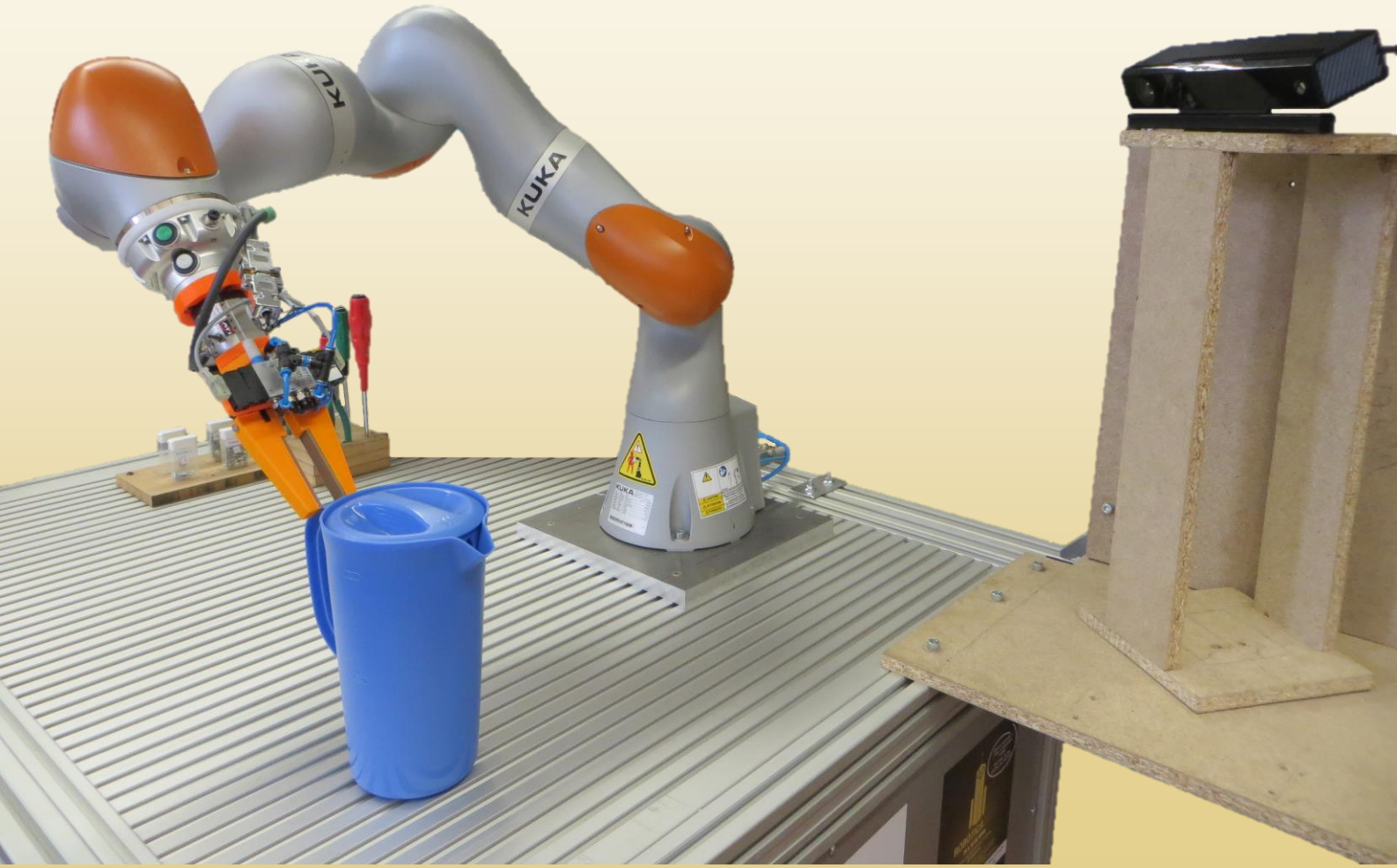


João Gonçalo Pires Ferreira da Silva

Object Segmentation and Classification from RGB-D Data

Dissertação de Mestrado em Engenharia Mecânica
na Especialidade de Energia e Ambiente

10/07/2017

· U C ·

Universidade de Coimbra

• U  C •

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEPARTAMENTO DE
ENGENHARIA MECÂNICA

Object Segmentation and Classification from RGB-D Data

Submitted in Partial Fulfilment of the Requirements for the Degree of Master in Mechanical Engineering in the speciality of Energy and Environment

Segmentação e Classificação de Objetos a partir de Dados RGB-D

Author
João Gonçalo Pires Ferreira da Silva

Advisor
Professor Pedro Mariano Simões Neto

Jury

President	**Professor Doutor Cristóvão Silva** **Professor Auxiliar da Universidade de Coimbra**
Vowel	**Doutor Nuno Alberto Marques Mendes** **Investigador Auxiliar da Universidade de Coimbra**
Advisor	**Professor Doutor Pedro Mariano Simões Neto** **Professor Auxiliar da Universidade de Coimbra**

Coimbra, July, 2017

“At the desk where I sit, I have learned one great truth. The answer for all our national problems - the answer for all the problems of the world - come to a single word. That word is education.”

Lyndon B. Johnson, in Congressional Record, 2001.

To my parents.

ACKNOWLEDGEMENTS

First, I would like to thank my parents and the rest of my family for all the support and kindness, that they provided. With them, I was able to achieve my goals and dreams throughout my life.

I would also like to express my gratitude to my advisor Professor Pedro Neto, for his patience, encouragement and guidance. With his help, I was able to develop this dissertation in my own way, but his advice was fundamental in the completion of this dissertation.

Then, I would like to thank my colleagues from the Collaborative Robotics Laboratory of the University of Coimbra for all the patience, support and advice. They received me with open arms from day one, always treating me as a friend and helping me when they could.

Finally, I would like to thank my friends for always being there when I needed them, with their patience, cheerfulness and support.

ABSTRACT

Object classification is a key factor in the development of autonomous robots. Object classification can be greatly improved with previous reliable segmentation and feature extraction. With this in mind, the main objective of this dissertation is to implement an object classification algorithm, capable of classifying objects from the Yale-CMU-Berkeley (YCB) object and model set, through the use of a novel unsupervised feature extraction method from red, green, blue and depth (RGB-D) data and feedforward artificial neural networks (FFANNs).

In the method presented here, after the acquisition of data from an RGB-D camera, noise is removed and the objects in the scene are isolated. For each isolated object, k-means clustering is applied to extract a global main colour and three main colours. Three scores are computed based on the fitting of primitive shapes (cylinder, sphere or rectangular prism). Object dimensions and volume are estimated by calculating the volume of the best primitive shape previously fitted. Then with these features, FFANNs are trained and used to classify these objects.

Experimental tests were carried out in 20 objects, from the YCB object and model set and results indicate that this algorithm has a recognition accuracy of 96%, with five objects in the workspace at the same time and in random poses.

Also, a method of calculating the location of an object, based on the location of the geometric centre, of the best primitive shape previously fitted is developed.

Keywords: Machine learning, Features, Object classification, Microsoft Kinect, Neural networks.

RESUMO

A classificação de objetos é um fator chave no desenvolvimento de robôs autónomos. A classificação de objetos pode ser grandemente melhorada com uma anterior segmentação e extração de características confiáveis. Com isso em mente, o principal objetivo desta dissertação é implementar um algoritmo de classificação de objetos, capaz de classificar objetos do conjunto de objetos e modelos de Yale-CMU-Berkeley (YCB), através do uso de um novo método de extração de características não supervisionado a partir de dados de vermelho, verde, azul e profundidade (RGB-D) e de redes neuronais artificiais do tipo *feedforward* (FFANNs).

No método aqui apresentado, após a aquisição de dados a partir de uma câmara RGB-D, o ruído é removido e os objetos na cena são isolados. Para cada objeto isolado, agrupamento *k-means* é aplicado para extrair uma cor global e três cores principais. Três pontuações são calculadas com base no encaixe de formas primitivas (cilindro, esfera ou prisma retangular). As dimensões do objeto e volume são estimados calculando o volume da melhor forma primitiva ajustada anteriormente. De seguida, com essas características, FFANNs são treinadas e usadas para classificar esses objetos.

Testes experimentais foram realizados em 20 objetos, do conjunto de objetos e modelos de YCB e os resultados indicam que este algoritmo tem uma precisão de reconhecimento de 96%, com cinco objetos no espaço de trabalho ao mesmo tempo e em poses aleatórias.

Também é desenvolvido, um método de cálculo da localização de um objeto, com base na localização do centro geométrico, da melhor forma primitiva ajustada anteriormente.

Palavras-chave: Aprendizagem automática, Características, Classificação de objetos, *Microsoft Kinect*, Redes neuronais.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE AND ACRONYMS

Nomenclature

The notation that we are going to follow throughout this dissertation, is going to be as described:

- Bold capital letters are used to denote matrices, for example ${}^{1}_{T}\mathbf{R}$ represents the first rotation matrix;
- Bold and italic small letters are used to denote vectors, for example $\boldsymbol{c}_{\mathrm{p}_j}$ represents the red, green and blue (RGB) colour vector of point p_j;
- Italic small letters or words are used to denote scalars, for example n_{pct} represents the total number of acquired point clouds;
- Small letters are used to denote points, for example p_{M} represents a generic point of the merged point cloud;
- Italic capital letters are used to denote point clouds, for example PC_{M} represents the merged point cloud.

Acronyms

2D – Two-dimensional

3D – Three-dimensional

ANNs – Artificial neural networks

CAD – Computer-aided design

CKM – Convolutional k-means descriptors

CNNs – Convolutional neural networks

DL – Dictionary learning

ELMs – Extreme learning machines

FFANNs – Feedforward artificial neural networks

HMP – Hierarchical matching pursuit

HMP2D – Hierarchical matching pursuit for 2D voxel data

HMP3D – Hierarchical matching pursuit for 3D voxel data

LIDAR – Light detection and ranging

LinSVM – Linear support vector machine

PCA – Principal component analysis

RANSAC – Random sample consensus

RBF – Radial basis function

R-CNNs – Combination of region proposals and CNNs

RFs – Random forests

RGB – Red, green and blue

RGB-D – Red, green, blue and depth

RNNs – Recursive neural networks

SIFT – Scale invariant feature transform

SURF – Speeded-up robust features

SVM – Support vector machine

YCB – Yale-CMU-Berkeley

1. INTRODUCTION

Nowadays, automation and robotics are the corner stones of many industries. Their involvement in the workspace contributes to higher production and accuracy. At the same time, robots are moving into our households, since they can have simple and repetitive tasks delegated to them.

This makes robot autonomy a major research topic. One of the main reasons is the fact that, robots work more and more alongside humans and humans may change the environment, in many different and unpredictable ways, so a robot needs to be able to constantly adapt to their surroundings. It means that they need to be able to make autonomous decisions. Some examples of such behaviour are autonomous vehicles that recognize traffic signs and react in accordance, robotic vacuum cleaners and robotic lawn mowers that avoid obstacles in real time.

In the last few years, the view on the subject of industrial robots has changed, from one where robots work separated from humans, many times in a caged environment, to one where they work alongside humans and collaborate with them to perform tasks.

In order to be able to do that, they need to be able to classify objects, tools or parts that a human might need and obstacles in their path. They also need localization and navigation skills, in order to know where these elements are and/or where to put them. By doing this, collaborative robots can act autonomously in order to improve productivity, by using the robot's best assets which are accuracy and the ability to perform monotonous task very fast.

Collaborative robots may also be called to handle objects never seen before. For example, a human may want to exchange a tool with a neighbour robot. In [1], the authors were interested in considering the problem of grasping novel objects using computer vision and machine learning.

On-board vision sensors as well as additional red, green, blue and depth (RGB-D) devices, such as Microsoft Kinect, can be used to obtain a three-dimensional (3D) model of the target object. Then, machine learning techniques can be exploited to find good

candidates of grasping positions based on the geometric structure of the object (using a two-dimensional (2D) and/or a 3D model), robot trials and learning from the user.

In the first case, the 3D point cloud of the object is analysed with grasping representations, as in [2] or rules for object grasping based on models, as in [3].

In the second case, the robot tries to autonomously grasp the target object. Positive or negative results will be used in a machine learning framework, for example using reinforcement learning [4], to improve future tasks.

Finally, in the last case, computer vision algorithms can be exploited to detect and understand grasping positions, by observing human collaborators [5].

1.1. Problem Specification and Challenges

The problem with vision-based classification is the reliability, which can be difficult to achieve depending on several factors. Object classification can be greatly improved with previous reliable segmentation, which allows isolation of the objects and feature extraction. This feature extraction serves as input for a classification algorithm.

Recent deep learning methods present state of the art results for classification, no features are required and the objects are classified from the raw red, green and blue (RGB) or RGB-D data. However, deep learning requires large amounts of training data. Object features will allow to improve the deep learning accuracy when there is less training data, by giving it more inputs. In addition, these features can also be used as a unique input for any supervised classifier that is expected to output accurate results with small amounts of training data.

Effective segmentation and feature extraction are achieved with no previous knowledge of the objects and no training data. In general, there are six major difficulties in the segmentation and feature extraction process for object classification:

- No information about objects pose (position + orientation) in the scene;
- Each object can have multiple poses in the scene, so that the same object may look different depending on the perspective the image frame is captured. This is more critical when we have large libraries of objects to classify since this increases the size of those libraries;

- Only part of the object is captured by the camera, depending on the object's relative pose in relation to the camera;
- Feature extraction can be made difficult depending on the lighting conditions and pose, because they affect the colour that is captured and since the camera only captures the part of the object facing it, the captured colour can have a wide variation;
- The background can be dynamic;
- Clustered scenes can have possible occlusions.

