mesa, todos os pontos que estejam fora do limite da mesa são irrelevantes e portanto devem ser eliminados. Neste caso se o tampo da mesa se encontrar a 1m da câmara o parâmetro correspondente ao valor máximo segundo o eixo Z deve ser 1, para que sejam ignorados todos os pontos para lá dessa distância. O mesmo se aplica aos eixos X e Y.

O segundo filtro para redução de pontos denomina-se *VoxelGrid* e consiste na criação de uma grelha tridimensional envolvente à nuvem, em que todos os pontos que se encontram no interior de cada bloco dessa grelha se reduzem a um único ponto, sendo este o centro dos demais. Um dos parâmetros de entrada deste filtro é a dimensão pretendida para a grelha. Foi utilizada uma grelha cúbica com 5 milímetros de lado.

O resultado desta primeira aplicação de filtros é visível na Figura 4.1.



(a) Nuvem de pontos inicial (b) Nuvem de pontos resultante

Figura 4.1: Aplicação dos filtros para reduzir o número de pontos

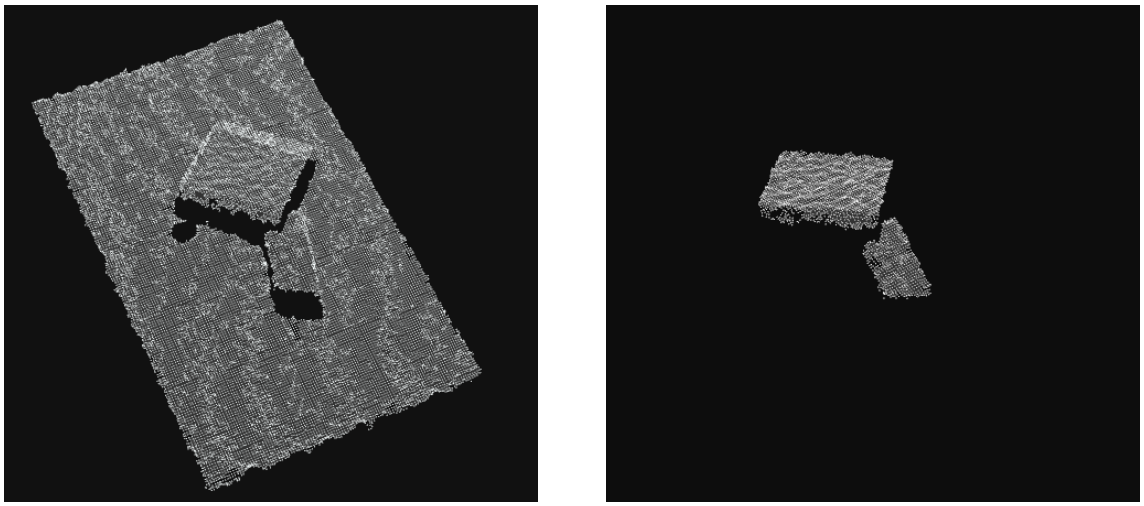
Na Figura 4.1a está representada a nuvem de pontos obtida pela Kinect em que é visível a elevada densidade de pontos. Nesta nuvem estão representados 2 objetos sobrepostos, a superfície da mesa e o chão. Com a aplicação dos filtros referidos, resulta como se vê na Figura 4.1b, uma nuvem menos densa e já sem os pontos que estão fora da área de trabalho.

Esta primeira operação permite que a nuvem possa ser trabalhada de forma rápida nos passos seguintes.

4.2.2 Divisão da nuvem de pontos

Uma forma de tornar o processo de identificação mais rápido e robusto é a divisão da nuvem de pontos em várias nuvens mais pequenas. Apesar do algoritmo ser repetido mais vezes, os dados a analisar são menos e, portanto, o tempo despendido é menor.

A primeira divisão aplicada à nuvem, baseia-se na implementação de um filtro que remove o plano de fundo formado pela mesa. Foi criado um objeto para a classe *SACSegmentation*, com o método iterativo conhecido por *RANSAC*[1], para estimar os pontos pertencentes a esse plano. Foram então obtidos os índices dos pontos que verificavam essa condição e com eles foi criada uma nova nuvem, sem os pontos correspondentes à mesa, através do método *ExtractIndices* (Figura 4.2).



(a) Nuvem de pontos inicial (b) Nuvem de pontos resultante

Figura 4.2: Remoção do plano formado pela mesa

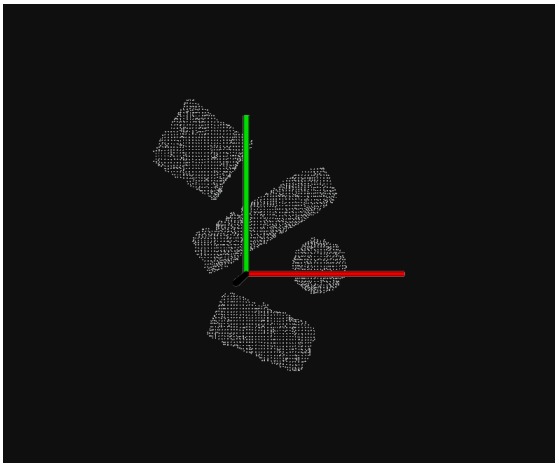
Após remover o plano correspondente à mesa, foi aplicado outro filtro denominado *RadiusOutlierRemoval* para remover alguns pontos isolados. Foram considerados isolados todos os pontos que num raio de 20 mm não tenham um mínimo de 30 pontos vizinhos.

Por último, a nuvem resultante é dividida em várias nuvens através das funções da classe *EuclideanClusterExtraction*. Para objetos que se encontrem separados, este método permite obter um objeto por nuvem. No entanto, se dois ou mais objetos estiverem encostados, por vezes são colocados na mesma nuvem. Nas figuras 4.3 e 4.4 estão representados estes dois casos.

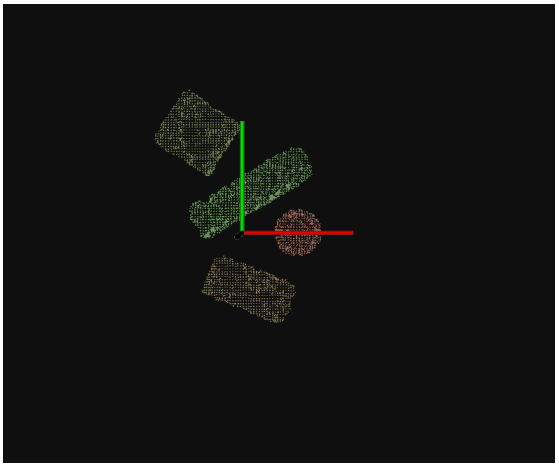
Como se pode ver na Figura 4.3a existem quatro aglomerados de pontos bem distintos, correspondendo cada a um objeto diferente. Na Figura 4.3b cada um desses aglomerados de pontos tem uma cor diferente, ou seja, são nuvens de pontos distintas. Neste caso o método de separação funciona com sucesso e portanto cada nuvem corresponde a um objeto diferente.

Na Figura 4.4a estão os mesmos objetos mas desta vez encostados uns aos outros. Visualmente é percetível que são quatro objetos diferentes e facilmente se identifica cada um deles, no entanto este método não consegue dividir a nuvem de pontos, pois o critério

[1] *RANSAC - RANdom SAmple Consensus*

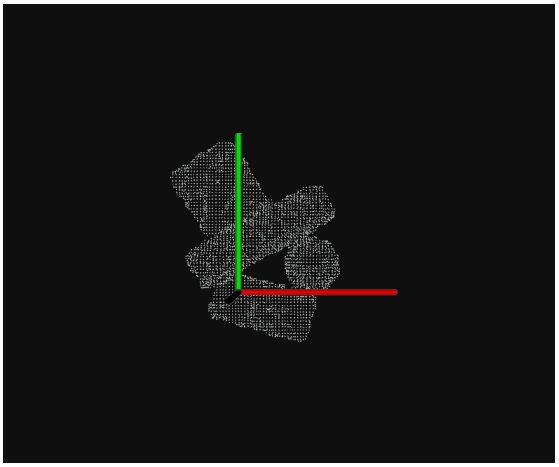


(a) Nuvem de pontos inicial

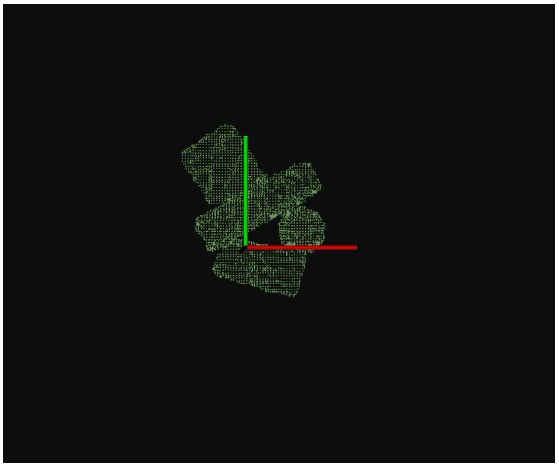


(b) Nuvens de pontos resultantes

Figura 4.3: Caso em que os objetos estão separados



(a) Nuvem de pontos inicial



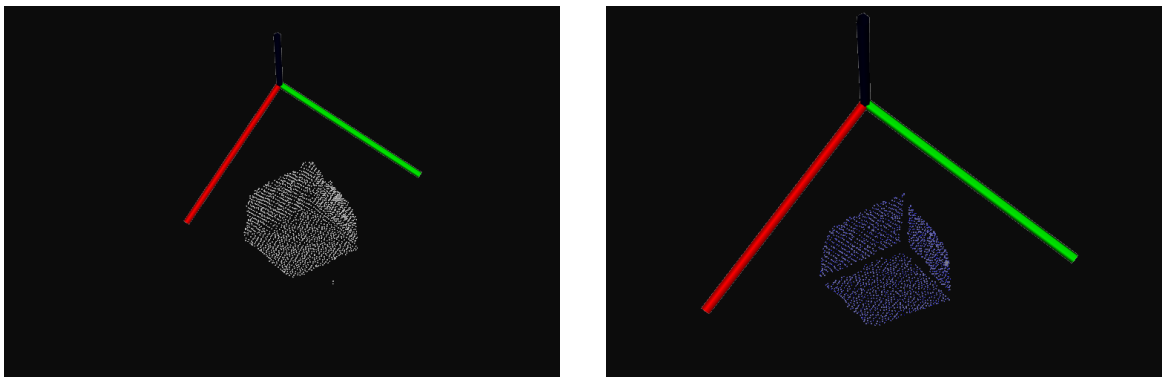
(b) Nuvem de pontos resultante

Figura 4.4: Caso em que os objetos estão juntos

de separação é a distância entre pontos. Como resultado todos os pontos representados na Figura 4.4b têm a mesma cor, ou seja, pertencem todos a uma só nuvem.

Uma vez que não é possível através da distância separar os objetos, utilizou-se a classe *RegionGrowing* que recorre à comparação entre os ângulos das normais de cada ponto para dividir a nuvem. Utilizando as funções desta classe consegue-se não só separar os objetos, quando se encontram juntos, como no mesmo objeto separar as suas faces (Figura 4.5).

A divisão de um objeto pelas suas faces permite que possam ser extraídas as características correspondentes à sua forma e dimensão. Com estas duas caraterísticas é possível identificar o tipo de objetos como se descreve de seguida.