In face of the above, several challenges can be pointed out:

- Creating an unsupervised segmentation and feature extraction algorithm that is robust enough to accurately eliminate noise, isolate objects and create distinctive features. This has to be done in unstructured environment, without previous knowledge about the object, pose, background, lighting conditions and others and from single RGB-D camera;
- Association of a primitive shape to each object;
- Extraction of colour features;
- Estimation of objects volume and location;
- Use of a robust classification algorithm capable of correctly identifying the objects in real-time, having shape, colour and volume features as input.

1.2. Related Work

Reliable object classification is today a major challenge in robotics field. The appearance of novel sensing technologies allows the recording of quality RGB and depth images (RGB-D). Depth information is relatively invariant to lighting or colour variations, allows background subtraction, video ground truth annotation via 3D reconstruction, and improved object classification. Nevertheless RGB-D cameras, such as Microsoft Kinect, are still affected by noise and missing depth information, due to for example, reflective properties of materials as well as their coatings, that interfere with the camera by overexposing the depth image [6]. Despite of this, their cost makes them worthwhile alternatives, for prototyping before committing to a more expensive sensor.

Existing object recognition algorithms for RGB-D images use features such as scale invariant feature transform (SIFT) [7], spin images [7], [8], visual, shape and geometrical features [9]. Other methods generalize this idea and combine RGB-D image features such as size, 3D shape and depth edges [10].

Krainin et al. developed a system to build a 3D surface model of objects by grasping the object and moving it in front of an RGB-D camera [11]. The idea was to allow a robot equipped with an RGB-D camera to create models of new objects, by interacting with them using different grasps and combining the information from several snapshots.

Unsupervised feature learning methods suffered a massive evolution in the last few years, allowing for the improvement of object classification and pattern recognition in general. Deep learning methods have been extensively studied for RGB images. However, their implementation for RGB-D images is still matter of study [12].

Blum et al. introduced convolutional k-means descriptors (CKM) for RGB-D data, in which they learn features in an unsupervised fashion [13]. They adapted the unsupervised learning approach proposed in [14] to process RGB-D images. They begin by learning features only in the vicinity of interest points with the depth information being added later. To detect the interest points, they opted for speeded-up robust features (SURF) [15] due to its computational efficiency and quality of results. The descriptor is then extracted similarly to SURF's, and according to the authors, the number of learned features is typically high, which may then require a dimensionality reduction, using principal component analysis (PCA), for input in a linear support vector machine (LinSVM) classifier. They evaluated their work using the Washington RGB-D Object Dataset [7].

Bo et al. proposed unsupervised feature learning from grey scale intensity, RGB, depth, and surface normal [16]. Features are then used in hierarchical matching pursuit (HMP) with two layers, to learn hierarchical feature representation, then LinSVM is used as the classifier. The algorithm was tested on five datasets, although here we are only going use the test on the Washington RGB-D Object Dataset [7], since it's one of the most widely available.

Convolutional neural networks (CNNs) have recently been shown to be remarkably successful for recognition on RGB images [17]. In [18], Farabet et al. introduced a model for scene parsing, that is based on multi-scale CNNs for feature extraction, other similar works, use a combination of region proposals and CNNs (R-CNNs), such as [19].

Also, features extracted from CNNs and colorization methods to represent depth in a 2D image have demonstrated good results in RGB-D object classification [20].

Lai et al. introduced a new variant of HMP, to use with point clouds directly, called hierarchical matching pursuit for 3D voxel data (HMP3D) [21]. They use a LinSVM classifier trained using a synthetic dataset of virtual scenes, generated using computer-aided design (CAD) models. They evaluated their algorithm using the Washington RGB-D Scenes Dataset [7]. It was concluded that in some situations, HMP3D may not even require to be combined with hierarchical matching pursuit for 2D voxel data (HMP2D), in order to improve the results.

A deep learning method for object classification, based on a combination of CNNs and recursive neural networks (RNNs) for learning features and classifying RGB-D images, is presented in [22]. The CNNs layer learns low-level translationally invariant features which are the input to fixed-tree RNNs to compose higher order features. Results indicate state of the art performance in category recognition on the Washington RGB-D Object Dataset [7]. With the same method being applied in [23], along with support vector machine (SVM) to improve accuracy of classification in the same dataset.

A two stream CNNs (RGB and depth) for object recognition is proposed in [24], using the Washington RGB-D Object Dataset [7] and implementing their work in the Caffe framework [25]. They were able to achieve an accuracy of 84.1% using only RGB and 83,8% using only depth, which were improved to 91.3% when combined, for category recognition. Another method in [26], used CNNs to learn features, which were inputs to a LinSVM classifier to classify objects from Washington RGB-D Object Dataset [7], this method achieved 91.4% accuracy on category recognition. CNNs also demonstrated superior performance in detecting object grasping areas [27].

Extreme learning machines (ELMs) can be used for object classification. These are very simple feedforward artificial neural networks (FFANNs) with a single layer of hidden neurons. In [28], ELMs are used both in the feature extraction and the classification parts of the algorithm, achieving an accuracy of 89.3% in the category recognition task of the Washington RGB-D Object Dataset [7]. Also using the same dataset, random forests (RFs) [29] and SVM [30], can be used as classifiers, using these they were able to achieve 88.1% and 75.6% accuracy, respectively, in category recognition.

As can be seen in [31], the problem of volume estimation with the Microsoft Kinect can be approached with reasonable accuracy for medium size objects. Their setup consisted of a turn table with markers, on which the object is placed and a Microsoft Kinect camera takes several snapshots of the object from different orientations. This allowed them to get a complete view of the object, in order to get the best possible point cloud for volume estimation.

Another approach to volume estimation is the one used by B. Q. Ferreira et al. in [32], it consists of capturing one stationary image, then delimitating the object's planes, after getting the vertices of that object and computing the volume using these vertices.

One approach that can improve the volume estimation, is proposed by D. Schiebener et al., in [33], using the Yale-CMU-Berkeley (YCB) object and model set [34], these authors proposed an approach to complete an object's model, when a complete view of the object is not available. This approach is based on the assumption that most objects have symmetries and uses these symmetries to complete the view of an object. Although this approach is useful in some situations, this is not always the case, especially if the object has no plane of symmetry. This also shows some of the difficulties of estimating the hidden parts of an object, since these can have very different shapes.

In our setup though, we are only getting pictures from one stationary position, making the volume estimation not very precise, since we cannot see the hidden part of an object. But on the other hand, since we don't need a very precise estimation, this makes the process of image acquisition and setup much easier.

An interesting study is presented by A. Broad and B. Argall [35], in here they use CNNs, as a classifier for object recognition in the YCB object and model set [34]. By taking the object's location and size, they define bounding boxes, after this they transform these bounding boxes from RGB-D space to RGB space and classify each object by passing these RGB bounding boxes through these CNNs.

Another method of obtaining a point cloud for object recognition is using light detection and ranging (LIDAR), as proposed by Artur Maligo and Simon Lacroix in [36]. Such method is not going to be developed in this paper, although it presents another possible method to obtain data for object recognition and can be applied in terrestrial autonomous vehicles, such as self-driving cars, as in [37] which achieved an accuracy of 98.5% in classifying using LIDAR data as input.

Considering similar work to our own, which is instance or object recognition or classification from RGB-D data, but with different datasets, since works with the same dataset, namely are going to be analysed further along in this paper, we get Table 1.1. This table has the reference to the paper containing that method of classification, the number of objects with which the method was test, average number of real samples per class used in training, the maximum classification accuracy, the classifier used and the year of the publication of that paper.

Table 1.1. Comparison between methods of classification.

Reference	Number of objects	Average number of real samples per class	Classification accuracy [%]	Classifier	Year
[13]	300	109	92.1	LinSVM	2012
[16]	300	93	92.8	LinSVM	2013
[20]	300	93	94.1	LinSVM	2015
[38]	300	117	96.9	Dictionary learning (DL)	2015
[39]	300	117	94.4	Radial basis function (RBF) kernel SVM	2014
[40]	300	117	96.1	Statistical similarity (Their own classifier)	2017
[41]	300	100	94.8	LinSVM	2015
[42]	300	100	97.2	ELMs	2016
[43]	300	93	93.5	ELMs	2015
[44]	300	100	99	CNNs	2015
[45]	300	117	70	LinSVM	2016
[46]	153	33	42.1	Ensemble classifier	2015
[47]	5	360	94.8	SVM	2014

From [13] until [46] in Table 1.1, we considered only the tests on the Washington RGB-D Object Dataset [7], since it's one of the most widely available and allows for comparison between methods. In [47] used its own object set and the accuracy is the average accuracy of all objects.

Applications of this can be seen in [48], in which robots execute several tasks of manipulation of household objects. In the case of the robot Cosero, it actually is equipped with a Microsoft Kinect RGB-D camera. This robot shows some of possible applications of our algorithm, such as baking an omelette, which requires the robot to know where several needed objects, ingredients and cooking utensils, are in order to perform this task.

Another application is in [34] and Figure 1.1, here we see a robot attempting to grasp a power drill. This robot can be an assistant to a factory worker and give this worker the tools that he needs, in order to do that, the robot needs to know what objects are needed and where they are, this can be achieved with the use of an object classification algorithm.

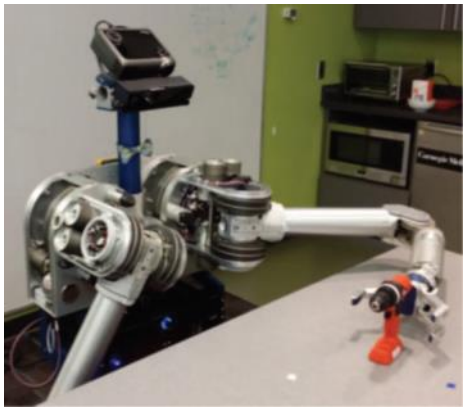


Figure 1.1. Robot grasping power drill of object set [34].

This analysis is a reference of existing studies, mainly in the last 5-6 years and allows us to conclude, that in many of these approaches, the classification uses deep learning, although these methods present good results, they require large amounts of training data.

The possibility of object classification using other features, extracted in an unsupervised fashion, is not considered with the sufficient detail, especially considering that these features could allow to reduce the required training data.

1.3. Proposed Approach and Overview

This dissertation, introduces a novel unsupervised feature extraction method from RGB-D data that outputs a primitive shape, that best fits each object, the four main colours and the estimated volume and location of each object. Also introduces the application of this method in combination with FFANNs, in order to create an object classification algorithm capable of identifying 20 objects from the YCB object and model set [34], in an unstructured environment, with very few real training data and with more than one object in the workspace at the same time.

This algorithm, in Figure 1.2, works by first acquiring the point clouds and merging them together. Inaccessible points and noise are removed. Due to the relative referential frame, the point cloud is rotated in three different stages, first to the vertical position, then to the horizontal position and finally we optimize the table plane rotation to the horizontal, to achieve better results in the following stages of the algorithm.

We trim this resulting configuration, with not much more than the objects on top of the table. Then the table plane is removed, based on a histogram to ensure the accuracy of the method. After this, object segmentation is done, by finding regions in the point cloud containing points and surrounded by empty spaces, in order to separate each object in to its own point cloud, so that objects in the scene are isolated.

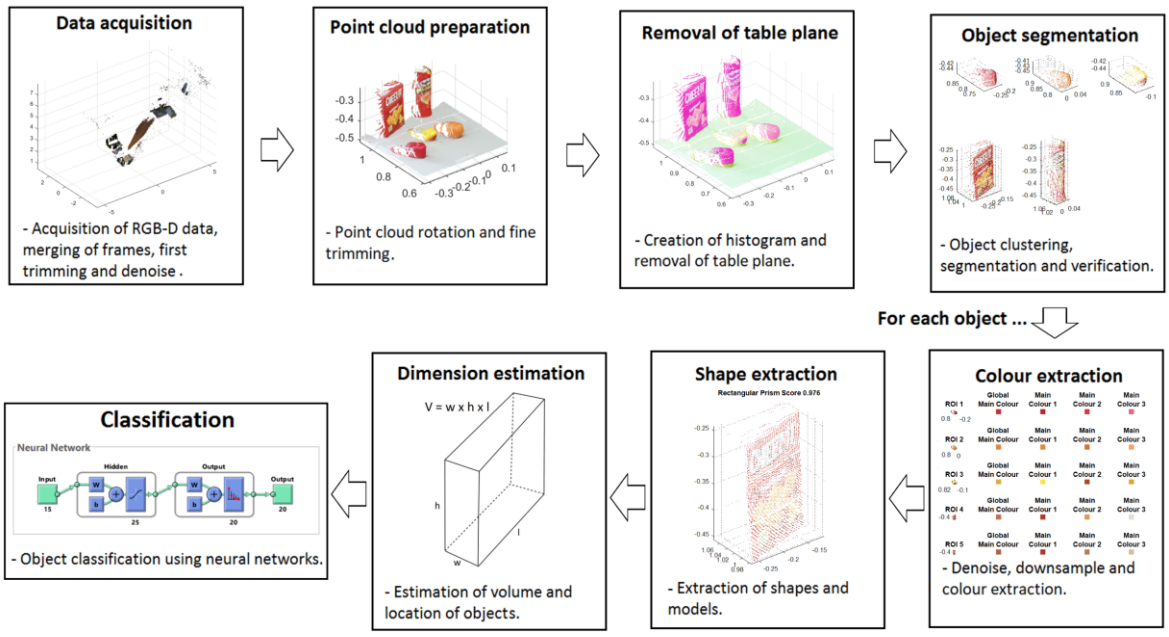


Figure 1.2. Overview of the proposed approach.