(a) Antes (b) Depois

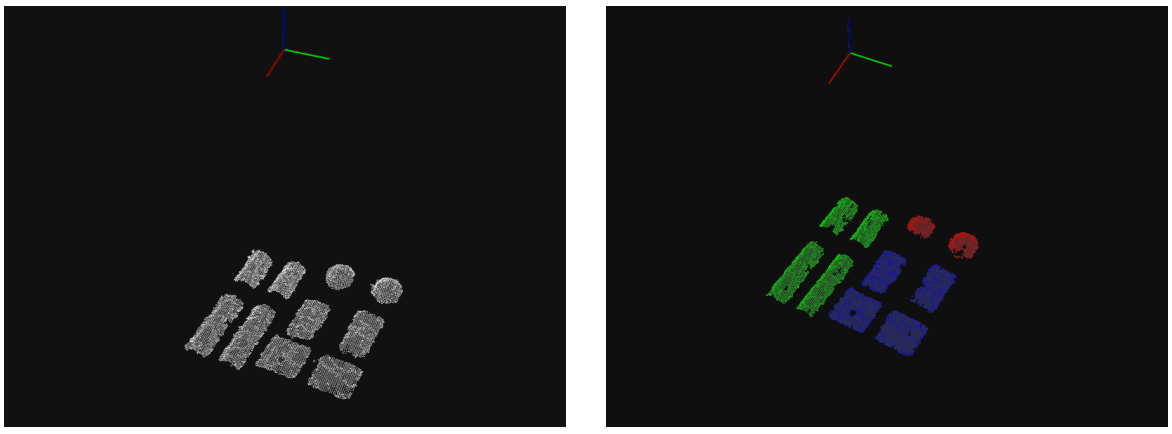
Figura 4.5: Separação das faces do mesmo objeto

4.3 Identificação

O processo de identificação dos objetos encontra-se dividido em duas partes. Na primeira, as nuvens obtidas anteriormente são submetidas a um algoritmo que procura superfícies geométricas conhecidas em cada uma. Na segunda fase é então feita a identificação do objeto candidato. Apesar de ser possível identificar vários objetos ao mesmo tempo, optou-se por apenas identificar aquele que é escolhido para apanhar. Esta decisão tem como único objetivo reduzir o tempo de processamento.

4.3.1 Procura e classificação de superfícies

Nesta primeira fase pretende-se classificar cada aglomerado de pontos consoante a sua geometria como se pode ver na Figura 4.6.



(a) Antes (b) Depois

Figura 4.6: Classificação das superfícies

Depois de separado, cada aglomerado de pontos da nuvem inicial (Figura 4.6a) é analisado e classificado (Figura 4.6b) como sendo uma esfera (a vermelho), um cilindro (a verde), um plano (a azul) ou desconhecido no caso de não pertencer a nenhum dos anteriores.

Para obter este resultado utilizou-se uma sequência de operações visível no pseudocódigo 4.1

Algorithm 4.1 Identificação das superfícies

```
for Nuvem=1:Número de Nuvens do
    if Nuvem corresponde com uma superfície esférica then
        Classificar a nuvem como esfera
        Obter centróide
        Obter vetor normal à superfície
    else
        Separar a nuvem por superfícies
        for Superfície=1:Número de Superfícies do
            if Superfície corresponde com uma superfície cilindrica then
                Classificar a nuvem como cilindro
                Obter centróide
                Obter vetor normal à superfície
            else
                if Superfície corresponde com uma superfície plana then
                    Classificar a nuvem como plano
                    Obter centróide
                    Obter vetor normal à superfície
                else
                    Classificar a nuvem como desconhecida
                    Obter centróide
                    Obter vetor normal à superfície
                end if
            end if
        end for
    end if
end for
```

Para saber se os pontos de cada nuvem formam uma superfície com as geometrias referidas utilizam-se as funções da classe *SACSegmentationFromNormals* com o método *RANSAC* e com os modelos *SPHERE*, *CYLINDER* e *PLANE*. A utilização desta classe implica o cálculo prévio dos vetores normais da nuvem, pois este é um dos parâmetros de entrada. Esses vetores foram obtidos através das funções da classe *NormalEstimation*.

A nuvem de pontos resultante contém os pontos da nuvem de entrada, que satisfazem a equação do modelo pretendido. No entanto, é necessário distinguir uma nuvem cuja maioria dos pontos satisfaz a equação, de uma nuvem em que apenas alguns pontos a verificam. Uma forma de o fazer é definir um valor de razão entre o número de pontos das nuvens, ou seja, se a nuvem de saída tiver mais do que uma determinada percentagem dos pontos da nuvem de entrada, é considerado que essa nuvem tem a geometria procurada. Os valores de rácio utilizados foram de 0.7 para as esferas e 0.8 para os cilindros e planos.

Classificada a superfície, é determinado o seu centróide com recurso à função *Compute3DCentroide* e selecionado o vetor normal correspondente ao ponto mais próximo.

Depois de aplicadas estas operações, resultam cinco vetores que contêm informação

de todas as nuvens analisadas. O primeiro vetor é constituído pelas nuvens de pontos consequentes da divisão da nuvem principal. O segundo contém as coordenadas dos centróides. No terceiro estão guardados os vetores normais à superfície formada pelos pontos. No quarto e quinto vetor encontram-se a distância dessas superfícies à câmara e a sua classificação, respetivamente.

4.3.2 Escolha e identificação do objeto

Com os dados contidos nos vetores anteriormente referidos, é possível escolher e identificar o objeto a apanhar.

O critério utilizado para determinar qual o objeto a ser apanhado é unicamente a distância à câmara. Ou seja, o objeto que se encontra mais próximo da câmara tem o valor de distância mais baixo e, portanto, é esse que deve ser recolhido.

Na figura 4.7 está representado com cor laranja os pontos que formam o plano mais próxima à câmara e a branco o respetivo vetor normal, com origem no seu centro.

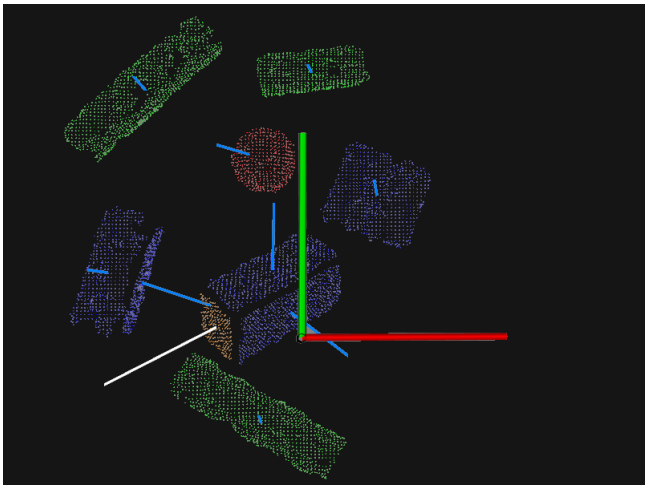


Figura 4.7: Superfície selecionada com base na proximidade

A escolha deste critério tem como base, evitar que os objetos vizinhos interfiram na aproximação da garra e recolha do objeto. Neste caso, como o objeto se encontra mais próximo da câmara não existe a possibilidade de outro poder estar por cima deste e impedir que seja recolhido em segurança.

Para identificar o objeto correspondente à nuvem selecionada são utilizadas as características dos objetos, nomeadamente a dimensão. Esta é calculada através da obtenção dos pontos mínimos é máximos da nuvem projetada pelas funções da classe PCA. Uma vez que as nuvens já estão classificadas como sendo esferas, cilindros ou planos, basta dentro de cada categoria encontrar o respetivo objeto.

4.4 Posição e Orientação

Para poder recolher o objeto, é necessário enviar as coordenadas em que este se encontra para o controlador do robô. No entanto, todas as coordenadas são relativas ao

referencial da câmara e portanto não podem ser enviadas diretamente sem antes serem corrigidas.

Para que as coordenadas enviadas estejam corretas em relação ao referencial do robô é necessário alinhar os referenciais e fazer uma calibração. Este é o primeiro procedimento a fazer sempre que se inicia o programa.

Além da posição é necessário conhecer a orientação em que o objeto se encontra. Assim, é possível calcular os ângulos a enviar para o controlador do robô para se fazer a aproximação ao objeto segundo o vetor normal à sua superfície.

4.4.1 Alinhamento dos referenciais

Os valores das coordenadas dos pontos das nuvens, são relativos ao referencial C (referencial da câmara). No entanto, este não se encontra alinhado com o referencial global G (referencial do robô) como mostra o Figura 4.8a.

Para orientar o referencial da câmara de forma que os seus eixos tenham direção e sentido iguais às do referencial global utilizou-se a matriz de transformação T (Equação 4.1).

$$T = Rot(z,\theta) \times Rot(y,\beta) \times Rot(x,\alpha) = \begin{bmatrix} c_\beta c_\theta & s_\alpha s_\beta c_\theta - c_\alpha s_\theta & c_\alpha s_\beta c_\theta + s_\alpha s_\theta & 0 \\ c_\beta s_\theta & s_\alpha s_\beta s_\theta + c_\alpha c_\theta & c_\alpha s_\beta s_\theta - s_\alpha c_\theta & 0 \\ -s_\beta & s_\alpha c_\beta & c_\alpha c_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

Para a posição em que foi colocada a câmara (Figura 4.8a) foram utilizados na equação os seguintes valores dos ângulos de Euler: $\alpha = \pi$, $\beta = 0$ e $\theta = \frac{\pi}{2}$.

O referencial da câmara passou a ter os seus eixos com a mesma direção e sentido dos eixos do referencial global. Na Figura 4.8b é visível o resultado da transformação geométrica efetuada.

(a) Antes da transformação (b) Depois da transformação

Figura 4.8: Referenciais da câmara e global

4.4.2 Calibração

Uma vez que os eixos dos referenciais já têm sentidos e direções iguais é necessário fazer a calibração para conhecer a translação que existe entre a origem de ambos os referenciais.

A calibração inicia-se com o robô a adotar uma posição pré estabelecida. É enviada uma mensagem para o nodo que trata os dados das nuvens de pontos, com as coordenadas da garra do robô (${}^{G}P$). No lado do nodo */Kinect*, quando é recebida a mensagem com as coordenadas e ordem de calibração, é analisada a nuvem de pontos, que corresponde à garra do robô, e obtidas as coordenadas no referencial da câmara (${}^{C}P$).

Finalmente, com as coordenadas referentes ao mesmo ponto, obtidas em referenciais diferentes, é possível conhecer a transformação (${}^{G}T_{C}$) que completa a Equação 4.2. Como a componente de rotação é igual à matriz identidade, uma vez que os eixos foram anteriormente alinhados, a matriz ${}^{G}T_{C}$ tem apenas componente de translação (Figura 4.9).

$$ {}^{G}P = {}^{G}T_{C}\ {}^{C}P \tag{4.2}$$

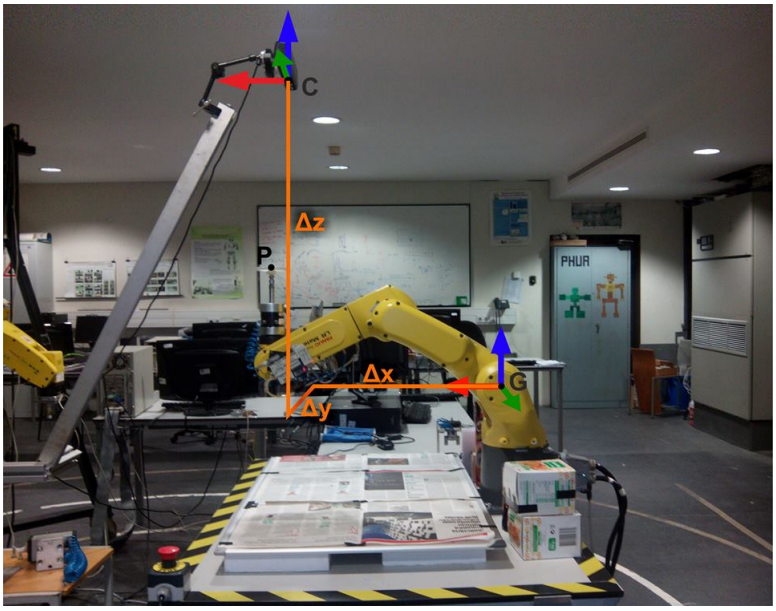


Figura 4.9: Translação entre referenciais

O valor que corresponde à translação existente entre os referenciais, ou seja, a posição da câmara no referencial global (${}^{G}C$), é guardado, pois é necessário sempre que é calculada a posição de um objeto.

4.4.3 Cálculo da posição dos objetos

Uma vez que já é conhecida a posição da câmara no referencial global (${}^{G}C$), através do método descrito, é possível determinar a posição dos objetos nesse mesmo referencial (${}^{G}O$) através da Equação 4.3.

$$^{G}O = {}^{G}C + {}^{C}O \tag{4.3}$$

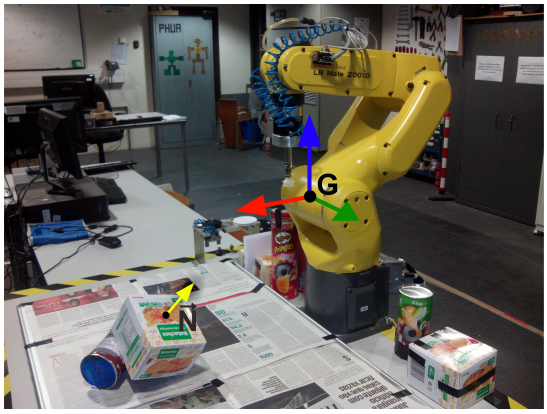
Em que ^{C}O é a posição central da superfície do objeto no referencial da câmara.

A posição do objeto no referencial global (^{G}O) é guardada numa variável e atualizada cada vez que é processada uma nova nuvem, para ser enviada quando solicitada pelo nodo */Fanuc*.

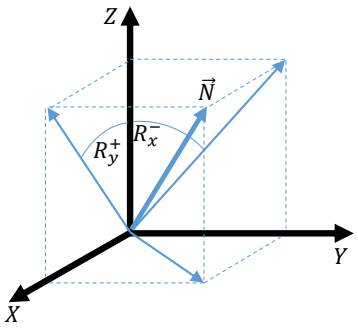
4.4.4 Cálculo da orientação dos objetos

Para calcular a orientação de um objeto no referencial global é necessário utilizar o vetor normal à superfície do objeto ($\vec{N}$), determinado anteriormente (Figura 4.10a). Tendo em conta que os eixos dos referenciais da câmara e global foram alinhados, os valores calculados com base neste vetor encontram-se todos no referencial global.

Através do vetor normal, é possível obter as componentes de rotação R_x e R_y, como é visível na Figura 4.10b. O cálculo destes ângulos é feito pelas equações 4.4 e 4.5.



(a) Vetor normal à superfície



(b) Representação no referencial global

Figura 4.10: Utilização do vetor normal

$$R_x = tan^{-1}(\frac{-N_y}{N_z}) \tag{4.4}$$

$$R_y = tan^{-1}(\frac{N_x}{N_z}) \tag{4.5}$$

No entanto, como estes valores resultam da projeção do vetor normal nos planos XZ e YZ, estes valores não podem ser usados diretamente no robô. É necessário conhecer quais os ângulos reais R_x e R_y que, usados nas matrizes de rotação $Rot(x,R_x)$ e $Rot(y,R_y)$, tornam o vetor unitário $\vec{v} = [0 \;\; 0 \;\; 1]'$ no vetor $\vec{N}$, como se pode ver pela Equação 4.6

$$\vec{N} = Rot(x,R_x) \times Rot(y,R_y) \times \vec{v} \tag{4.6}$$

Como a rotação sobre o eixo Y se dá após a rotação sobre X, pode considerar-se que R_x é o ângulo real de rotação sobre o eixo X.

Para calcular o valor real de R_y é necessário rodar o vetor $\vec{N}$ sobre o eixo X com o valor de $-R_x$ (Equação 4.7), desta forma, o vetor $\vec{N}$ fica coincidente com o plano XZ e o ângulo real R_y pode ser obtido pela Equação 4.8.

$$\vec{N^*} = Rot(x, -R_x) \times \vec{N} \tag{4.7}$$

$$R_y = tan^{-1}(\frac{N_x^*}{N_z^*}) \tag{4.8}$$

Obtidos os valores de R_x e R_y, através do vetor normal à superfície, falta conhecer a rotação em torno do eixo Z (R_z). Para obter este valor é necessário calcular o vetor principal $\vec{V}$ da superfície selecionada (Figura 4.11). Este é determinado com recuso à classe PCA[2].

(a) Representação no objeto (b) Representação na nuvem de pontos

Figura 4.11: Utilização do vetor principal

Obtido o vetor principal $\vec{V}$, o cálculo do ângulo R_z é efetuado através da equação 4.9.

$$R_z = tan^{-1}(\frac{V_y}{V_x}) \tag{4.9}$$

Calculadas as coordenadas do objeto e os ângulos necessários para obter a orientação do vetor normal correspondente, é publicada no tópico */reply* uma mensagem com esses dados. Todos os valores enviados são referentes ao referencial global, ou seja, do robô.

[2]PCA - *Principal Component Analysis*

4.5 Recolha dos Objetos

Como já foi referido no Capítulo 3, o nodo */Fanuc* é o responsável pela comunicação com o controlador do robô. Além da troca de mensagens, é nele que são calculadas as posições de aproximação e feita a escolha do programa que o robô deve utilizar, para apanhar os objetos ou fazer a calibração.

Nesta fase, todos os pontos calculados e enviados para o controlador do robô encontram-se no referencial global, correspondente ao sistema de coordenadas WORLD do robô.

Sempre que o nodo */Fanuc* recebe uma mensagem do nodo */Kinect*, com coordenadas de um novo ponto, é desencadeada uma sequência de ações. Em primeiro lugar é estabelecida a comunicação com o controlador do robô através do endereço de IP e porta de comunicação referidos no Capítulo 3. De seguida é iniciada a classe CommunicationHandler, onde estão contidas várias funções para enviar e receber informação do controlador. Por fim, dependendo do sucesso da comunicação com o robô, é feito o pedido de novas coordenadas, ou o término do ciclo.

Nesta secção é descrito como são calculados os pontos de aproximação, como é feita a escolha dos programas e qual a sequência de movimentos de cada um.

4.5.1 Posição de aproximação

Assim que se inicia a classe CommunicationHandler, são introduzidos como parâmetros de entrada os valores referentes às coordenadas da superfície do objeto a recolher. Estes valores correspondem aos dados contidos na mensagem enviada pelo nodo */Kinect*.

Para que o objeto seja recolhido corretamente é necessário enviar a garra para uma posição próxima ao objeto e só depois fazer o movimento de aproximação. Este procedimento permite que esse movimento se efetue segundo o vetor normal à superfície do objeto e portanto, sejam evitadas colisões.

Para calcular as coordenadas do ponto de aproximação (P_a) é necessário conhecer o ponto central da superfície do objeto (P_s) e a orientação do seu vetor normal ($\vec{N}$). Esta informação está contida na mensagem que o nó */Kinect* envia sempre que são pedidas novas coordenadas. Utilizou-se então a equação 4.10 para encontrar as coordenadas desse ponto.

$$P_a = P_s + Rot(x,\alpha) \times Rot(y,\beta) \times \begin{bmatrix} 0 \\ 0 \\ D \\ 1 \end{bmatrix} \tag{4.10}$$

O valor (P_s) corresponde ao valor ${}^{G}O$ descrito no Capítulo 5 e contido na mensagem recebida. Os valores de α e β correspondem aos ângulos R_x e R_y descritos anteriormente e que se encontram na mesma mensagem. A constante D diz respeito à distância pretendida entre o objeto e o ponto de aproximação, podendo o seu valor ser alterado.

Uma vez obtidas as coordenadas do ponto de aproximação é necessário calcular a orientação em que a garra se deve encontrar.

Como é visível na Figura 4.12 o eixo Z do referencial da garra tem de estar colinear com o vetor normal à superfície. Uma vez que os ângulos Rx e Ry do vetor ($\vec{N}$) já foram determinados anteriormente é possível encontrar os ângulos formados por um vetor

colinear, mas com sentido oposto. É segundo esta orientação que a garra se deve alinhar para fazer a aproximação aos objetos.

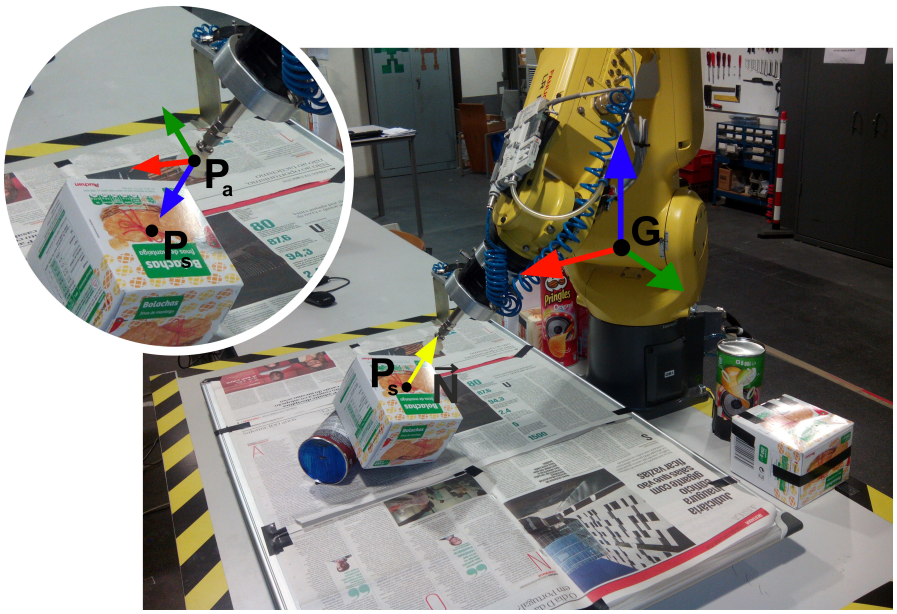


Figura 4.12: Posicionamento da garra para recolha do objeto

Foram utilizadas as equações 4.11 e 4.12 para determinar os ângulos de *roll* (R) e *pitch* (P) a enviar para o controlador.

$$R = 180 + R_x \tag{4.11}$$

$$P = -R_y \tag{4.12}$$

O valor de *yaw* (Y) corresponde ao valor R_z recebido.