As we can see in Figure 1.2, denoise and downsampling are performed to improve the accuracy and speed of the algorithm. After that feature extraction can begin, for each isolated object k-means clustering is applied to extract a global main colour and three main colours.

Then three scores are computed based on the fitting of primitive shapes (cylinder, sphere and rectangular prism), and the best shape and model are chosen based on the primitive shape that has the highest score. Object's volume is estimated by calculating the volume of the model or the point cloud. Then the location is estimated, by calculating the geometric centre of the model or point cloud, in accordance with the volume estimation.

Experimental tests were carried out in 20 objects from the YCB object and model set [34] in an unstructured environment. It was demonstrated that:

- The object's primitive shapes (cylinder, sphere and rectangular prism), main colours (a total of four) and volume are estimated;
- The proposed method allows to accurately and in an unsupervised fashion, isolate the objects in the scene and reduce the noise;
- The extracted features can be used as main input for a classification algorithm;
- The location of the objects can be reasonably obtained using this algorithm, after proper calibration;
- The algorithm was able to achieve 96% accuracy, with five objects in the workspace at the same time and in random poses, with just an average of 13 real samples per object in training, much less than the other previously referred methods, leading to a training time of around 6 minutes.

2. OBJECT SEGMENTATION AND CLASSIFICATION

Knowing the challenge that we face and the proposed approach to solve it. The objective of this section is to explain in depth the algorithm, with its scheme in Appendix A.

2.1. Point Cloud Acquisition and Processing

2.1.1. Acquiring

In order to improve the quality of the RGB-D data, we obtain n_{pct} point clouds in a row, from Microsoft Kinect V2. The acquiring procedure is made statically, that is, the objects and the sensor stay in the same pose during data acquisition. These point clouds are then merged using a box grid filter with a small cell size, just like in downsampling operations.

Assuming a cell size of s_{c} and n_{pct} point clouds $PC_{n_{\mathrm{pc}}}$, $n_{\mathrm{pc}} \in [1, n_{\mathrm{pct}}]$, each with overall dimensions of $\left[s_{\mathrm{x},n_{\mathrm{pc}}}, s_{\mathrm{y},n_{\mathrm{pc}}}, s_{\mathrm{z},n_{\mathrm{pc}}}\right]$ being X, Y and Z the respective Cartesian axis, the total number of divisions for each point cloud will be $\left[\frac{s_{\mathrm{x},n_{\mathrm{pc}}}}{s_{\mathrm{c}}}, \frac{s_{\mathrm{y},n_{\mathrm{pc}}}}{s_{\mathrm{c}}}, \frac{s_{\mathrm{z},n_{\mathrm{pc}}}}{s_{\mathrm{c}}}\right]$. In each of these divisions, colours and normal of all contained points are averaged and become a single occurrence in the merged point cloud, PC_{M}, as shown in Figure 2.1, where the position of that occurrence is the geometric centre of the respective division.

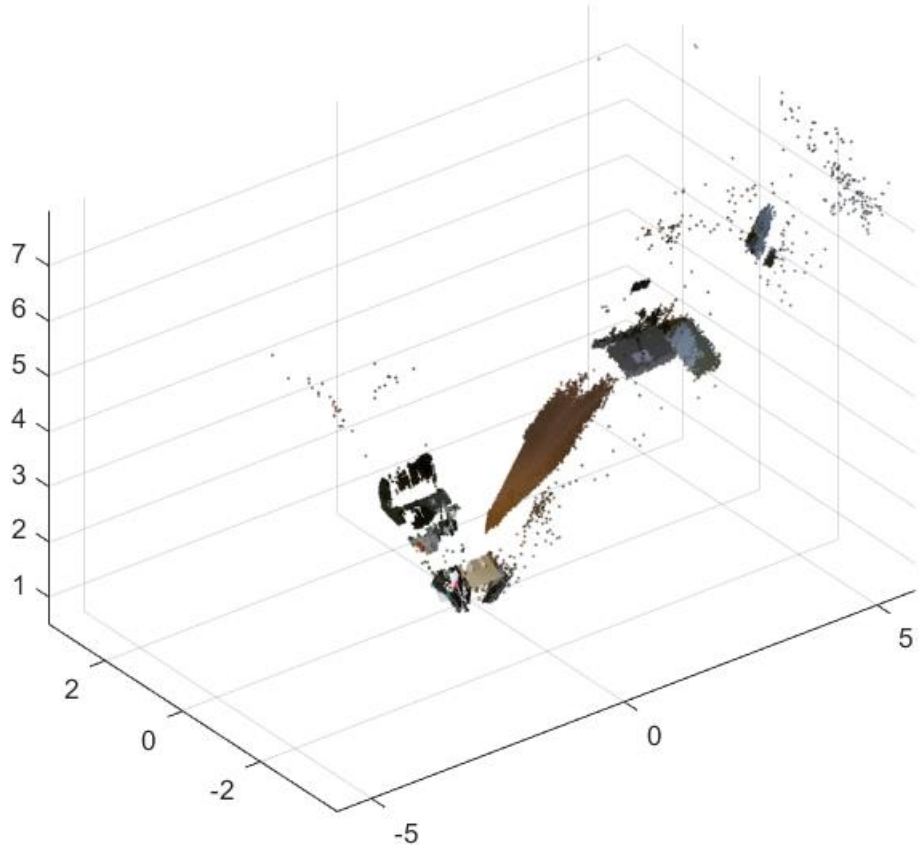


Figure 2.1. Point cloud, PC_{M} , after merging.

The small cell size, s_{c}, ensures that most data will be kept, but without duplicates or redundancies, leading to a point cloud with less points and allowing for faster processing. After this, the invalid points are removed using a MATLAB function, in order to eliminate acquisition errors. The merging procedure also helps avoiding acquisition errors and noise by averaging the values of the n_{pct} point clouds.

2.1.2. First Trimming

After obtaining and merging these n_{pct} point clouds, into PC_{M}, a trimming operation designed to remove useless data for posterior grasping algorithms is performed, that is, the point cloud is trimmed based on the range and position of a manipulator, for example, a robot. In this stage, we assume a sphere with a radius, r and we find every point, $\mathrm{p_M} \in PC_{\mathrm{M}}$ with coordinates $[x_{\mathrm{p_M}}, y_{\mathrm{p_M}}, z_{\mathrm{p_M}}]$, inside it. In Equation (2.1), the radius is referred by r and the zero indexed constants, refer to centre of the trimming sphere.

$$\left(x_{\mathrm{p_M}} - x_0\right)^2 + \left(y_{\mathrm{p_M}} - y_0\right)^2 + \left(z_{\mathrm{p_M}} - z_0\right)^2 \leq r^2 \tag{2.1}$$

The resulting point cloud PC_{T}, shown in Figure 2.2, is significantly smaller than PC_{M}, both in terms of number of points and overall size, but contains only the points inside this sphere, which are the useful points that the manipulator can grasp. This makes the algorithm faster, since we are reducing significantly the number of point that it has to analyse.

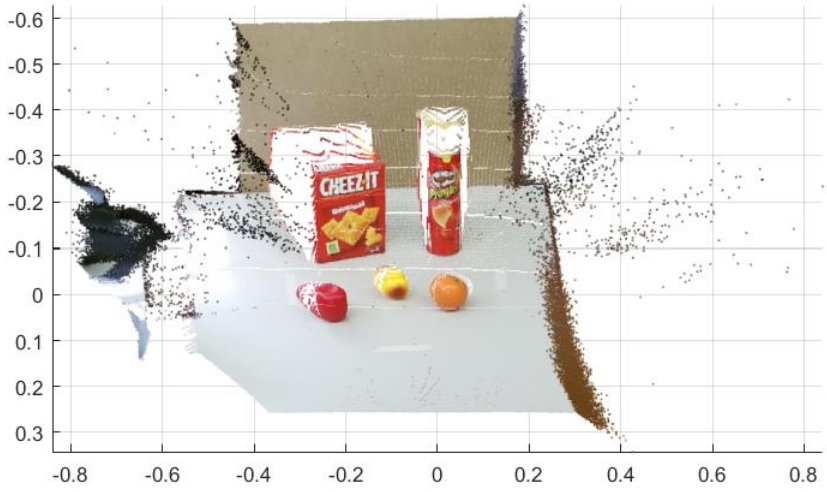


Figure 2.2. Point cloud, PC_{T}, after first trimming.

With this trimmed point cloud, PC_{T}, we execute a denoise operation which finds the mean distance d_{n} of the nearest n_{n} number of points, and then compares it to the distance d_{d}, which is a function of n_{d} number of the standard deviations, from the mean distance, μ_d, to all neighbouring points. By doing this, we are effectively removing sparse data and some isolated points that tend to stack near the trimmed surface, if the condition in Equation (2.2) is not met.

$$d_n \leq d_d = \mu_d + n_d \times \sigma_d \tag{2.2}$$

Here σ_d is the standard deviation of the distance to all neighbouring points. Large values of n_{n} contribute to more intense point removal and increased risk of useful point removal. Small values of n_{n} lead to a moderate point removal but may fail to remove noise.

2.1.3. Rotations

This denoised point cloud is then aligned in 3 rotations, as we can see in Figure 2.3, and Equation (2.3).

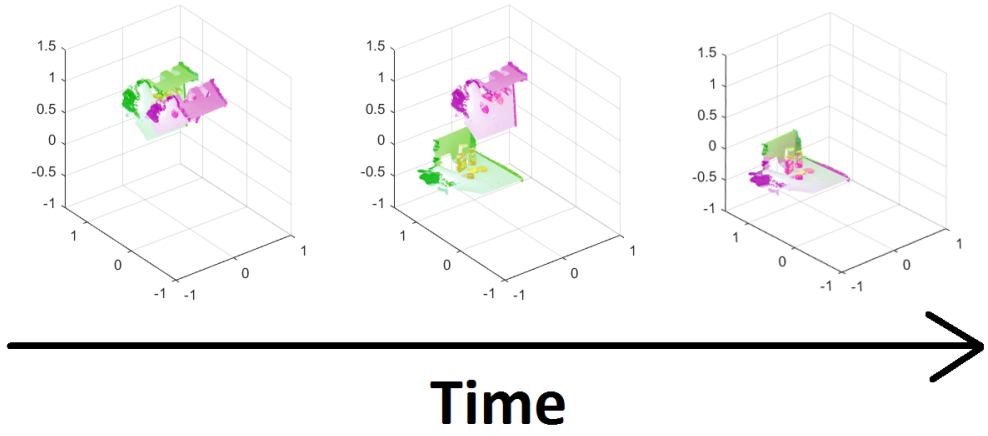


Figure 2.3. Rotations.

$$PC_{\mathrm{R}} = {}^{1}_{T}\mathbf{R}\ {}^{2}_{1}\mathbf{R}\ {}^{R}_{2}\mathbf{R}\ PC_{\mathrm{T}} \tag{2.3}$$

Where PC_{T} refers to the point cloud after first trimming, ${}^{1}_{T}\mathbf{R}$ to the first rotation matrix, ${}^{2}_{1}\mathbf{R}$ to the second rotation matrix, ${}^{R}_{2}\mathbf{R}$ to the third rotation and PC_{R} refers to the point cloud after all rotations are applied to it.

The first rotation uses a random sample consensus (RANSAC) algorithm [49], to fit a plane to the input point cloud. RANSAC and its adaptations/variations are all iterative methods, the implementation of RANSAC aims at finding the most representative set of data by finding the model specified, that contains the biggest number of inliers, points between the model and a tolerance.

From the plane model that we fitted, we extract its normal and use a Y-aligned vector to align the point cloud vertically. We compute the angles between the two vectors, after that we compute the equivalent rotation matrix, ${}^{1}_{T}\mathbf{R}$, which, we then use to rotate the point cloud. The result of this operation is a vertically aligned point cloud, as if the table were a wall.

The second rotation, rotates the point cloud to a horizontal position. By taking the Y-aligned vector and defining a new Z-aligned vector, we execute the same rotation procedure as explained before, this time using ${}^{2}_{1}\mathbf{R}$. This two stage rotation, is necessary in order to make sure that the point cloud points upwards and not downwards, when the Microsoft Kinect V2 angle with the horizontal is high.

The third rotation, ${}^{R}_{2}\mathbf{R}$, iteratively tries to optimize the fitting of a plane, that is in the form of $ax + by + d = z$, to the lowest Z-coordinate of points of each cell in a grid, and remove the points that are more than a n_z number of standard deviations away, from the mean distance to the plane along the Z-axis, μ_z. By repeating this procedure, we are trying to isolate the table plane since it is the biggest provider of points and it contains the lowest points. After this we apply the rotation ${}^{R}_{2}\mathbf{R}$.

To explain this rotation in more detail, we create a 2D grid, and place the minimum of coordinate along the Z-axis of all points of each cell, in its corresponding cell, in order to get an approximation of the floor plane, since this has the lowest points. We create a scatter plot of Z-coordinates with these points and iteratively fit a plane to this scatter plot. Then we remove points, it they are a distance d_z equal to an average, μ_z, plus a number of standard deviations away, n_z, from the plane along the Z-axis and then repeat this task. If this number of executions is too small, the plane quality will be poor, not representative of the table, if it's too large, we risk removing most of the points ensuing a poor result. The distance, $d_{\mathrm{p_R}}$, from a point $\mathrm{p_R}$ of the scatter plot, with coordinates $[x_{\mathrm{p_R}}, y_{\mathrm{p_R}}, z_{\mathrm{p_R}}]$ to the plane defined above, is equal to Equation (2.4).

$$d_{\mathrm{p_R}} = ax_{\mathrm{p_R}} + by_{\mathrm{p_R}} + d - z_{\mathrm{p_R}} \tag{2.4}$$

Considering all points, σ_z is the standard deviation of $d_{\mathrm{p_R}}$ of these points, and they are kept if the condition in Equation (2.5) is met.

$$d_{\mathrm{p_R}} < d_z = \mu_z + n_z \times \sigma_z \tag{2.5}$$

After this the rotation ${}^{R}_{2}\mathbf{R}$ is applied, taking in to account the normal of the last fitted plane and a Z-aligned vector.