4.5.2 Escolha do programa

Depois de serem enviadas coordenadas e configurações de junta para os registos do robô, através da instrução SETREG, é necessário escolher qual o programa a utilizar para definir os movimentos deste. A escolha desse programa depende da tarefa a executar, seja ela, movimento livre, calibração ou recolha de objeto.

Os programas utilizados já foram apresentados na Tabela 3.4, no entanto nesta secção será explicado em pormenor quando é que cada um é utilizado bem como o seu funcionamento.

Posição inicial

Quando é iniciada a comunicação com o robô são enviadas coordenadas de um ponto e configurações de junta que o robô deve adotar (Figura 4.13). Este é o estado inicial, no qual o robô se deve encontrar antes de iniciar qualquer movimento e onde deve retornar

após cada movimento. Para efetuar este movimento é utilizado o programa NUNOPOS, que realiza um movimento rápido de junta para uma posição pretendida.

Figura 4.13: Posição inicial

Movimentos para fazer calibração

Para os movimentos do processo de calibração existem dois programas. O primeiro a ser chamado é o NUNOCAL que faz o robô movimentar-se da posição inicial (Figura 4.14a) para uma posição fixa, onde apanha um objeto de calibração (Figura 4.14b). O objeto é movimentado (Figura 4.14c) para outra posição fixa, onde permanece durante alguns segundos para ser feita a calibração (Figura 4.14d). Após esta estar feita é então chamado o programa NUNOCALT que recoloca o calibre na mesma posição (Figura 4.16e) e movimenta o braço do robô para a sua configuração inicial (Figura 4.16f).

Movimentos para apanhar objetos

Quando se pretende apanhar um objeto, existem quatro programas que se podem utilizar dependendo das situações. Na Figura 4.15 está um esquema que demonstra as situações em que cada um é chamado.

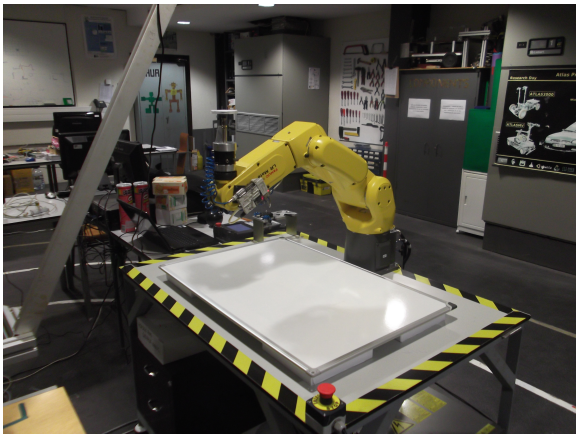
Se o objeto é conhecido e classificado, ou seja, a identificação corresponde a um objeto da tabela 4.1, é selecionado o programa NUNOC. Este faz o movimento da posição inicial (Figura 4.16a) para a posição de aproximação ao objeto (Figura 4.16b), apanha-o (Figura 4.16c), passa por uma posição de segurança (Figura 4.16d), arruma-o consoante a identidade (Figura 4.16e) e volta à posição inicial (Figura 4.16f).

No caso do objeto escolhido não ser conhecido, é necessário removê-lo da posição em que se encontra e assim permitir que este seja identificado posteriormente. Para poder fazer esta ação foi criado o programa NUNOD. Este faz a garra partir da posição inicial (Figura 4.17a), aproximar-se do objeto em segurança (Figura 4.17b), apanhá-lo

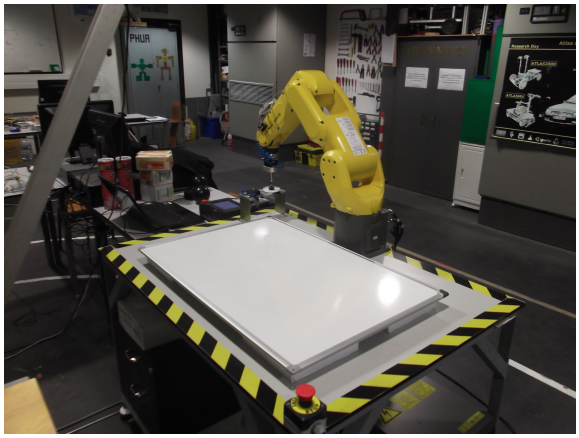
(a) Posição inicial

(b) Apanhar objeto de calibração

(c) Ir para posição de calibração



(d) Calibração



(e) Repor objeto de calibração

(f) Posição inicial

Figura 4.14: Processo de Calibração

(Figura 4.17c) e largá-lo de um ponto superior (Figura 4.17d), terminando o movimento na posição inicial.

Se o objeto for conhecido mas ainda não estiver classificado, como acontece quando a superfície detetada é o topo de um objeto cilíndrico, é necessário posicioná-lo em frente

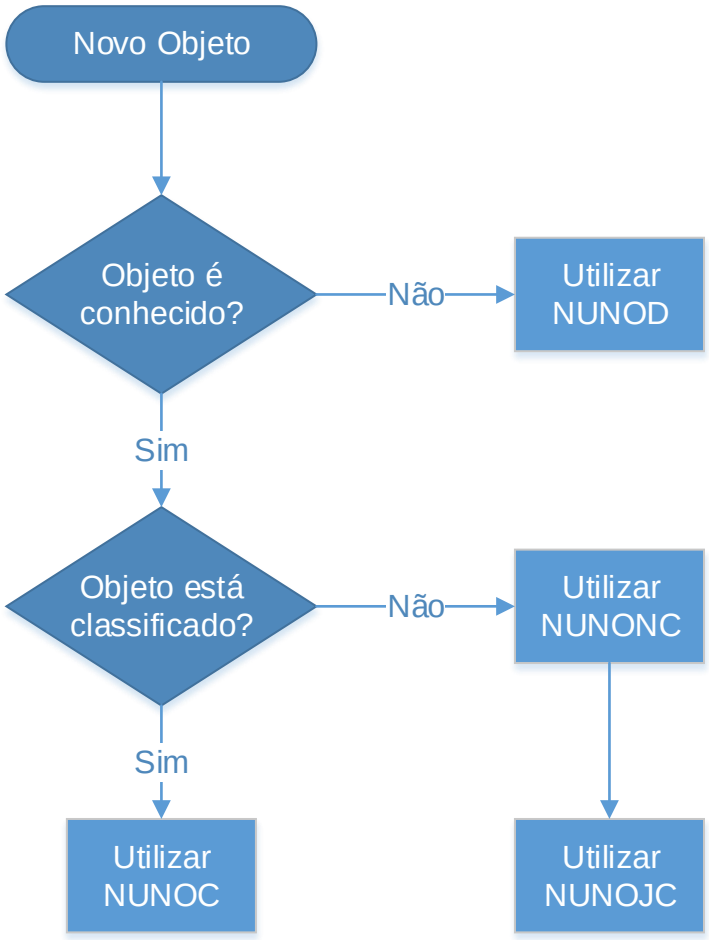
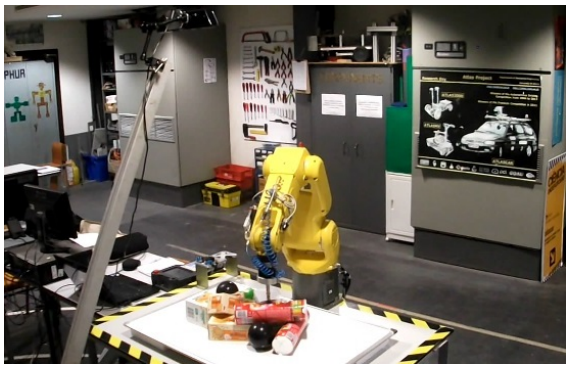


Figura 4.15: Fluxograma da escolha de programa para recolha dos objetos

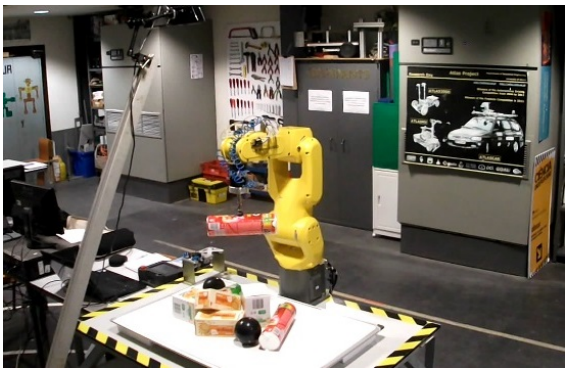
à câmara. Desta forma é possível visualizar o objeto de outra perspetiva e consecutivamente identificá-lo. Para realizar este conjunto de movimentos é executado o programa NUNONC, que à semelhança dos anteriores, parte de uma posição inicial (Figura 4.18a), movimenta-se até ao ponto de aproximação (Figura 4.18b), recolhe o objeto (Figura 4.18c) e posiciona-o à frente da câmara (Figura 4.18d). Após se conhecer qual a classificação, pretende-se que o objeto seja arrumado e para isso foi criado o programa NUNOJC. Este faz o braço movimentar-se até à posição de arrumação para libertar o objeto (Figura 4.18e) e de seguida volta à configuração inicial (Figura 4.18f).

(a) Posição inicial

(b) Posição de aproximação

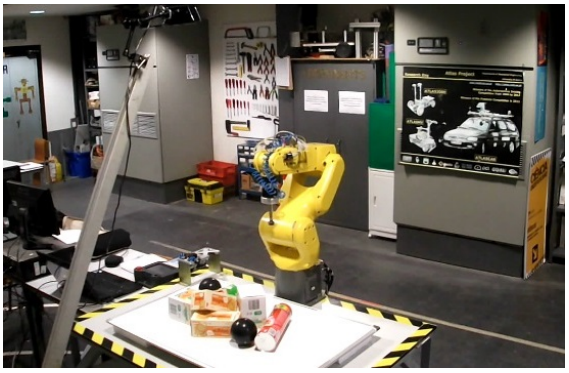


(c) Recolher objeto



(d) Posição de segurança

(e) Separar objeto



(f) Posição inicial

Figura 4.16: Recolher objeto conhecido

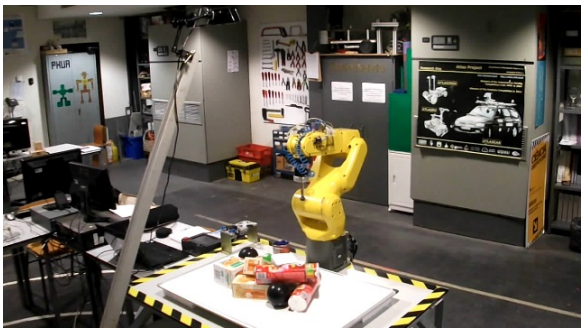
(a) Posição inicial

(b) Posição de aproximação

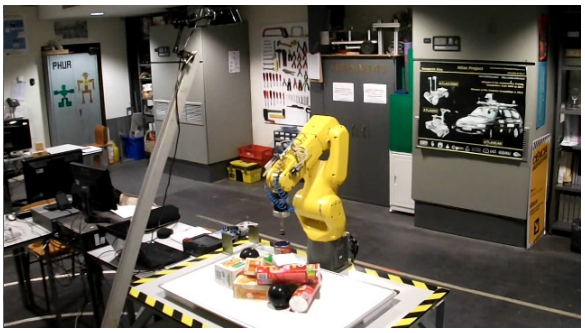
(c) Recolher objeto

(d) Libertar objeto

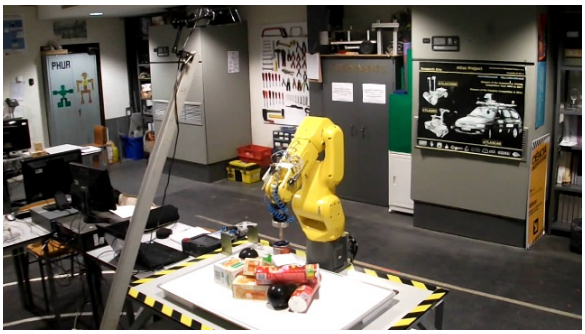
Figura 4.17: Recolher objeto desconhecido