This final operation verifies the alignment of the table plane with the horizontal, since the table is the biggest provider of points and has the lowest ones. We can use that to detect the alignment of the normal and flip it if needed. In order to make the table plane align with the horizontal as best as possible, giving a better input to the table segmentation algorithm. This allows for the Microsoft Kinect V2 to have its pose changed and be able to compensate it, without requiring a fixed pose.

2.1.4. Fine Trimming

After being properly aligned, a new trimming operation is performed. Since we only want to process the objects on the table, the trimming area is defined as a rectangular prism, that contains all the objects on the table, as in Figure 2.4.

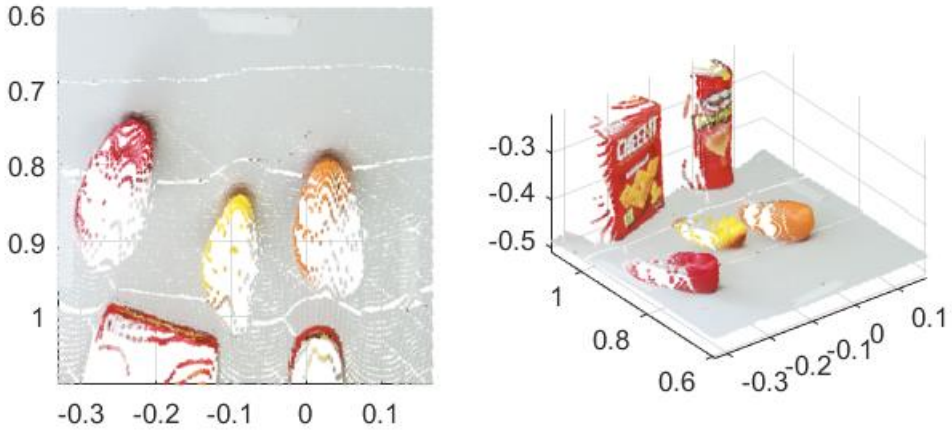


Figure 2.4. Point cloud after fine trimming.

Our trimming function, used previously, is able to trim the space according to a sphere, as used previously but it can also trim according to a cube, rectangular prism and even an ellipsoid in any orientation. By giving the appropriate number properties: a shape chosen from the ones mentioned before, a coordinates vector 1×3 for the geometric centre of the shape, a length(s) which are a scalar for the sphere and cube or 1×3 vector for the rectangular prism and ellipsoid and an orientation vector containing the angles relative to the original frame, 3×1 or 1×3, if desired.

2.1.5. Removal of Table Plane

The next step involves the removal of the table plane or table segmentation, as seen in Figure 2.5, the green being what was removed and the pink being what was kept. By plotting a histogram, $n_{\text{bin}} \in [1, n_{\text{bins}}]$, of the counts of points at each coordinate along the Z-axis, as in Figure 2.6, with $binwidth$ of bin width and then organizing the points by bins. After that, locating the trimming bin, bin_{trimming}, to a bin that is bin_{offset} above the maximum number of points bin, which we define as bin_{max}, as in Equation (2.6). Following the trimming, we obtain a new point cloud which is the point cloud after table segmentation PC_S, $PC_S \in [bin_{\text{trimming}}, bin_{n_{\text{bins}}}]$.

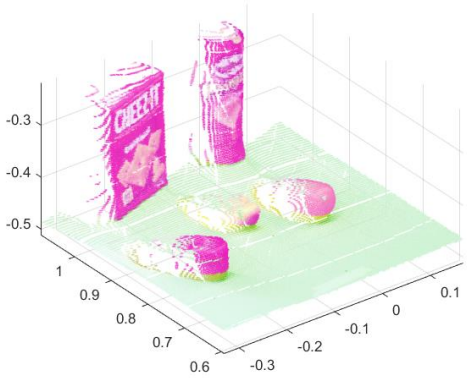


Figure 2.5. Table segmentation.

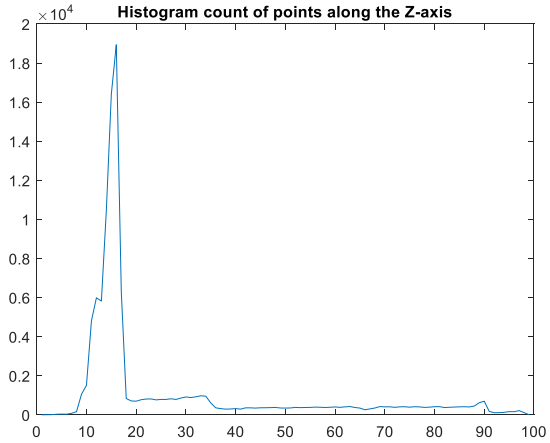


Figure 2.6. Histogram counts of points at each coordinate along the Z-axis.

$$bin_{\text{trimming}} = bin_{\max} + bin_{offset} \tag{2.6}$$

The bin_{offset} is due to the fact that, with objects on the table the Microsoft Kinect V2 distorts the table plane, creating peaks and valleys that need to be eliminated, thus the existence of this offset, since the bin with most points sometimes is inside the table. The accuracy and reliability of the method increased substantially with this method.

We also tried this operation performed based on the plane that fits most of the points, since the table is the biggest provider of points it should be the table, again using RANSAC a plane was fitted. With this plane, we got the Z-coordinate of this plane relative to the camera and separated the points which are above and below this plane. Here we should have got a reasonable approximation of the table plane. However, there were some occurrences in which the table wasn't removed accurately or completely. It was clear that

most problems were due to the fact that the plane ended up inside the table or too high on the table ignoring smaller objects. So, we went with the first approach.

2.1.6. Object Segmentation

After a second denoise operation, the point cloud is now ready to be segmented by finding clusters of data, each of these representing a possible object. By removing the Z-coordinate, dividing the resulting 2D space in cells, as in Figure 2.7, and placing a count of points in each cell, we begin to check groups of non-empty adjacent cells which we define as clusters.

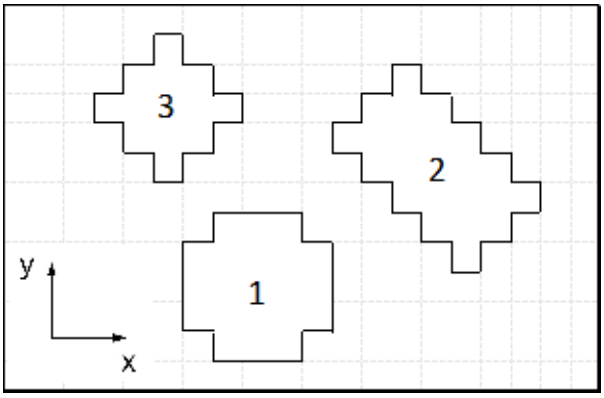
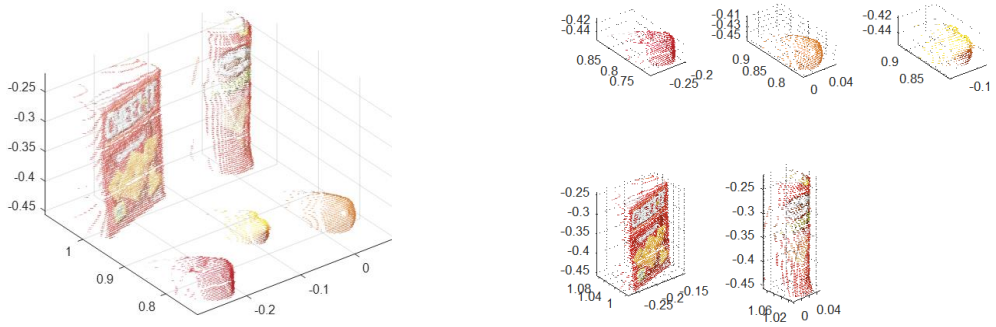


Figure 2.7. Clustering example.

Each of these groups are point cloud objects $PC_{O,l}, l \in [1, n_O]$, which must be processed individually, to do this we separate the points of the original point cloud according to these clusters, as in Figure 2.8.



(a) (b)

Figure 2.8. Point cloud (a) before clustering and (b) after clustering.

After this, we get the count of points ($n_{\text{pts},l}$) and cells ($n_{\text{cells},l}$) in each cluster, by summing all the points and cells that are clustered to that object. Then we perform the validation in Equation (2.7), in order to determine if a cluster is meaningful. To do this we stipulate that the rounded toward negative infinity base ten logarithm of the number of points

of a given cluster l, $n_{\text{pts},l}$, must be bigger than the corresponding rounded toward negative infinity base ten logarithm of the number of cells in that cluster l, $n_{\text{cells},l}$.

$$\lfloor log(n_{\text{pts},l}) \rfloor > \lfloor log(n_{\text{cells},l}) \rfloor \tag{2.7}$$

This allows for the removal of small clusters of useless points, more effectively, based on point cloud density and without the need for a denoise operation that could fail to produce this change but could damage the models.

Nevertheless, a verification procedure is implemented which requires the point cloud objects to have at least a pre-defined number of points, n_{v}, this parameter is heavily dependent on the quality of the point cloud and sensor, the average object distance from the camera to the objects and the average object size. It increases with the first and third and decreases with the second.

Even with this we encountered some problem since with objects that were close to each other, the clustering was not done properly due to the fact that noise and drag appeared in the point cloud, making a connection between more than one object and making them cluster as one.

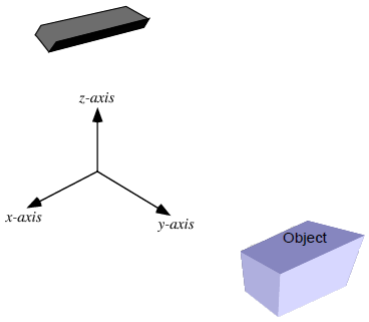


Figure 2.9. Axis explanation.

The numbering of the objects is done, as in Figure 2.7, relative to the camera with the axis in the form of Figure 2.9, in first instance from left to right, in increasing X-coordinate order and in second instance from near to far, in increasing Y-coordinate order.

2.2. Feature Extraction and Classification

After the segmentation, in order to lessen the time of calculations and to obtain features accurately, we execute a third or fine denoise, followed by a downsampling and a fourth or coarse denoise, as in Figure 2.10, to improve the stability and quality of the object's

representation, by removing erroneous data. The feature extraction is now performed, as in Algorithm 1, using the point cloud, $PC_{\mathrm{D},g}, g \in [1, n_{\mathrm{D}}]$, which is the point cloud after denoise and downsampling have been done to it. From now on, we will take a single object into consideration.

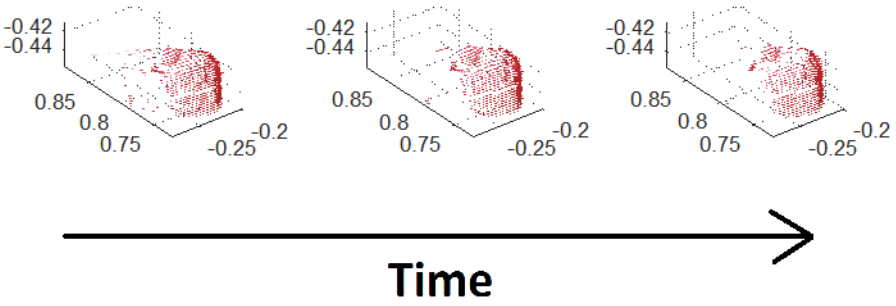


Figure 2.10. Evolution of denoise and downsampling of $PC_{\mathrm{O},l}$ into $PC_{D,g}$.

Algorithm 1: Feature extraction.
Input: *Point cloud object*
For each *Isolated Object*
% Colour extraction.
Colours = Extract_Colours *(Point_Cloud)*
For *m = 1:nruns*, $nruns \geq 0$
%Geometric shape fitter.
[Scores, Models] = Shape_Fitter *(Point_Cloud)*
End for
%Shape, score, volume and location extraction.
[Shape, Score, Volume, Location] = Shape_Volume_Extraction *(Scores, Models)*
End for
Output: *Colours, Shape, Score, Volume and Location*

2.2.1. Colour Extraction

We consider the colour to be important feature, as it allows for the differentiation between similar objects, for example: apples and oranges and pens of different colours. Each point, $\mathrm{p}_j, j \in [1, n_j]$, of the point cloud, $PC_{\mathrm{D},g}, g \in [1, n_{\mathrm{D}}]$, contributes with an RGB colour vector, $\boldsymbol{c}_{\mathrm{p}_j} = [r_{\mathrm{p}_j}, g_{\mathrm{p}_j}, b_{\mathrm{p}_j}]$, leading to a total of $n \times 3$ colours steaming from n points, making a matrix of size $n_j \times 3$.

If we consider only the colours of individual objects, these usually have distinct sets of colours, and as such, a clustering function seems appropriate to extract them. A clustering function aims at finding hidden groups or patterns of similar objects in a set of data, k-means clustering assumes that k, $k \in \mathbb{N}$, groups exist and tries to partition the data, so that each point will belong to the nearest "*mean*" or cluster. The algorithm picks k random points to initialize the procedure by designating them as "*means*". For the remaining points, the closest "*mean*" is found and the point will now belong to that cluster. After all points have been clustered, a new "*mean*" is computed for each cluster by taking in to account all the points of each cluster. The procedure is then repeated with the new "*means*" until no significant change is detected or an iteration limit is reached.

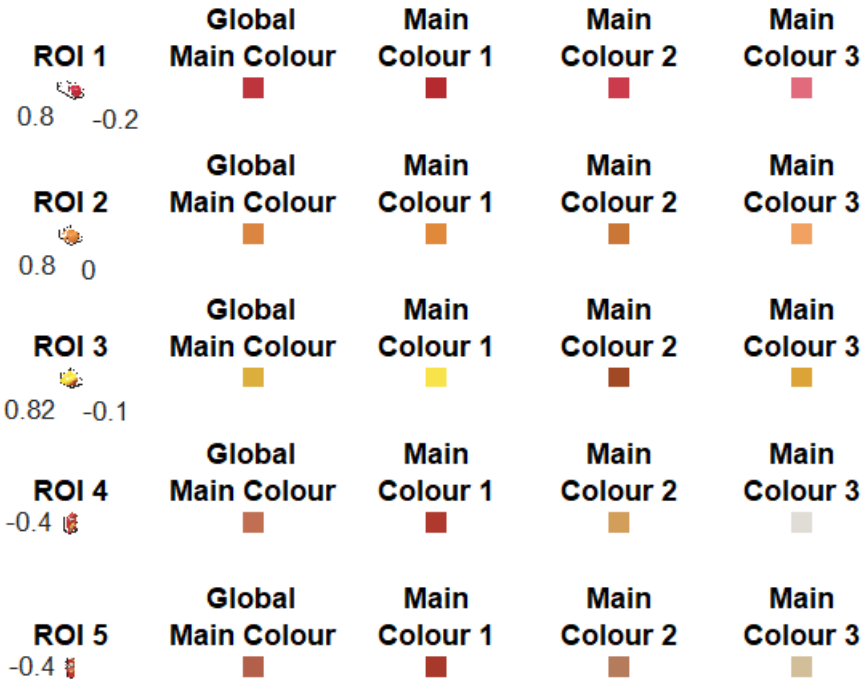


Figure 2.11. Result of colour extraction.

We extract a total of four colours, as we can see in Figure 2.11. The first colour is the global main colour obtained by assuming that there is only one dominant colour, one cluster in k-means clustering. The second, third and fourth extracted colours are three main colours. We obtain them by assuming the existence of three clusters that are sorted in descending order of importance. The rationale behind this choice, besides what was previously referred, is also deeply connected to the number primary colours of RGB colour model, 3 which are red, green and blue.

2.2.2. Shape Extraction

The geometric or shape features are extracted iteratively, as in Algorithm 2. Each point cloud object $PC_{\mathrm{D},g}, g \in [1, n_{\mathrm{D}}]$ is evaluated using three shape fitters: cylinder (1), sphere (2) and rectangular prism (3). Again, these models are fitted by means of RANSAC.

Due to the iterative nature of RANSAC algorithms, we initialize the algorithm $nruns$ times to improve reliability, reproducibility and choose the best possible fitting, the fitting that has the maximum score, $s_{max,g}, g \in [1, n_{\mathrm{D}}]$ for each point cloud object $PC_{\mathrm{D},g}$, as in Equation (2.8).

$$s_{max,g} = max(s_{i,g}) \tag{2.8}$$

The scores, $s_{i,g}, i \in [1, n_{\mathrm{fitters}}] \wedge g \in [1, n_{\mathrm{D}}]$, for each fitting of point cloud object $PC_{\mathrm{D},g}$, are evaluated by considering the number of inliers, $n_{\mathrm{inliers},i}$, for each fitting i, against the total number of points, $n_{\mathrm{points},g}$, of point cloud object g, as in Equation (2.9).

$$s_{i,g} = {}^{n_{\mathrm{inliers},i}}/_{n_{\mathrm{points},g}} \tag{2.9}$$

We also extract a volume, vol_g, and a maximum length along all axis, $maxlength_g$, of the point cloud object $PC_{\mathrm{D},g}, g \in [1, n_{\mathrm{D}}]$, which are used to evaluate the size relationship between the computed shape or fitted model and the point cloud $PC_{\mathrm{D},g}$, as shown in Equation (2.10) and Equation (2.11).

If one of the values of radius or volume of the model are more than maximum radius, $maxradius_g$ or maximum volume, $maxvol_g$, respectively, which are a function of maximum scale, $maxscale$, then those calculations are discarded. Either by giving them a score, $s_{i,g}$, of zero, in the case of the radius verification or by discarding the volume calculation for being inaccurate, because the size of model is too big when compared with the point cloud's, $PC_{\mathrm{D},g}$,size giving it the volume of a rectangular prism, containing the sides at minimum and maximum coordinates of the point cloud, in each Cartesian axis.

$$maxradius_g = \frac{(maxscale \times maxlength_g)}{2} \tag{2.10}$$

$$maxvol_g = maxscale \times vol_g \tag{2.11}$$

Also, if the maximum score, $s_{max,g}$, for a particular object g is equal or below a certain threshold, it will be attributed another shape which is no match shape (4).

Algorithm 2: Shape fitting and model extraction.

Input: *Point cloud object*

For each *Isolated Object*

  For each *Shape*

    Score(*Shape*) = 0

    For $m = 1{:}nruns, nruns \geq 0$

      %Geometric shape fitter.

      Trial_Model (*Shape*) = Shape_Fitter (*Shape*, *Point_Cloud*)

      compute *Trial_Score* (*Shape*)

      if *Trial_Score* (*Shape*) > *Score* (*Shape*) **then**

        Model (*Shape*) = *Trial_Model* (*Shape*)

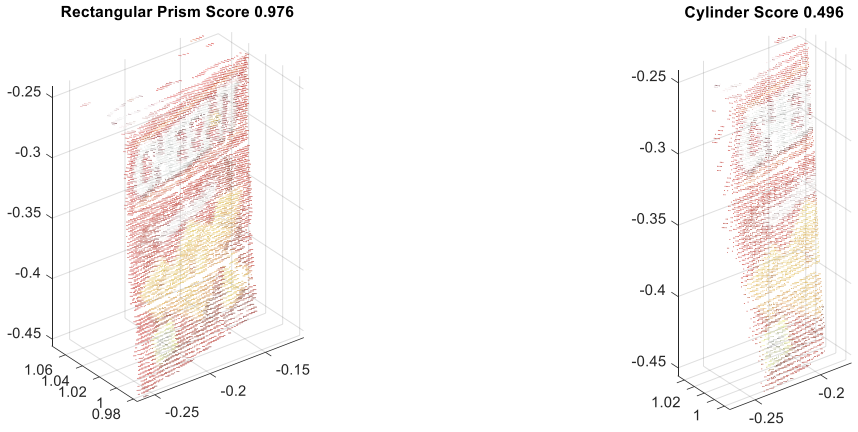
        Score (*Shape*) = *Trial_Score* (*Shape*)

      End if

    End for

End for

Output: *Scores and Models*



(a) (b)

Figure 2.12. Shape extraction results for (a) rectangular prism fitter and (b) cylinder fitter.

The cylinder fitter, shown in Figure 2.12 (b), assumes three reference vectors for fitting, one aligned with the X-axis, one aligned with the Z-axis and one that is the normal of a plane fitted to the point cloud, $PC_{\mathrm{D},g}$, using RANSAC. Then creates a cylinder fit for each of these reference vectors using MATLAB functions by means of RANSAC. In the case of the sphere fitter, there is no sense in using reference vectors since the sphere is always

the same independently of its orientation, but we also fit this shape using MATLAB functions by means of RANSAC.

The rectangular prism shape fitter, shown in Figure 2.12 (a), assumes the existence of two scenarios, one where the rectangular prism is aligned with camera, Figure 2.13 (a), here we extract the plane 1 by fitting a vertical plane, with normal aligned with Y-axis, and extract the top plane by fitting a horizontal plane, with normal aligned with Z-axis. The rest of the planes are obtained by creating planes that intersect the point cloud's minimum and maximum coordinates at each of the Cartesian axis and that have a normal aligned with the Cartesian axis.

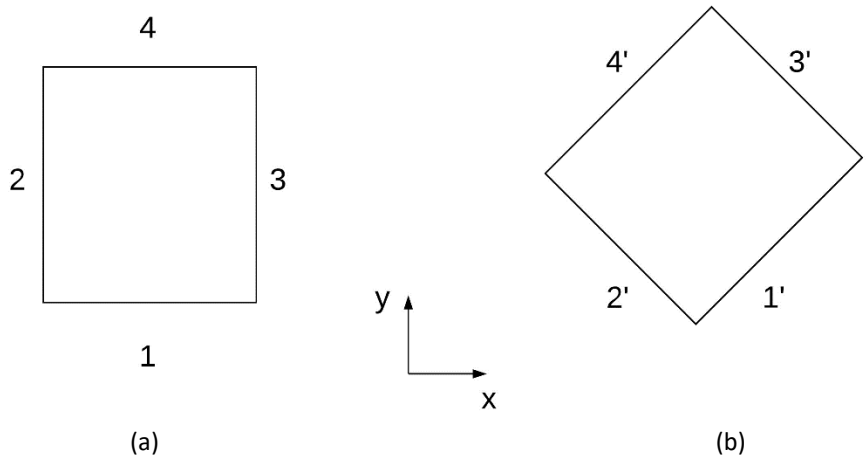


Figure 2.13. Rectangular prism shape fitting for (a) prism aligned with camera and for (b) prism in a random orientation with the camera.

The other scenario assumes that the rectangular prism is in a random orientation with the camera, Figure 2.13 (b), here there are three planes to be fitted: one horizontal, the top of the rectangular prism, and two intersecting vertical planes, planes 1' and 2'.

To get a normal for each plane to be fitted, we find the points that would be equivalent to the corners, the minimum and maximum coordinates along the X-axis and along the Y-axis. Combining this information, two vectors can be created that are parallel to the planes 1' and 2', by making them equal to the difference of coordinates, along the XY 2D space, between the point with minimum coordinate along the Y-axis and both points with the minimum and maximum coordinates along the X-axis. With this information, the normal for each plane, reference vector, can be computed, by getting vectors that are perpendicular to the previous ones.

The planes 1' and 2'can now be fitted, using the previously calculated reference vectors in a RANSAC algorithm. After this the top plane, can be fitted using a reference

vector along the Z- axis. The planes 3’ and 4’ are defined by creating vertical planes, that contain the points corresponding to the minimum and maximum coordinates along the X-axis and are parallel to the planes 1’ and 2’, only in the XY 2D space. In both cases the floor plane is created by making a horizontal plane, that intersects the minimum value of coordinates along Z-axis of the point cloud.

Another method tested, assumed that the planes 3’ and 4’ were defined by creating planes parallel in the 3D space to planes 1’ and 2’, with the rest being equal to what was referred. In both cases the results were similar, but slightly better considering the method of vertical planes 3’ and 4’ that ended up being used, as we can see in Appendix B, here method 1 refers to the vertical plane or parallel only in the XY 2D space method and method 2 refers to the parallel in 3D space plane method. The both methods were tested in random orientation towards the camera, considering the horizontal position, where the biggest side of the prim was against the top of the table and the vertical position, where the smallest side of the rectangular prism was against the top of the table, as in Figure 2.14 and keeping the testing setup between methods.

Figure 2.14. Testing setup in vertical position.

2.2.3. Dimension Extraction and Volume Estimation

With the shape primitives fitted, we now extract the dimensions and then estimate the volume. The dimensions themselves are obtained by direct extraction from the models, using their geometric properties in the cases of cylinders and sphere or by calculating the side lengths, knowing the corners by intersecting two planes, in the case of the rectangular prism.

In order to get the corners of the rectangular prism, we use the plane equation is in the form $ax + by + cz + d = 0$, considering the intersection at the Z-coordinate of the point cloud's centre, z_c, since this should be the average value of area of the rectangle containing the four side planes of the rectangular prism. Then we need to solve a system of equations, in order to get the point of intersection between two planes at that Z-coordinate z_c, for example Equation (2.12) is for generics planes 1 and 2.

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -d_1 - c_1 z_c \\ -d_2 - c_2 z_c \end{bmatrix} \tag{2.12}$$

After this, we calculate an approximation of the area of rectangle by finding the length of each of the sides of rectangle, along the XY 2D space at the Z-coordinate of the point cloud's centre and use difference of coordinates of two points, of two different planes to compute the height of the rectangular prism, one corresponding to the top of the prism, which is calculated by finding the coordinate along the Z-axis, of the plane top fitted at centre of the rectangle and the other corresponding to the bottom plane, which is perfectly horizontal, so we only need to obtain its Z-coordinate.

To finalize, we calculate volume using the geometric equations for each shape. If the model didn't respect the volume or score verifications, the volume of the object would be estimated by the size of a rectangular prism that contained the points that corresponded to the minimum and maximum coordinates in each of the axis, in other words the size of the point cloud.

In the test that were mentioned previously, in Appendix B, the object foam brick had a big difference in the value of its volume from the extracted to the real one, which was taken from the dimension values in [34]. That was due to the distortion of the image, by the inaccuracy of the Microsoft Kinect V2 sensor, as in [6] and the size of the trace behind the object was very significant compared to the size of the object, making the value of the volume very different from the real one.

Another method used assumed that the height of the rectangular prism was equal to the height at one of the corners, this gave us similar results to the ones in the previous method, since the bottom plane is horizontal and the top plane is almost horizontal, so we decided to use the first one.

2.2.4. Location Estimation

The location of the objects relative to the camera is calculated, in the cases of sphere and cylinder, by using the property centre of that model and in the cases of rectangular prism, no match shape and no volume verification, we calculate its location by getting the centroid of the model or point cloud, in other words we find the geometric centre of these rectangular prisms.