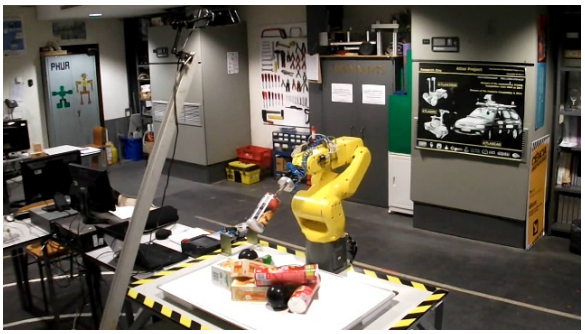
(a) Posição inicial



(b) Posição de aproximação



(c) Recolher objeto



(d) Classificação do objeto

(e) Separar objeto

(f) Posição inicial

Figura 4.18: Recolha do objeto não classificado

Capítulo 5

Testes e Análise de Resultados

Para ser possível avaliar o desempenho do programa desenvolvido é necessário realizar testes e analisar os resultados.

Neste trabalho, pode fazer-se essa avaliação através da percentagem de sucessos na identificação e recolha dos objetos. Como tal, optou-se por fazer duas séries de ensaios. Na primeira, apenas é colocado um objeto sobre a mesa de trabalho, em diversas posições, para testar a capacidade que o programa tem em identificar um objeto isolado. Na segunda série de ensaios, são colocados os 10 objetos sobre a mesa em diversas posições. Nesta, é avaliada a capacidade de identificação dos objetos quando empilhados, bem como, o sucesso da recolha.

5.1 Objeto individual

O primeiro teste realizado consistiu na identificação e recolha dos objetos, colocados individualmente e em várias posições sobre a mesa. Foram realizados 20 ensaios por cada tipo de objeto, perfazendo um total de 100.

O objetivo deste teste é avaliar a capacidade que o programa tem em identificar os objetos sem que exista interferência de outros, ou seja, classificar uma nuvem de pontos que corresponda a um só objeto.

Os resultados obtidos com este teste estão resumidos na Tabela 5.1

Tabela 5.1: Resultados dos testes com objeto individual

	Tempo (ms)	Classificação OK	Recolha OK	Total NOK	Total OK
Esfera 1	87	20	13	0	13
Cilindro 1	118	18	13	0	13
Cilindro 2	107	14	14	0	12
Caixa 1	113	16	17	2	15
Caixa 2	124	15	15	1	12

Os valores apresentados na primeira coluna da tabela correspondem ao tempo médio de processamento para identificação do objeto. Na segunda e terceira coluna estão indicadas as vezes que cada objeto foi corretamente identificado e bem recolhido pelo

manipulador, respetivamente. Na quinta e sexta colunas está indicado o número de "insucessos" (NOK) e "sucessos" (OK).

É considerado "sucesso" sempre que o objeto é bem identificado e recolhido, pois será colocado no local correto. Por outro lado, é considerado "insucesso" sempre que um objeto é mal identificado e o robô o consegue recolher, pois irá ser colocado no local errado.

Sempre que um objeto é classificado com classificação 5 (objeto não identificado) e é recolhido pelo manipulador, a classificação é considerada errada e a recolha correta, no entanto, como o objeto é libertado não é considerado "sucesso" ou "insucesso".

No gráfico da Figura 5.1 é possível visualizar a percentagem de classificações corretas e recolhas corretas.

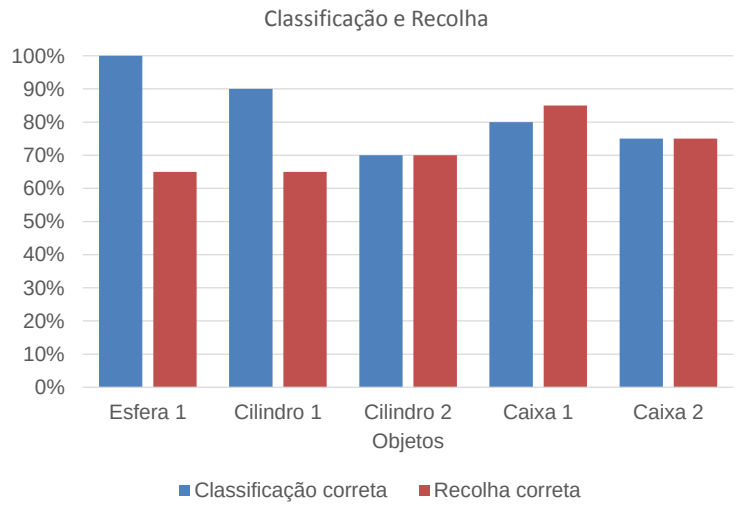


Figura 5.1: Percentagem de classificações e recolhas bem sucedidas do Teste 1

Analisando o gráfico, verifica-se que a identificação dos objetos, quando estão isolados, é feita corretamente em média mais de 80% das vezes. A recolha situa-se pouco acima dos 70%.

Durante os testes verificou-se que, apesar da garra se aproximar corretamente do objeto, quando havia contacto o objeto movia-se e a recolha não era bem sucedida. Uma das causas está relacionada com o peso dos objetos. A utilização de objetos demasiado leves levou a que, ao mínimo toque da garra, quando se encontravam em posições pouco estáveis, os objetos se movessem e impossibilitassem a recolha. A utilização de uma mesa com a superfície demasiado deslizante foi outro fator que dificultou a estabilidade dos objetos, pois o pouco atrito existente entre os objetos e a mesa foi favorável a que estes se movessem.

No entanto, como o principal objetivo deste teste incidia sobre a identificação dos objetos, e esta em nada é afetada pelas condicionantes referidas, optou-se por manter os dados e não repetir este teste.

No gráfico da Figura 5.2 é apresentada a percentagem de "sucessos" e "insucessos" anteriormente referidos.

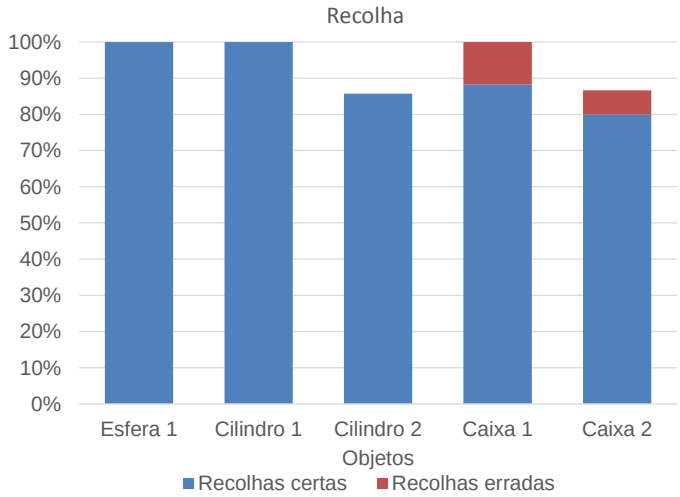


Figura 5.2: Percentagem de "sucessos" e "insucessos" do teste 1

Analisando o gráfico, verifica-se que em mais de 90% das tentativas os objetos foram recolhidos acertadamente e apenas 3.5% foram recolhas erradas. Nos casos em que a percentagem de recolhas corretas e erradas não perfaz um total de 100%, o défice da percentagem corresponde a identificações com classificação 5 (objeto não identificado), como referido anteriormente.

5.2 Objetos empilhados

Neste teste foram realizados 10 ensaios, onde cada um consistiu na recolha de todos os objetos dispostos na mesa, terminando quando não restassem mais objetos. O número total de objetos empilhados sobre a mesa, de forma aleatória, foi 10 (2 de cada tipo) em cada ensaio.

As tabelas com os dados recolhidos em cada ensaio, dos quais resultou a Tabela 5.2, encontram-se no Anexo B.

Tabela 5.2: Resultados dos testes com objetos empilhados

Ensaio	1	2	3	4	5	6	7	8	9	10
Tentativas	13	10	15	14	17	11	14	11	13	10
Tempo médio (ms)	195	195	217	184	189	215	231	200	202	201
Classificação OK	9	9	11	8	9	11	9	10	10	9
Recolha OK	11	10	11	13	13	10	11	10	11	10
Sucesso total	9	9	9	8	9	10	9	9	9	9

Cada coluna da tabela é referente a um ensaio para o qual estão indicadas: o número de tentativas necessárias, para recolher todos os objetos colocados sobre a mesa; o tempo médio que o programa demora a identificar o objeto; o número de vezes que a classificação é feita corretamente; o número de vezes que a recolha é bem executada; e a totalidade de "sucessos".

Um dos valores que é importante ter em conta quando se observa a tabela é o número de tentativas necessárias para recolher os 10 objetos da mesa. Os melhores ensaios tendo em conta este parâmetro foram o segundo e o décimo, pois o número de tentativas não ultrapassou o número de objetos. Por outro lado no ensaio 5 foram necessárias 17 tentativas, o que é um valor bastante elevado para uma aplicação deste tipo.

À semelhança do Teste 1, também neste teste ocorreram dificuldades na recolha dos objetos, pelas mesmas razões, e portanto, os resultados que se encontram na Tabela 5.2 resultam da repetição do teste em outras condições. Tendo em conta que a dificuldade na recolha se encontrava relacionada com o peso dos objetos e o pouco atrito existente entre eles e a mesa, alteraram-se essas condicionantes. Aumentando o peso dos objetos e criando uma superfície com mais atrito a média de tentativas necessárias para recolher todos os objetos caiu de 16 para 13, uma vez que as recolhas mal sucedidas devido a estes motivos passaram a ser quase inexistentes. Esta alteração em nada afetou a identificação dos objetos.

No gráfico da Figura 5.3 são apresentadas as percentagens relativas às classificações e recolhas corretas em cada ensaio.