After this an offset is applied based on the calibration that is done, this step is detailed further along this dissertation, in the location accuracy and calibration subchapter. This location though is considered to be relative to the RGB camera, in Figure 2.15, not the depth sensor, since its exact position is unknown.

Figure 2.15. Scheme of the Microsoft Kinect V2 sensors.

2.2.5. Classification

To classify the objects using Algorithm 3, we use FFANNs, which we load in to our algorithm. Then we run each sample through these FFANNs and get a classification result. In order to run each sample through the FFANNs, we have to arrange the features before running it, so that they are in the form that the FFANNs were trained. Finally, we write the output of the classification, the data that the FFANNs used to classify the objects and their location after calibration, in to an Excel file.

Algorithm 3: Classification.

Input: *Colours, Shape, Volume and Score*

For each *Isolated Object*

Input (object)= Load (*features*)

End for

Load *neural_network*

For each *Isolated Object*

Classification (object) = neural_network (*Input (object)*)

End for

%Writing of results into Excel file.

Output: *Classification and Location*

3. EXPERIMENTS AND DISCUSSION

3.1. Setup

The point clouds in this dissertation were obtained using Microsoft Kinect V2. The Microsoft Kinect V2 is placed on top of a structure, which makes the centre of the RGB camera stand 46.5 cm above the table, looking down at the maximum angle allowed by the Microsoft Kinect V2 stand, between 27º and 30º and at a distance of between 55 cm and 105 cm along the Y-axis, to the object. This setup is in Figure 3.1 and was the one that yielded the best results according to several setups that we tested.

Figure 3.1. Microsoft Kinect V2 setup.

The Microsoft Kinect V2 is able to capture 1920x1080 RGB images and 512x424 depth images at 30 Hz and in the range of 0.5 m to 4.5 m.

The objects that were used are from the YCB object and model set [34].

We decided to set the following conditions as defaults of the algorithm:

- We use $n_{\text{pct}} = 10$, as a good equilibrium between number of points and computational time;
- The cell size in the merging operation was chosen as $s_{\text{c}} = 0.00001\ m$;
- In first trimming stage, we used a sphere as our trimming shape, the zero indexed constants, refer to centre of that sphere and are all equal to 0, since

the trimming is centred on the Microsoft Kinect V2, which is the origin of our referential and the radius (r) is equal to 1.5 m;

- We found that the values of $n_{\mathrm{n}} = 400$ and $n_{\mathrm{d}} = 1$, to be good in this case, at removing noise but not removing useful data in the first denoise operation;
- The iterative optimization of point removal and plane refitting is usually made 12 times, with cell of size of 0.01 m and the removal of points is done with $n_z = 1$;
- In the fine trimming stage, we used a rectangular prism as our trimming shape, with the dimensions of all sides equal to 0.5 m, in practice being a cube. Its centre was the centre of the point cloud after rotation with an offset of +0.15 m along the X-axis, to the right, in order to detect which is the area of the table in which objects are;
- We used $bin_{offset} = 4$ and $binwidth = 0.003\,m$, since these values provide a good table removal;
- We found that the values of $n_{\mathrm{n}} = 4$ and $n_{\mathrm{d}} = 1$, to be good in the case of second denoise operation;
- In the clustering stage, the grid size was 0.01 m, as this provided good enough clustering;
- In the validation of clustering, we used $n_{\mathrm{v}} = 500$, because it was a good value for these household objects;
- We used the values of $n_{\mathrm{n}} = 20$ and $n_{\mathrm{d}} = 1$, as they were good in this case, at removing noise but not removing useful data in the third denoise operation;
- The cell size in the downsampling operation was $s_{\mathrm{c}} = 0.001\,m$;
- The values of $n_{\mathrm{n}} = 10$ and $n_{\mathrm{d}} = 1$ were used, as they were good in the case of the fourth denoise operation;
- We chose to use $nruns = 10$ and maximum number of random trials in the fitting equal to 1000, since this was enough to ensure the convergence of the scores;
- We decided to use $maxscale = 1$, since this gave us good results in eliminating erroneous models;

- The inlier tolerances for shape fitters were 0.01 m for cylinder fitter, 0.005 m for plane fitter, 0.005 m for sphere fitter and a 10º for angular tolerance in the cylinder and plane fitters;
- The threshold for no match shape (4) is $s_{max,g} \leq 0.5$.

These numbers were reached by performing several trials, in our setup and evaluating the impact on the next stages.

3.2. Cataloguing

The cataloguing was done considering 20 objects from the YCB object and model set [34], which are in the Table 3.1 and Figure 3.2.

Table 3.1. Cataloguing objects and their corresponding number.

Object	Number
Sugar box	1
Potted meat can	2
Cracker box	3
Wood block	4
Gelatine box	5
Pitcher	6
Mug	7
Master chef can	8
Tomato soup can	9
Chips can	10
Apple	11
Peach	12
Plum	13
Lemon	14
Mini soccer ball	15
Orange	16
Softball	17
Tennis ball	18
Baseball	19
Bowl	20

Figure 3.2. Cataloguing objects.

The conditions were set to the defaults of the software that was developed and explained before.

The objects were catalogued with the orientations in Figure 3.3 and in Figure 3.5, some were catalogued in two different orientations and others were catalogued in four different orientations.

The objects that closely resemble a sphere were catalogued in two different orientations, one with the logo facing the camera, if applicable, being the front orientation and one with no logo facing the camera, being the back orientation. The objects that closely resemble a cylinder were catalogued in two different orientations, vertical with both the front and back facing the camera.

The objects that closely resemble a rectangular prism were catalogued in four different orientations, in the vertical orientation, with both sides, front and back, of the object facing the camera and with the objects biggest side aligned with the camera or in a random orientation. Both random orientations corresponded to a counter clockwise rotation from the aligned orientation.

Of this we exempt the bowl, due to reasons that will be explained in the cataloguing results subchapter and the wood block, since in this object the front and back are the same, making only sense to catalogue in both the vertical and horizontal orientations and in only the random and aligned orientations, as in Figure 3.3.

Figure 3.3. Explanation of orientations.

With these orientations, we catalogued the objects in five different positions, as in Figure 3.4, the first position is in the centre of the workspace, a square that has the position referred in the setup, the next four are closer to each of the corners of the workspace, from the camera position 2 is near the near left corner, position 3 is near the near right corner, position 4 is near the far left corner and position 5 is near the far right corner.

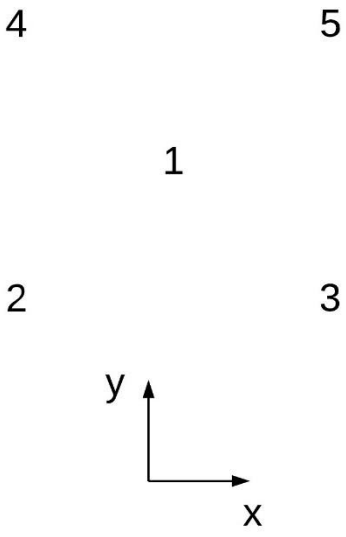


Figure 3.4. Explanation of positions.

The fitting tolerances were 0.005 m for sphere and plane fitting, 0.01 m for cylinder fitting, due to the fact that by our empirical testing the cylinders fitted worse than the other shapes, so we had to favour them by giving them a bigger tolerance and 10º for angular tolerance in all fittings. Also, we set $maxscale = 1$, these values were achieved by empirical testing of several objects and were found to be the ones that gave the algorithm the best performance.

We take the following features from Microsoft Kinect V2 and the algorithm for cataloguing:

- Shape;
- Score;
- Measured volume;
- Global main colour;
- Three main colours;
- Location.

The next features had to be provided by us and were taken from page 43 of article [34], except in the cases of the pitcher and bowl, since in both cases the diameter of base and top, where very different, we assumed the diameter to be the average of this two values, which we measured, and used to calculate the real volume. These features are:

- Object;
- Real volume.

One important aspect of this work, is that samples for both training and testing were taken in a unstructured environment, as oppose to the ones in the datasets [7], [34], making the variability of these be much higher than the works referred in the state of the art, and making the detection of objects more difficult.

The cataloguing means and standard deviations for each feature of each object, except for the location and real volume, are shown in Appendix C and in Appendix D, respectively.

Note that even though the volume in the appendices is always in cubic centimetres, in the algorithm we used cubic meters, since for the algorithm it's the same to use either one.

3.2.1. Cataloguing Results

Depending on the tolerances used, objects can have similar scores for very different geometries, for example: a cylinder with a small base can be reasonably approximated to a rectangular prism, apples, are slightly flattened at the poles which increase the score of the cylinder and rectangular prism fitters.

On the other hand, some volume estimations can be very different from the real volume since a drag is always formed behind the object, in an unpredictable way, making the hidden part of the object appear much further back than it actually is and the volume bigger than real one. This drag could also lower the score of an object, because the points of the drag would not appear in the fitted result leading to for example a wrong shape detection.

If the objects are small or they have low height the Microsoft Kinect V2 distortion of the image and the limited precision of the segmentation can make a big error appear in the volume or eliminate the object.

With this in mind, this subchapter is dedicated to discussing the unexpected results or acquisition errors in the cataloguing process. To be considered an error, the sample must have at least one of the following:

- Wrong best fit;
- Score lower than 0.9 of the best fit;
- Measured volume that is below 20% of the real volume or that is 10 times bigger than the real volume.

We exempted the mini soccer ball, of the score being below 0.9, since it isn't a perfect sphere, by being composed of many flat surfaces, it's expected that its score on the sphere shape is below 0.9.

In the cases of the potted meat can, master chef can and tomato soup can, Microsoft Kinect V2 had problems acquiring reflective surfaces [6] and thus distorted them, making the score, shape and volume estimation have a big error. This was more apparent under artificial lighting, where the surfaces were more reflective, because of the angle of incidence of the light, made them reflect more towards the camera. This also happened in the top surface of the chips can, which had a lid made of a transparent and reflective plastic, but in here the effect were much lower only making the score of the best fit low, when compared with instances of no reflection.

Also, the potted meat can had round corners making the score low when compared with the other objects with rectangular prism shape, even making the fitting better as a cylinder in some instances because of this.

In some cases, the algorithm would get the best score for the rectangular prism as a random one even though it was aligned in reality, only picking up the part of the object facing the camera and making a wrong volume estimation, lower than the real one. This was the case in some samples of the cracker box, gelatine box.

Also, in some cases, namely in the wood block, the score for an aligned rectangular prism was low, since using our fitting function for aligned prism, the points on the sides don't count for inliers, but count negatively towards the score, making the score low. This also explains the fact that sometimes, the best score would be of the rectangular prism as a random one, even though the object was aligned in reality, since for the aligned prism fitter the points corresponding to the sides do not count as inliers, but in the case of the random prism fitter, the points of one of the sides count as inliers.

The gelatine box in some instances had low score, because Microsoft Kinect V2 distorted the top by making it have a large drag with an inclination equal the inclination of Microsoft Kinect V2, between 27º and 30º from the horizontal, making very few points of the top plane count as inliers, because the angular tolerance for fitting was only 10º.

The pitcher and mug got low scores, in some samples, because of the handles not being considered in the fitted result as inliers, but counting negatively, towards the score, in the total points. Also, the pitcher was not a perfect cylinder since the top had a bigger diameter than the bottom, contributing more towards this.

In the cases of the chips can and master chef can there were some instances where a large drag was formed behind the object, making the score of the cylinder shape low, since most of the points of the drag did not count as inliers. This made the best fit be a rectangular prism, that counted more of those points as inliers.

In the cases of the orange, apple, plum and peach, the best fitting was always expected to be a sphere and in the case of the lemon, it was expected to be a cylinder, but instead, sometimes it was something else or had a low score. In the cases of the orange and apple, this was due to the fact that they were not perfect spheres, having the top and/or bottom as flat surfaces. In the cases the peach, lemon and plum, this was due to the fact that

they were small objects and the tolerances for inlier points, in fitting, were big when compared with the overall dimensions of the object.

In both cases counting almost the same points of the object in all fitting shapes, making all the shape scores very close, thus not having a very stable fitting shape and getting different shapes on different trials with the same object. To contribute to this, none of these objects were perfect primitive shapes. To correct this, we tried to put bigger tolerances on the sphere fitter, but this made all small objects appear as spheres, so we decided to keep the tolerances as they were, since we gained precision in some samples only, at the expense of others.

It is also important to note that since the fitting was affected, the volume estimation also was affected, as was the case in some instances of the lemon, where the volume estimation was much lower than the real volume. Contributing also to this was the fact that the plum, peach and lemon where small objects and that a significant part of the object was removed, during the table removal stage of the algorithm, due to its limited precision. So, in the case of the lemon, orange, apple, plum and peach, we assumed the nomenclature, catalogued and tested them all, as if they were cylinders, in all trials.

We also tried the bowl, the best shape that we expect was a cylinder, but because the sides are curved, it gave us a best fit of a sphere, in most cases, with a relatively low score of around 0.8 in all cases, including the ones that gave us the cylinder shape. With this in mind and taking into account that by rotating the bowl, it is still perceived in the same way, we decided to catalogue the bowl in two orientations, the top up orientation in Figure 3.5 (a) and the bottom up orientation in Figure 3.5 (b).

(a) (b)

Figure 3.5. Bowl in the (a) top up and (b) bottom up orientations.

The shape error, made the volume and location estimations have a big error, although this error was consistent in all testing, making use of the volume verification that was previously referred, the error was lessened in the volume and location estimations. This show the importance that the colour has in classification, due to the inaccuracy of the shape and volume estimation and the importance of the volume verification.

One final source of defects was that some objects had undergone some damage along the transportation before reaching us, making this also an origin of errors for our algorithm. This was clear in the case of the sugar box, where there were some instances, that the object was detected with a low score, due to the fact that some points of the damaged object, did not go as in the inlier points of the rectangular prism shape fitter, as it would happen if the object was not damaged.

Overall the errors were due to the fact the Microsoft Kinect V2 had low accuracy and distorted the objects as input to this algorithm, making every estimation have an error, especially the volume estimation. Also, the fact that we only used one camera, made us only see one side of the object, meaning that the data that we had was very few, making us have to estimate the hidden parts of the object, which was not precise and gave us a great error.

3.3. Classification

After the cataloguing and since the number of samples is very low for training artificial neural networks (ANNs), in order to make the training faster, we created algorithm that creates virtual samples, based on the real ones that we took during the cataloguing process, by creating virtual positions with the same number of orientations as in cataloguing.

This algorithm works by first defining the number of virtual positions that we want, and loading each of the targets and samples that we took. After this, we organize the samples according to each object and get for each feature of each object a mean and a standard deviation, based on the samples that we took during the cataloguing process. We create each virtual sample, by creating random values for each feature of each object, from the normal distributions with the mean and standard deviation calculated previously.

After this and due to the large number of virtual samples that is created, it is possible that some values are outside the real ranges that those features have, for example negative values of volume and shapes that are not positive integers between 1 and 4.

The range of shapes is a positive integer between 1 and 4, so we round the value of random number generated and make that if a value is smaller than 1, it should be equal to 1 and if it's bigger than 4, it should be 4 and have its score equal to a random uniformly distributed number between 0 and 0.5, to be consistent with the previously set score verification. Also, the score range will have a similar verification, since every score that is equal or below 0.5 will be attributed the no match shape (4), and have its score equal to a random uniformly distributed number between 0 and 0.5 and if the score is bigger than 1 it should be equal to 1.

For the case of the range of volume, we say that if a volume is less or equal than 0, then the value of volume would be equal to the absolute value, of that random number that was generated. In the case of the colour, since the values from Microsoft Kinect V2 range from 0 to 255, we considered every value that is less or equal than zero to be equal to zero and would correspond to a value of colour that is so low, that Microsoft Kinect V2 would pick it up as 0 and every value larger than 255, would correspond to a value that is so big, that Microsoft Kinect V2 would pick it up as maximum value or in other words 255.

This is a problem of pattern recognition, which has 15 inputs, that are shape, score, measured volume and the four colours, since each colour has 3 values corresponding to the values of red, green and blue, this explains the number of inputs. We have 20 outputs, since this is the number of target classes.

To solve this problem, we decided to use FFANNs which are one of the simplest types of ANNs and tested various conditions of training, in order to estimate the optimal conditions for training.

We started by finding the best number of virtual positions for training, we trained five times each with 10, 100, 1000, 10000 and 100000 virtual positions. We did this testing with 10 neurons in the hidden layer and in MATLAB's default conditions of random distribution of data of 70% for training, 15% for validation and 15% for testing. After this, we took the average fraction of misclassified samples of five tests, at each number of virtual positions and got Table 3.2 and Figure 3.6.

In the following tests of this subchapter, we used as testing data the real samples that we took in cataloguing and changed the training data as a new training was done.

Table 3.2. Fraction of misclassified samples by number of virtual positions.

	Fraction of misclassified samples [%]					
Number of virtual positions	**Training 1**	**Training 2**	**Training 3**	**Training 4**	**Training 5**	**Average**
10	12,4	13,6	16,8	15,2	15,6	14,7
100	7,2	8,4	8,8	7,2	6	7,5
1000	4,4	6,8	4,8	8,4	6	6,1
10000	4,4	5,2	4,4	4,8	6,4	5,0
100000	4,8	5,2	5,6	5,6	5,6	5,4

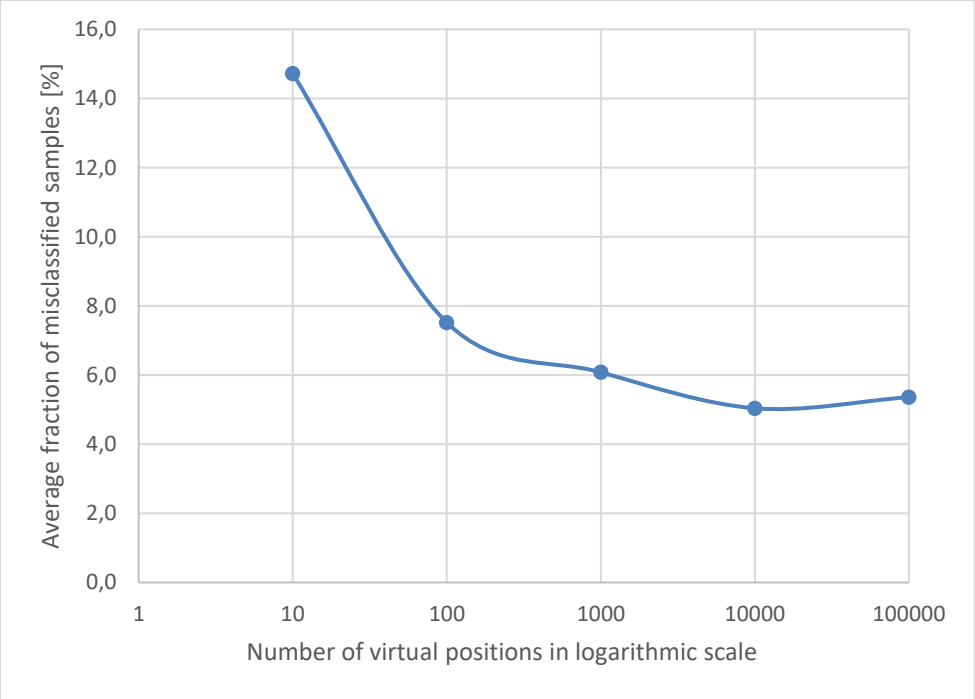


Figure 3.6. Fraction of misclassified samples by number of virtual positions.

Since the minimum average fraction of misclassified samples corresponded to 10000 virtual positions, we used that value in the following tests, in order to train the FFANNs and chose the first three tests, of the 10000 virtual positions, to use in the Table 3.3 and Figure 3.7 as comparison, as they were the 10 hidden neurons case. But as a note we also got good results with 1000 virtual positions in training.

Now we will change the number of hidden neurons, between 5, 10, 15, 20, 25, 30, 40, 50, 75 and 100, to see what is the effect of this change. In this and the next test

though, we are only to average the fraction of misclassified samples of three tests, since the accuracy of the algorithm is already good. Keeping the same data division as before for training, validation and testing. We got the following results.

Table 3.3. Fraction of misclassified samples by number of hidden neurons.

	Fraction of misclassified samples [%]			
Number of hidden neurons	**Training 1**	**Training 2**	**Training 3**	**Average**
5	8	4,4	8	6,8
10	4,4	5,2	4,4	4,7
15	5,2	4	4,8	4,7
20	3,2	4,8	5,2	4,4
25	4	2,8	4,4	3,7
30	3,2	5,2	5,6	4,7
40	4,8	4,8	4	4,5
50	3,6	3,6	4	3,7
75	2,8	4	2,8	3,2
100	5,6	4,4	4,4	4,8

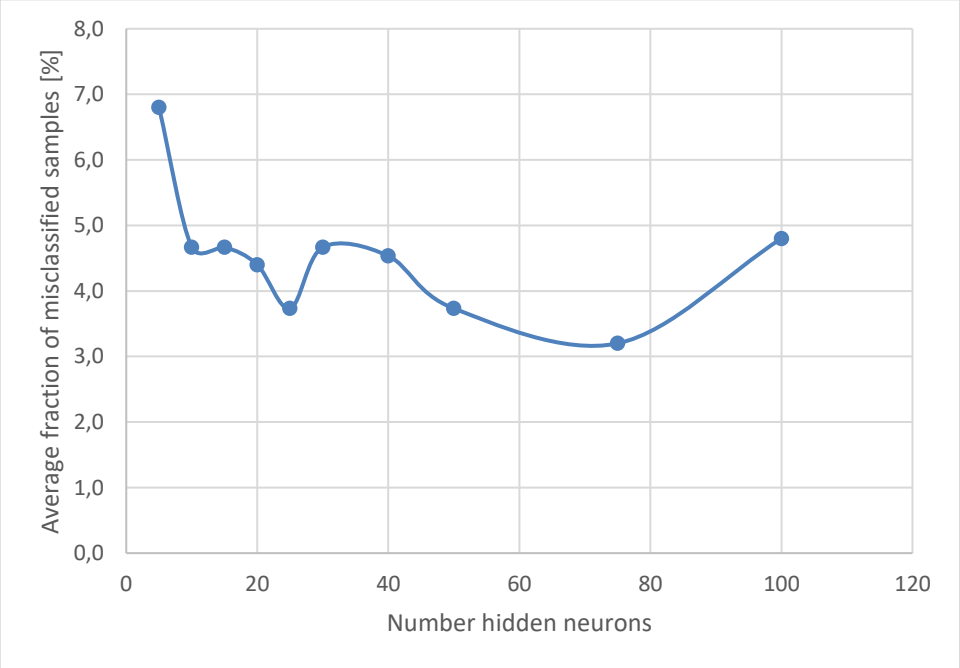


Figure 3.7. Fraction of misclassified samples by number of hidden neurons.

Considering that one of the minimum fraction of misclassified samples was in a training using FFANNs of 25 hidden neurons and that after that there was little to gain in adding more neurons, since we couldn't get below the 2.8% fraction of misclassified

samples. We decided to used 25 neurons in the hidden layer and change the division of data. The validation and testing data can be 5%, 10%, 15%, 20% and 25% of the total data, with them being equal in all the tests. We used the three tests of 25 neurons that we previously got as comparison, as they were the 15% case and got Table 3.4 and Figure 3.8.

Table 3.4. Fraction of misclassified samples by data division.

	Fraction of misclassified samples [%]			
Data division (validation and testing) [%]	**Training 1**	**Training 2**	**Training 3**	**Average**
5	2,8	4,8	4	3,9
10	5,6	4	4	4,5
15	4	2,8	4,4	3,7
20	5,6	4,4	3,6	4,5
25	3,2	4	5,6	4,3

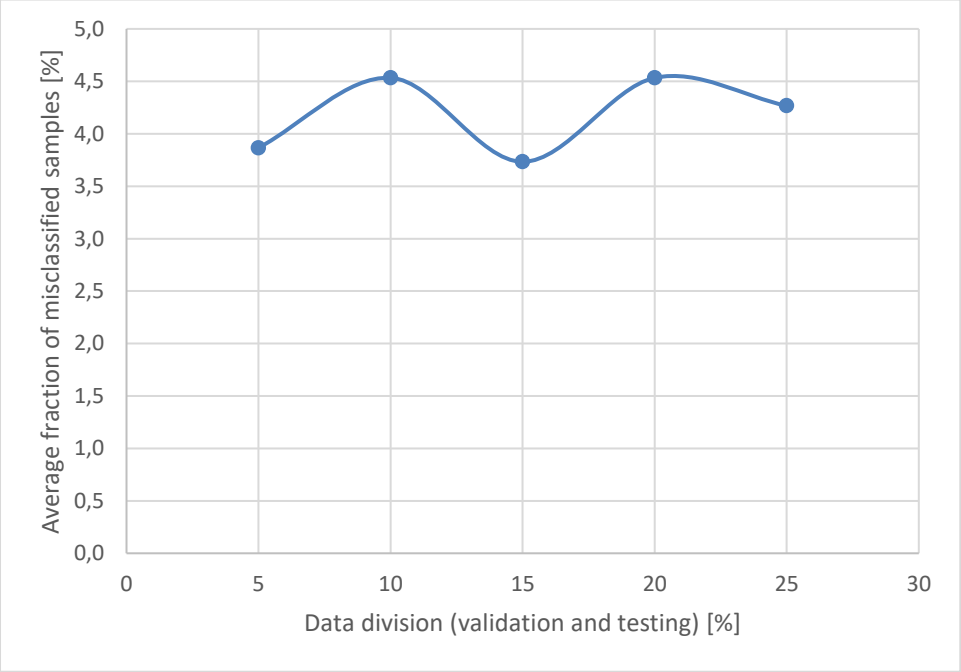


Figure 3.8. Fraction of misclassified samples by data division.

The results show that changing the data divisions had little effect on the classification, in which the best results were achieved with training containing 10000 virtual positions (500000 samples), 25 hidden neurons and the divisions set by MATLAB's default conditions of random distribution of data of 70% for training, 15% for validation and 15% for testing.

The confusion matrices in Appendix E, are the ones that we got in testing with the best training conditions and are in the previous tables.

Ignoring instances of confusion, that happened in only one of the tests in Appendix E, we can see that the FFANNs confused tomato soup can and the chips can 5 times, the cracker box and the gelatine box 5 times, the gelatine box and the chips can 2 times, the sugar box and the wood block 2 times, the sugar box and gelatine box 5 times, the potted meat can and master chef can 2 times and the cracker box with the chips can 5 times.

Based on the means of the features of each object in Appendix C, we see that the confusions can be explained as follows.

In the case of the tomato soup can and the chips can, they were confused because they have the same shape, similar scores and similar colours being the red and the white. Since the volumes are in the same order of magnitude and since the volume estimation has a big error, the confusion happened.

The confusion between the cracker box and the gelatine box was due to the fact that these two objects had the same shape, almost the same score and similar colours, being the red and the white, also as we saw in the cataloguing, there were some instances where volume estimation had a big error in these objects, being this the possible source of the confusion, since this was one of the few features that distinguished them.