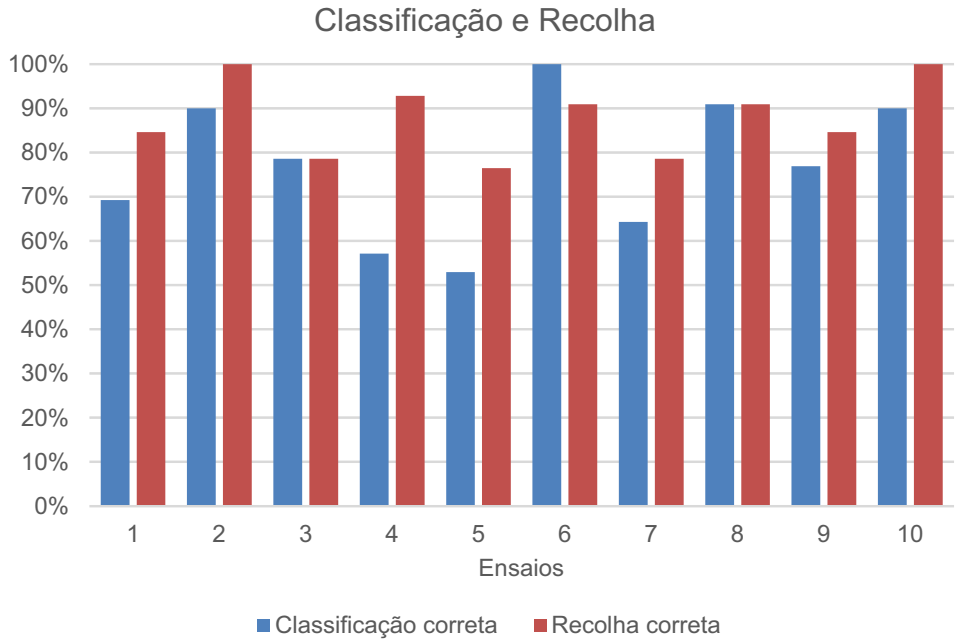


Figura 5.3: Percentagem de classificações e recolhas bem sucedidas do teste 2

Como se observa no gráfico, a percentagem de identificações corretas foi de 77% com um desvio padrão de 17%, valor abaixo dos 80% obtidos na identificação individual e com elevada variação. Esta diferença demonstra que a identificação dos objetos quando estes estão empilhados é realizada com menos eficiência.

Quanto à recolha, foi obtida uma média de objetos bem recolhidos de 87% com desvio padrão de 5%, bastante superior aos 70% obtidos no teste individual. Como já foi referido, as alterações efetuadas nos objetos e na superfície da mesa foram fundamentais para esta melhoria considerável.

Na Figura 5.4 é apresentado o gráfico com a percentagem de "sucessos" e "insucessos" deste teste.

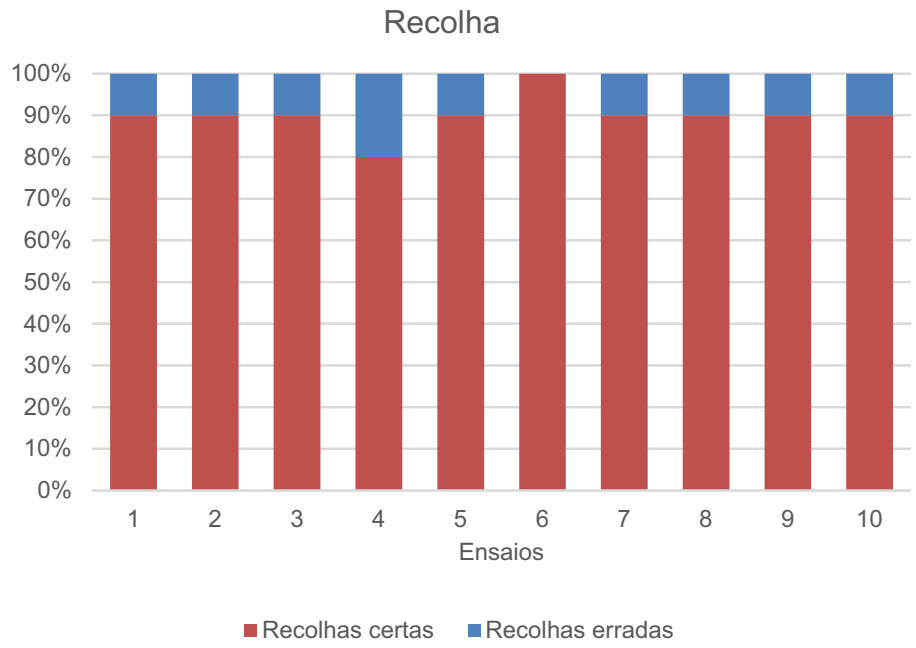


Figura 5.4: Percentagem de "sucessos" e "insucessos" do Teste 2

Deste gráfico retém-se que 90% das recolhas são feitas corretamente e o objeto é arrumado no local correto. Os restantes 10% correspondem a recolhas incorretas. Neste caso, os objetos com classificação 5 não são contabilizados pois mesmo que o robô os recolha eles são libertados de seguida e nunca há risco de serem arrumados no local errado.

Capítulo 6

Conclusões

Com este trabalho pretendia-se desenvolver um sistema de *bin picking* utilizando apenas um dispositivo de perceção 3D. Este devia ser capaz de identificar objetos posicionados aleatoriamente e conseguir recolhê-los corretamente.

Para cumprir com os objetivos propostos, foram utilizados um robô Fanuc de 6 juntas, uma câmara Kinect e um computador. Trabalhou-se em ambiente Linux, utilizou-se a infraestrutura ROS e a biblioteca de código aberto PCL.

O trabalho foi dividido em duas partes, correspondendo a primeira ao tratamento dos dados, para identificação dos objetos e obtenção das suas coordenadas, e a segunda ao controlo do manipulador, para apanhar os objetos selecionados.

Na fase de identificação dos objetos foram testadas várias soluções, como as descritas em [19] e [20], em que existe uma base de dados com as nuvens de pontos dos objetos conhecidos e é procurada, nessa base de dados, a nuvem que mais se aproxima da que está a ser testada. No entanto, estas soluções mostraram-se inviáveis dado o extenso tempo necessário para processamento e a baixa correspondência com os objetos utilizados. Como alternativa, optou-se por identificar os objetos através da geometria e dimensão das superfícies. Com este método é feita a identificação e obtida a posição e orientação do objeto com um tempo médio de processamento na ordem dos 200ms. Além da redução no tempo de processamento, os resultados obtidos com este método mostraram-se mais fiáveis e portanto optou-se por esta solução.

Na fase de recolha dos objetos foram criados vários programas no controlador do robô para este executar diversas sequências de movimentos dependendo da tarefa a realizar. No programa de computador são realizados os cálculos para determinar os pontos de aproximação ao objeto e a orientação que a garra deve adotar para fazer uma recolha segura e eficiente.

Os resultados obtidos durante os testes mostram que o trabalho realizado cumpre com os objetivos propostos. A identificação dos objetos é conseguida unicamente através do processamento de dados 3D em tempo real e apresenta uma percentagem de sucesso positiva (77%). A recolha dos objetos também é conseguida de forma correta. Por último, o processamento de dados é efetuado de forma rápida como era pretendido.

6.1 Trabalho futuro

Sendo este um trabalho iniciado de raiz, ou seja, não se engloba no seguimento de outros trabalhos ou projetos já realizados no laboratório, encontra-se numa fase embrionária e, portanto, é possível a sua melhoria em trabalhos futuros.

De seguida apresentam-se algumas ideias que poderiam ser implementadas para completar e melhorar o sistema desenvolvido, bem como, indicações de possíveis caminhos a seguir.

6.1.1 Objetos colocados dentro de uma caixa

Para que o "verdadeiro" *bin picking* pudesse ser realizado, os objetos deveriam estar colocados dentro de uma caixa e não apenas dispostos sobre uma mesa. A utilização da mesa permitiu que o robô tivesse mais flexibilidade nos movimentos, no entanto, seria interessante adicionar esta variável ao problema e verificar de que forma os resultados seriam influenciados.

Neste caso, seria necessário fazer algumas alterações ao programa para que fosse capaz de ignorar as paredes da caixa e não as confundir com objetos. Uma das possíveis formas de o fazer pode passar por tornar os parâmetros de entrada da classe *PassThrough* dinâmicos de modo a corresponderem às dimensões da caixa. Desta forma a nuvem de pontos a analisar apenas iria conter as superfícies dos objetos, como acontece no programa atual.

6.1.2 Aprendizagem de novos objeto

No programa desenvolvido as dimensões dos objetos conhecidos foram colocadas diretamente no código do programa, ou seja, para adicionar um novo objeto é necessário alterar o código e colocar os valores manualmente. Este não é um procedimento correto para uma aplicação deste género, sendo necessário a sua alteração para um método mais flexível.

Mantendo o processo de identificação dos objetos através da geometria das superfícies e suas dimensões, como está implementado, é possível recorrer à aprendizagem de um objeto guardando os valores das suas dimensões em ficheiro. Deste modo, sempre que um objeto fosse visualizado pelo sensor, para ser adicionado à base de dados, uma nova linha seria adicionada ao ficheiro com todas as dimensões e caraterísticas do objeto.

6.1.3 Deteção de peça recolhida

Uma das situações que se verificou algumas vezes durante os testes, foi o facto do manipulador não conseguir apanhar os objetos. No entanto, todo o procedimento para os arrumar era desencadeado, como se a recolha tivesse sido efetuada com sucesso.

Nesta situação, a solução pode passar por implementar na garra um sensor de força ou proximidade que indique o sucesso ou insucesso da recolha. Por outro lado, a utilização de tal sensor pode impedir que a garra continue o movimento de aproximação ao objeto assim que toca no mesmo, evitando a danificação do objeto ou da própria garra.

6.1.4 Método de reconhecimento dos objetos

O método de reconhecimento implementado neste trabalho, baseou-se nas formas e dimensões das superfícies dos objetos. Para objetos pouco complexos, como os utilizados, verificou-se a funcionalidade deste método, contudo, objetos com geometrias um pouco mais complexas tornam este método insuficiente.

Outras formas de reconhecimento foram experimentadas, como as indicadas no Capítulo 2, mas o tempo de processamento necessário era inviável para uma aplicação deste género. No entanto, a utilização do método implementado em conjunto com uma dessas alternativas poderia apresentar resultados interessantes.

Bibliografia

[1] department IS. Global robotics industry: Record beats record; 2014. http://www.worldrobotics.org/index.php?id=home&news_id=273.

[2] Biomechanics D. Science Behind 3D Vision; 2014. http://www.depthbiomechanics.co.uk/?p=102.

[3] mtiinstruments. Laser Triangulation Sensors; 2014. http://www.mtiinstruments.com/products/lasertriangulation.aspx.

[4] Hansard M, Lee S, Choi O, Horaud R. Time-of-Flight Cameras: Principles,Methods and Applications. Springer London;.

[5] McIlroy P, Izadi S, Fitzgibbon A. 6-DoF Pose Estimation Using a Projected Dense Dot Pattern. 2014;.

[6] Shop MM. Robotic Bin Picking Automation Effectively Channels Random Part; 2014. http://www.mmsonline.com/products/robotic-bin-picking-automation-effectively-channels-random-parts.

[7] Robotics U. Random Bin Picking & Automated Assembly; 2014. http://www.universalrobotics.com/random-bin-picking.

[8] Chubb P. Kinect Teardown: Pleo, The Dinosaur Robot Similarities; 2014. http://www.product-reviews.net/2010/11/05/kinect-teardown-pleo-the-dinosaur-robot-similarities/.

[9] Robotics F. LR Mate 200iC Series & R-30iA Mate Controller; 2009. http://www.fanucrobotics.com/cmsmedia/datasheets/LR%20Mate%20200iC%20Series_10.pdf.

[10] Stepien M, Biesenbach R. Integration of a 2D Vision System into a Control of an Industrial Robot. KMUTNB: International Journal of Applied Science and Technology. 2014;.

[11] Buchholz D, Futterlieb M, Winkelbach S, Wahl FM. Efficient bin-picking and grasp planning based on depth data. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on. IEEE; 2013. p. 3245–3250.

[12] Instruments N. Sistemas de imagens 3D com NI LabVIEW; 2014. http://www.ni.com/white-paper/14103/pt/#toc1.

[13] Khan CMZA. Development and Evaluation of a Kinect based Bin-Picking System. Malardalen University;.

[14] Kent D. Construction of a 3D Object Recognition and Manipulation Database from Grasp Demonstrations. Worcester Polytechnic Institute;.

[15] ROS. About ROS; 2014. http://www.ros.org/about-ros/.

[16] Rusu RB, Cousins S. 3D is here: Point Cloud Library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA); 2011. .

[17] ROS. openni_launch/Tutorials/IntrinsicCalibration; 2014. http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration.

[18] Cancela R. Extensão e flexibilização da interface de controlo de um manipulador robótico FANUC. Universidade de Aveiro;.

[19] PCL. 3D Object Recognition based on Correspondence Grouping; 2014. http://pointclouds.org/documentation/tutorials/correspondence_grouping.php#correspondence-grouping.

[20] PCL. How to use iterative closest point; 2014. http://pointclouds.org/documentation/tutorials/iterative_closest_point.php#iterative-closest-point.

Anexos

Anexo A

Programas TP

A.1 NUNOPOS

```
/PROG NUNOPOS
/ATTR
OWNER = MNEDITOR;
COMMENT = ;
PROG_SIZE = 160;
CREATE = DATE 14-09-30 TIME 22:54:08;
MODIFIED = DATE 14-10-12 TIME 18:05:56;
FILE_NAME = ;
VERSION = 0;
LINE_COUNT = 2;
MEMORY_SIZE = 524;
PROTECT = READ_WRITE;
TCD: STACK_SIZE = 0,
TASK_PRIORITY = 50,
TIME_SLICE = 0,
BUSY_LAMP_OFF = 0,
ABORT_REQUEST = 0,
PAUSE_REQUEST = 0;
DEFAULT_GROUP = 1,*,*,*,*;
CONTROL_CODE = 00000000 00000000;
/APPL
/MN
1:J PR[70:robCOMM] 200msec FINE ;
2: Open hand 1 ;
/POS
/END
```

A.2 NUNOCAL

```
/PROG NUNOCAL
```

```
/ATTR
OWNER = MNEDITOR;
COMMENT = ;
PROG_SIZE = 242;
CREATE = DATE 14-09-30 TIME 23:14:24;
MODIFIED = DATE 14-10-12 TIME 18:07:34;
FILE_NAME = ;
VERSION = 0;
LINE_COUNT = 7;
MEMORY_SIZE = 586;
PROTECT = READ_WRITE;
TCD: STACK_SIZE = 0,
TASK_PRIORITY = 50,
TIME_SLICE = 0,
BUSY_LAMP_OFF = 0,
ABORT_REQUEST = 0,
PAUSE_REQUEST = 0;
DEFAULT_GROUP = 1,*,*,*,*;
CONTROL_CODE = 00000000 00000000;
/APPL
/MN
1:J PR[70:robCOMM] 100% FINE ;
2: Open hand 1 ;
3:J PR[71:robCOMM] 100% FINE ;
4:J PR[72:robCOMM] 100% FINE ;
5: Close hand 1 ;
6:J PR[71:robCOMM] 100% FINE ;
7:J PR[73:robCOMM] 100% FINE ;
/POS
/END
```

A.3 NUNOCALT

```
/PROG NUNOCALT
/ATTR
OWNER = MNEDITOR;
COMMENT = ;
PROG_SIZE = 214;
CREATE = DATE 14-09-30 TIME 23:37:46;
MODIFIED = DATE 14-10-12 TIME 18:06:56;
FILE_NAME = ;
VERSION = 0;
LINE_COUNT = 5;
MEMORY_SIZE = 566;
PROTECT = READ_WRITE;
```

```
TCD: STACK_SIZE = 0,
TASK_PRIORITY = 50,
TIME_SLICE = 0,
BUSY_LAMP_OFF = 0,
ABORT_REQUEST = 0,
PAUSE_REQUEST = 0;
DEFAULT_GROUP = 1,*,*,*,*;
CONTROL_CODE = 00000000 00000000;
/APPL
/MN
1:J PR[71:robCOMM] 100% FINE ;
2:J PR[72:robCOMM] 100% FINE ;
3: Open hand 1 ;
4:J PR[71:robCOMM] 100% FINE ;
5:J PR[70:robCOMM] 100% FINE ;
/POS
/END
```

A.4 NUNOC

```
/PROG NUNOC
/ATTR
OWNER = MNEDITOR;
COMMENT = ;
PROG_SIZE = 314;
CREATE = DATE 14-10-01 TIME 00:02:48;
MODIFIED = DATE 14-10-22 TIME 22:14:00;
FILE_NAME = ;
VERSION = 0;
LINE_COUNT = 11;
MEMORY_SIZE = 642;
PROTECT = READ_WRITE;
TCD: STACK_SIZE = 0,
TASK_PRIORITY = 50,
TIME_SLICE = 0,
BUSY_LAMP_OFF = 0,
ABORT_REQUEST = 0,
PAUSE_REQUEST = 0;
DEFAULT_GROUP = 1,*,*,*,*;
CONTROL_CODE = 00000000 00000000;
/APPL
/MN
1:J PR[70:robCOMM] 100% FINE ;
2:J PR[71:robCOMM] 100% FINE ;
3: Close hand 1 ;
```

```
4:L PR[72:robCOMM] 4000mm/sec FINE ;
5:L PR[71:robCOMM] 4000mm/sec FINE ; 6:J PR[73:robCOMM] 100% FINE ;
7:J PR[74:robCOMM] 100% FINE ;
8:L PR[75:robCOMM] 4000mm/sec FINE ;
9: Open hand 1 ;
10:L PR[74:robCOMM] 4000mm/sec FINE ;
11:J PR[70:robCOMM] 100% FINE ;
/POS
/END
```

A.5 NUNONC

```
/PROG NUNONC
/ATTR
OWNER = MNEDITOR;
COMMENT = ;
PROG_SIZE = 232;
CREATE = DATE 14-09-30 TIME 23:49:54;
MODIFIED = DATE 14-10-22 TIME 22:15:18;
FILE_NAME = ;
VERSION = 0;
LINE_COUNT = 6;
MEMORY_SIZE = 580;
PROTECT = READ_WRITE;
TCD: STACK_SIZE = 0,
TASK_PRIORITY = 50,
TIME_SLICE = 0,
BUSY_LAMP_OFF = 0,
ABORT_REQUEST = 0,
PAUSE_REQUEST = 0;
DEFAULT_GROUP = 1,*,*,*,*;
CONTROL_CODE = 00000000 00000000;
/APPL
/MN
1:J PR[70:robCOMM] 100% FINE ;
2:J PR[71:robCOMM] 100% FINE ;
3: Close hand 1 ;
4:L PR[72:robCOMM] 4000mm/sec FINE ;
5:L PR[71:robCOMM] 4000mm/sec FINE ;
6:J PR[73:robCOMM] 100% FINE ;
/POS
/END
```

A.6 NUNOJC

```
/PROG NUNOJC
/ATTR
OWNER = MNEDITOR;
COMMENT = ;
PROG_SIZE = 214;
CREATE = DATE 14-10-01 TIME 00:10:48;
MODIFIED = DATE 14-10-12 TIME 18:07:52;
FILE_NAME = ;
VERSION = 0;
LINE_COUNT = 5;
MEMORY_SIZE = 566;
PROTECT = READ_WRITE;
TCD: STACK_SIZE = 0,
TASK_PRIORITY = 50,
TIME_SLICE = 0,
BUSY_LAMP_OFF = 0,
ABORT_REQUEST = 0,
PAUSE_REQUEST = 0;
DEFAULT_GROUP = 1,*,*,*,*;
CONTROL_CODE = 00000000 00000000;
/APPL
/MN
1:J PR[73:robCOMM] 100% FINE ;
2:J PR[74:robCOMM] 100% FINE ;
3: Open hand 1 ;
4:L PR[74:robCOMM] 4000mm/sec FINE ;
5:J PR[70:robCOMM] 100% FINE ;
/POS
/END
```

A.7 NUNOD

```
/PROG NUNOD
/ATTR
OWNER = MNEDITOR;
COMMENT = ;
PROG_SIZE = 260;
CREATE = DATE 14-09-30 TIME 23:56:00;
MODIFIED = DATE 14-10-22 TIME 22:14:38;
FILE_NAME = ;
VERSION = 0;
LINE_COUNT = 8;
MEMORY_SIZE = 600;
PROTECT = READ_WRITE;
```

```
TCD: STACK_SIZE = 0,
TASK_PRIORITY = 50,
TIME_SLICE = 0,
BUSY_LAMP_OFF = 0,
ABORT_REQUEST = 0,
PAUSE_REQUEST = 0;
DEFAULT_GROUP = 1,*,*,*,*;
CONTROL_CODE = 00000000 00000000;
/APPL
/MN
1:J PR[70:robCOMM] 100% FINE ;
2:J PR[71:robCOMM] 100% FINE ;
3: Close hand 1 ;
4:L PR[72:robCOMM] 4000mm/sec FINE ;
5:L PR[71:robCOMM] 4000mm/sec FINE ;
6:J PR[73:robCOMM] 100% FINE ;
7: Open hand 1 ;
8:J PR[70:robCOMM] 100% FINE ;
/POS
/END
```

Anexo B

Tabelas dos ensaios

Tabela B.1: Ensaio 1

Ensaio 1								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	298	3.1	3.1		1	1	FALSE	TRUE
2	274	2.2	4	2.2	1	1	FALSE	TRUE
3	291	2.1	2.1		1	1	FALSE	TRUE
4	257	3.2	5		0	1	FALSE	FALSE
5	213	3.1	4		0	0	FALSE	FALSE
6	216	3.2	3.2		1	1	FALSE	TRUE
7	176	1	1		1	1	FALSE	TRUE
8	181	1	1		1	1	FALSE	TRUE
9	177	3.2	3.2		1	1	FALSE	TRUE
10	136	3.1	3.1		1	1	FALSE	TRUE
11	97	2.2	5		0	0	FALSE	FALSE
12	115	2.1	3.1		0	1	TRUE	FALSE
13	103	2.2	2.2		1	1	FALSE	TRUE

Tabela B.2: Ensaio 2

Ensaio 2								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	246	1	2.2		0	1	TRUE	FALSE
2	244	3.1	4	3.1	1	1	FALSE	TRUE
3	235	2.2	4	2.2	1	1	FALSE	TRUE
4	235	2.1	2.1		1	1	FALSE	TRUE
5	218	3.2	3.2		1	1	FALSE	TRUE
6	213	2.1	2.1		1	1	FALSE	TRUE
7	174	3.2	3.2		1	1	FALSE	TRUE
8	140	1	1		1	1	FALSE	TRUE
9	139	3.1	3.1		1	1	FALSE	TRUE
10	102	2.2	2.2		1	1	FALSE	TRUE

Tabela B.3: Ensaio 3

Ensaio 3								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	283	1	1		1	1	FALSE	TRUE
2	303	3.1	3.1		1	0	FALSE	FALSE
3	305	3.1	3.1		1	0	FALSE	FALSE
4	311	3.2	2.1		0	1	TRUE	FALSE
5	276	2.2	2.2		1	1	FALSE	TRUE
6	259	3.2	3.2		1	1	FALSE	TRUE
7	242	1	1		1	1	FALSE	TRUE
8	209	3.1	3.1		1	1	FALSE	TRUE
9	190	2.2	5		0	0	FALSE	FALSE
10	197	3.1	3.1		1	1	FALSE	TRUE
11	169	2.1	5		0	1	FALSE	FALSE
12	124	2.2	2.2		1	1	FALSE	TRUE
13	169	2.1	5		0	0	FALSE	FALSE
14	168	2.1	2.1		1	1	FALSE	TRUE
15	121	2.1	2.1		1	1	FALSE	TRUE

Tabela B.4: Ensaio 4

Ensaio 4								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	271	2.1	4	2.1	1	1	FALSE	TRUE
2	255	3.1	4	3.1	1	1	FALSE	TRUE
3	239	3.2	5		0	1	FALSE	FALSE
4	243	3.2	5		0	1	FALSE	FALSE
5	246	3.2	3.1		0	1	TRUE	FALSE
6	205	2.1	2.1		1	1	FALSE	TRUE
7	182	3.2	5		0	1	FALSE	FALSE
8	166	3.2	4	3.1	0	1	TRUE	FALSE
9	169	1	1		1	1	FALSE	TRUE
10	112	2.2	5		0	0	FALSE	FALSE
11	132	2.2	2.2		1	1	FALSE	TRUE
12	114	2.2	2.2		1	1	FALSE	TRUE
13	120	1	1		1	1	FALSE	TRUE
14	118	3.1	3.1		1	1	FALSE	TRUE

Tabela B.5: Ensaio 5

Ensaio 5								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	297	3.2	3.2		1	1	FALSE	TRUE
2	268	3.2	3.2		1	1	FALSE	TRUE
3	228	2.2	2.2		1	1	FALSE	TRUE
4	223	1	5		0	1	FALSE	FALSE
5	235	2.2	2.2		1	1	FALSE	TRUE
6	212	1	1		1	1	FALSE	TRUE
7	213	3.1	3.1		1	1	FALSE	TRUE
8	189	1	2.2		0	1	TRUE	FALSE
9	172	3.1	3.1		1	1	FALSE	TRUE
10	146	2.1	5		0	1	FALSE	FALSE
11	157	1.1	5		0	0	FALSE	FALSE
12	155	2.1	5		0	0	FALSE	FALSE
13	144	2.1	5		0	0	FALSE	FALSE
14	144	2.1	5		0	0	FALSE	FALSE
15	139	2.1	5		0	1	FALSE	FALSE
16	159	2.1	2.1		1	1	FALSE	TRUE
17	138	2.1	2.1		1	1	FALSE	TRUE

Tabela B.6: Ensaio 6

Ensaio 6								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	278	1	1		1	1	FALSE	TRUE
2	271	2.2	2.2		1	1	FALSE	TRUE
3	270	2.1	2.1		1	0	FALSE	FALSE
4	263	2.1	2.1		1	1	FALSE	TRUE
5	253	3.2	3.2		1	1	FALSE	TRUE
6	232	3.2	3.2		1	1	FALSE	TRUE
7	189	1	1		1	1	FALSE	TRUE
8	185	3.1	3.1		1	1	FALSE	TRUE
9	171	3.1	3.1		1	1	FALSE	TRUE
10	151	2.1	2.1		1	1	FALSE	TRUE
11	97	2.2	2.2		1	1	FALSE	TRUE

Tabela B.7: Ensaio 7

Ensaio 7								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	357	3.1	4	3.1	1	1	FALSE	TRUE
2	329	3.1	4		0	0	FALSE	FALSE
3	324	3.1	5		0	0	FALSE	FALSE
4	306	3.2	3.2		1	1	FALSE	TRUE
5	292	3.2	3.2		1	1	FALSE	TRUE
6	240	1	5		0	1	FALSE	FALSE
7	211	1	1		1	1	FALSE	TRUE
8	239	1	1		1	1	FALSE	TRUE
9	204	2.1	3.1		0	1	TRUE	FALSE
10	182	2.2	2.2		1	1	FALSE	TRUE
11	173	3.1	3.1		1	1	FALSE	TRUE
12	128	2.1	5		0	0	FALSE	FALSE
13	139	2.2	2.2		1	1	FALSE	TRUE
14	116	2.1	2.1		1	1	FALSE	TRUE

Tabela B.8: Ensaio 8

Ensaio 8								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	277	3.1	3.1		1	1	FALSE	TRUE
2	273	2.1	2.1		1	1	FALSE	TRUE
3	234	3.2	3.2		1	1	FALSE	TRUE
4	232	3.2	3.2		1	1	FALSE	TRUE
5	208	2.2	2.2		1	0	FALSE	FALSE
6	190	1	1		1	1	FALSE	TRUE
7	184	2.2	2.2		1	1	FALSE	TRUE
8	174	1	1		1	1	FALSE	TRUE
9	166	3.1	3.1		1	1	FALSE	TRUE
10	151	2.2	2.2		1	1	FALSE	TRUE
11	115	2.2	2.1		0	1	TRUE	FALSE

Tabela B.9: Ensaio 9

Ensaio 9								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	285	2.1	2.1		1	0	FALSE	FALSE
2	327	2.1	2.1		1	1	FALSE	TRUE
3	298	3.2	3.2		1	1	FALSE	TRUE
4	269	3.1	3.1		1	1	FALSE	TRUE
5	213	1	1		1	1	FALSE	TRUE
6	237	3.2	3.2		1	1	FALSE	TRUE
7	172	1	1		1	1	FALSE	TRUE
8	163	3.1	3.1		1	1	FALSE	TRUE
9	142	2.1	5		0	0	FALSE	FALSE
10	163	2.2	2.2		1	1	FALSE	TRUE
11	119	2.2	2.2		1	1	FALSE	TRUE
12	113	2.1	5		0	1	FALSE	FALSE
13	120	2.1	2.2		0	1	TRUE	FALSE

Tabela B.10: Ensaio 10

Ensaio 10								
Tent.	T(ms)	Obj	Class	Reclass	Class OK	Recolha OK	Total NOK	Total OK
1	299	3.1	4	3.1	1	1	FALSE	TRUE
2	266	3.1	4	3.1	1	1	FALSE	TRUE
3	245	2.2	4	2.2	1	1	FALSE	TRUE
4	231	2.1	2.1		1	1	FALSE	TRUE
5	211	3.2	3.2		1	1	FALSE	TRUE
6	186	3.2	3.2		1	1	FALSE	TRUE
7	160	1	1		1	1	FALSE	TRUE
8	158	1	1		1	1	FALSE	TRUE
9	135	2.1	2.1		1	1	FALSE	TRUE
10	118	2.2	3.1		0	1	TRUE	FALSE

Anexo C

Outros

C.1 Ficheiro Launch

```
<launch>

<node pkg="bin_picking"name="Keyboard"type="Keyboard"/>
<node pkg="bin_picking"name="Kinect"type="Kinect"/>
<node pkg="bin_picking"name="Fanuc"type="Fanuc"/>

<arg name="camera"default="camera"/>
<arg name="tf_prefix"default= />
<arg name="rgb_frame_id"
default="$(arg tf_prefix)/$(arg camera)_rgb_optical_frame"/>
<arg name="depth_frame_id"
default="$(arg tf_prefix)/$(arg camera)_depth_optical_frame"/>
<arg name="device_id"default="#1"/>
<arg name="rgb_camera_info_url"default= />
<arg name="depth_camera_info_url"default= />
<arg name="depth_registration"default="false"/>
<arg name="rgb"default="rgb"/>
<arg name="ir"default="ir"/>
<arg name="depth"default="depth"/>
<arg name="depth_registered"default="depth_registered"/>
<arg name="projector"default="projector"/>
<arg name="load_driver"default="true"/>
<arg name="publish_tf"default="true"/>
<arg name="rgb_processing"default="true"/>
<arg name="ir_processing"default="true"/>
<arg name="depth_processing"default="true"/>
<arg name="depth_registered_processing"default="true"/>
<arg name="disparity_processing"default="true"/>
<arg name="disparity_registered_processing"default="true"/>
<arg name="hw_registered_processing"default="true"/>
<arg name="sw_registered_processing"default="true"/>
```

```
<arg name="bond"default="false"/>
<arg name="respawn"default="$(arg bond)"/>
<arg name="num_worker_threads"default="4"/>
<group ns="$(arg camera)» <arg name="manager"
value="$(arg camera)_nodelet_manager"/>
<arg name="debug"default="false"/>
<include file="$(find rgbd_launch)/launch/includes/manager.launch.xml»
<arg name="name"value="$(arg manager)"/>
<arg name="debug"value="$(arg debug)"/>
<arg name="num_worker_threads"value="$(arg num_worker_threads)"/>

</include>

<include if="$(arg load_driver)"
file="$(find openni_launch)/launch/includes/device.launch.xml»
<arg name="manager"value="$(arg manager)"/>
<arg name="device_id"value="$(arg device_id)"/>
<arg name="rgb_frame_id"value="$(arg rgb_frame_id)"/>
<arg name="depth_frame_id"value="$(arg depth_frame_id)"/>
<arg name="rgb_camera_info_url"value="$(arg rgb_camera_info_url)"/>
<arg name="depth_camera_info_url"value="$(arg depth_camera_info_url)"/>
<arg name="depth_registration"value="$(arg depth_registration)"/>
<arg name="rgb"value="$(arg rgb)"/>
<arg name="ir"value="$(arg ir)"/>
<arg name="depth"value="$(arg depth)"/>
<arg name="depth_registered"value="$(arg depth_registered)"/>
<arg name="projector"value="$(arg projector)"/>
<arg name="respawn"value="$(arg respawn)"/>

</include>

<include file="$(find rgbd_launch)/launch/includes/processing.launch.xml»
<arg name="manager"value="$(arg manager)"/>
<arg name="rgb"value="$(arg rgb)"/>
<arg name="ir"value="$(arg ir)"/>
<arg name="depth"value="$(arg depth)"/>
<arg name="depth_registered"value="$(arg depth_registered)"/>
<arg name="projector"value="$(arg projector)"/>
<arg name="respawn"value="$(arg respawn)"/>
<arg name="rgb_p0rocessing"value="$(arg rgb_processing)"/>
<arg name="ir_processing"value="$(arg ir_processing)"/>
<arg name="depth_processing"value="$(arg depth_processing)"/>
<arg name="depth_registered_processing"
value="$(arg depth_registered_processing)"/>
<arg name="disparity_processing"value="$(arg disparity_processing)"/>
<arg name="disparity_registered_processing"
```

```
value="$(arg disparity_registered_processing)"/>
<arg name="hw_registered_processing"value="$(arg hw_registered_processing)"/>
<arg name="sw_registered_processing"value="$(arg sw_registered_processing)"/>

</include>

</group> <include if="$(arg publish_tf)"
file="$(find rgbd_launch)/launch/kinect_frames.launch»
<arg name="camera"value="$(arg camera)"/>
<arg name="tf_prefix"value="$(arg tf_prefix)"/>

</include>

</launch>
```