In the case of the gelatine box and the chips can, the confusion is understandable since here score and colours are similar, they have both the red and white colours, also there were some instances in the cataloguing, where the chips can was detected as a rectangular prim, as said previously, making the shape the same as the gelatine box.

Considering now the case of the confusion between the sugar box and the wood block, we can see that they have the same shape, similar scores and similar colours, being these between the white and yellow colours. Since the volumes are same order of magnitude and the volume estimation has a big error, the confusion happened.

Seeing now the case of the confusion between the sugar box and gelatine box, they have the same shape, similar scores and colour, this being the white colour. Due to the inaccuracy of the volume estimation an error happened.

In the case of the potted meat can and master chef can, they have similar shapes, since in some instances the master chef can has the rectangular prism shape and in some instance of the potted meat can has the cylinder shape, the scores are similar and they have

similar colours, being the blue and orange. Once more since the volumes are in the same order of magnitude and since the volume estimation has a big error, the confusion happened.

In the case of the confusion between the cracker box with the chips can, this was due to the score and all the colours being similar, the colours being the red, white and orange, plus, as said before, in some instances the chips can was detected has a rectangular prism and the cracker box had a big error in volume, making the confusion understandable.

To try to solve these confusion errors, we tried to change the training by changing number of samples, the number of hidden neurons and the random distribution of data between training, validation and testing, but this yielded no improvement in overall results since we made these confusions disappear at the expense of creating new ones. So, we decided to keep the best training.

Overall these tests showed the need to improve the volume estimation of the algorithm, and how important the volume is as a feature. The biggest issue that was encountered during the volume estimation, was that since we only use one RGB camera in a stationary position, with one single view of the objects, we only see the part of the objects facing the camera, meaning that the hidden parts of the objects need to be estimated, making an inaccurate volume estimation, in many occasions.

3.4. Training

Since as we saw earlier, the best results were achieved with training containing 10000 virtual positions (500000 samples), 25 hidden neurons and the divisions set by MATLAB's default conditions of random distribution of data of 70% for training, 15% for validation and 15% for testing, as previously explained. We chose the best FFANNs that we trained, to implement into our algorithm, by selecting the one that had the least fraction of misclassified samples, in testing with the real samples. We got following the results with these FFANNs, in Table 3.5 and Figure 3.9.

Table 3.5. Fraction of misclassified samples according to type of data.

Data	Fraction of misclassified samples [%]
Training	0,186
Validation	0,219
Testing	0,228

To achieve this result we needed 215 iterations, it took us 5 minutes and 43 seconds to train in a 4th generation Intel Core i7 processor.

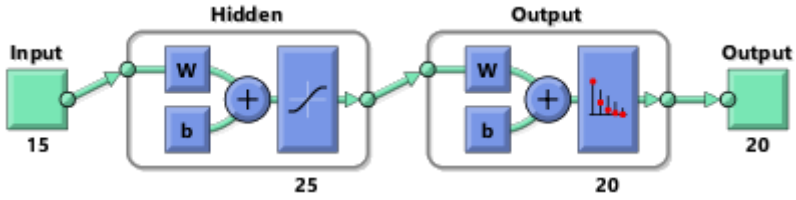


Figure 3.9. Schematic representation of used FFANNs.

We use two-layer FFANNs, with 25 sigmoid neurons in the hidden layer and 20 softmax output neurons, since this is the number of target classes.

In our testing, in Appendix F, we tested the FFANNs, as we did before, with the real data from cataloguing, which we used to create the virtual positions and got a fraction of misclassified samples of 2.8%.

In this test, the algorithm confused the cracker box and the gelatine box 2 times, the gelatine box and the chips can 1 time, the sugar box and gelatine box 2 times, the potted meat can and master chef can 1 time and the cracker box with the chips can 1 time, with all the reasons for these confusions being explained in the previous subchapter.

3.5. Testing

After this, we tested the algorithm with four object sets, in Table 3.6, containing five objects each. We did five tests per object set, randomizing the pose of the objects between tests, only paying attention that bigger objects can not cover small objects behind them, allowing for Microsoft Kinect V2 to see them all, as in Figure 3.10 (a) and opposed to what happens in Figure 3.10 (b).

(a) (b)

Figure 3.10. Master chef can (a) not covering and (b) covering the baseball.

Table 3.6. Testing sets.

Sets	Object 1	Object 2	Object 3	Object 4	Object 5
1	Gelatine box	Tomato soup can	Potted meat can	Mini soccer ball	Bowl
2	Wood block	Sugar box	Peach	Plum	Lemon
3	Master chef can	Pitcher	Softball	Tennis ball	Orange
4	Chips can	Cracker box	Apple	Baseball	Mug

There were some instances where we accidentally put two objects close to each other and that made the clustering process not preform correctly, these instances were discarded but the results showed the need to improve how the algorithm handles objects not in our dataset and the clustering process. This improvement can be done by maybe putting a smaller grid size in this stage of the algorithm. Also, shown that testing with multiple objects is more difficult than using one single object, since the presence of multiple objects on the scene close to each other, interferes with the objects in the scene causing confusion.

The results are in Table 3.7 and Figure 3.11.

Table 3.7. Recognition results in testing.

	Recognition accuracy				
Sets	**Object 1**	**Object 2**	**Object 3**	**Object 4**	**Object 5**
1	5/5	4/5	4/5	5/5	5/5
2	4/5	5/5	5/5	5/5	5/5
3	5/5	5/5	5/5	5/5	5/5
4	5/5	4/5	5/5	5/5	5/5

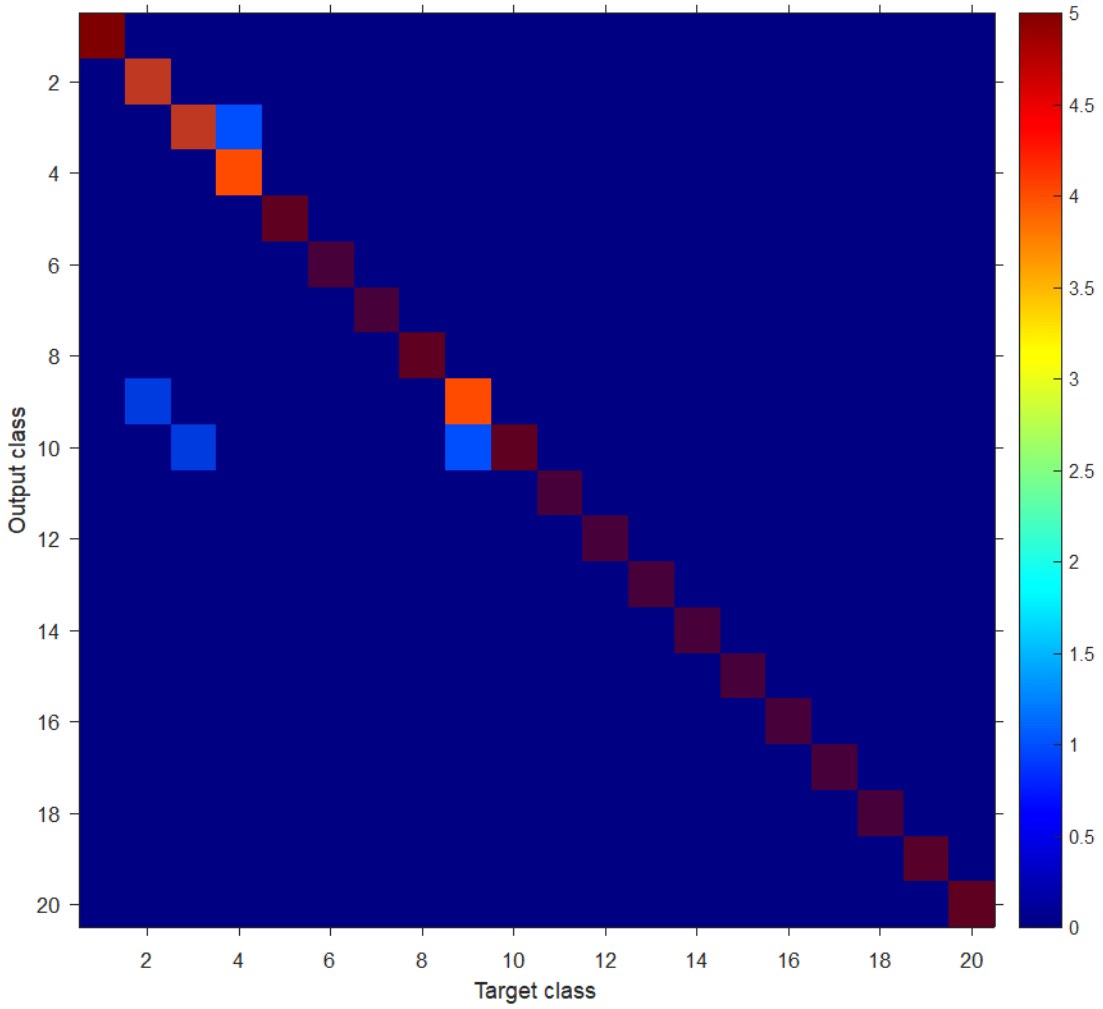


Figure 3.11. Confusion matrix of multiple object testing.

In this test, there were some instances of misclassification making the overall accuracy in this test equal to 96%, which is in line with state of the art techniques, that are in Table 1.1. To add to this, we have to take into account the fact that, we are testing with multiple objects in the workspace or scene in an unstructured environment, instead of just one in a structured environment, as they did in most papers of Table 1.1.

In Figure 3.11, the algorithm confused the potted meat can with the tomato soup can, this was due to the fact, that in one instance the potted meat can had the cylinder shape as best fit and a measured volume very similar to the average volume of the tomato soup can, in cataloguing. Plus, Microsoft Kinect V2 distorted the potted meat can making the score low and causing the confusion.

The algorithm confused the tomato soup can with the chips can and the cracker box with the chips can, due to reasons that were explained before and plus in this case the cracker box was very near the side orientation, which was an orientation not in our catalogue and showed the need to take more samples.

Also, we got the confusion of the wood block with the cracker box, this was due to the fact that in Appendix C, they had the same shape, similar scores, similar volumes and some of the colours were similar, namely the third main colour and the reds in all the RGB colour vectors. Plus, in this case there were even more colours that were similar to the ones of the cracker box, namely the blues in all the RGB colour vectors.

Since this field is new, we were only able to find method [35] using the YCB object and model set [34]. This method, achieved a perfect classification in single object instance recognition, using the same object set [34], as we did. But only classified 11 objects, with only 8 from the object set in [34], whereas our algorithm classifies 20 objects from the same object set [34], also our algorithm required much less real data for training, only 250 total samples, an average of 13 samples per class, whereas the algorithm in [35] required 1744 total samples, an average of 159 samples per class. Making the cataloguing for our algorithm faster, since we need less real samples.

Also, in our tests with multiple objects, we tested with five objects instead of three, as in [35], making the classification more difficult, since here we have more objects to interfere with the segmentation and classification algorithms. Still we achieved an accuracy of 96%, whereas the method in [35] achieved an accuracy of 100%, with only the objects in its training set.

Since our algorithm got an accuracy of 96% with multiple objects on the scene and in an unstructured environment, we can say that algorithm has good accuracy when compared with similar algorithms, as seen in the Table 1.1 and in the related work [35].

3.6. Location Accuracy and Calibration

To estimate the location, we use the centroid of the best model that was fitted. In order to calibrate Microsoft Kinect V2, we need to choose an object that it identifies well. Based on the means and standard deviations, in Appendix C and in Appendix D, and the real values of the features, we chose the tennis ball as our calibration object, since the measured

volume and real volume were very close, the measured volume had a small standard deviation and the best shape was always a sphere.

The tests were performed considering five positions, just as we did before in Figure 3.4, we took the real coordinates of those locations and then compared them with the ones that we took from the algorithm, five times, in order to calibrate our device.

With the average distance between the real and measured positions along each of the Cartesian axis, we can create an offset to put in our algorithm, in order to give a more precise estimation and to have an idea of the error, that our algorithm has in the location estimation.

The positions were as described in the cataloguing section, in Figure 3.4 and were chosen, in order to get an idea of what the error was in the different parts of our workspace. The location error was due to [6]:

- The sensor by inadequate calibration;
- The measurement setup by the lighting conditions and image shape;
- The properties of object surface as in reflective surfaces.

The average differences of the real location minus the measured location and the average distance from the real location to the measure location for each position are in Table 3.8.

Table 3.8. Location test and calibration results.

Position	Average difference of coordinate X-axis [cm]	Average difference of coordinate Y-axis [cm]	Average difference of coordinate Z-axis [cm]	Average distance [cm]
1	-5,22	-1,36	-1,01	5,50
2	-4,53	-1,94	-0,87	5,01
3	-4,81	-2,03	-0,67	5,27
4	-4,69	-1,81	-1,12	5,15
5	-5,04	-1,70	-1,26	5,47

After this, we created an algorithm that divides the workspace into four equal spaces, along the XY 2D space and creates an offset to the location in each space, based on the average difference of coordinates between the real and measured positions, from positions 2 to 5. This algorithm is implemented after the location is obtained. Also, the first position was only to get an idea of what was the error was in the centre of the workspace.

We got a minimum distance of 4.78 cm and a maximum distance of 5.75 cm between the real and the measured positions, these values are useful to give an idea to the robot where this object is. From Table 3.8, we can see that the biggest difference is in the X-coordinate, this difference is on average -4.86 cm and is due to the fact that the location is relative to the depth sensor not the RGB camera, which is about 5 cm to the right of the RGB camera, while looking at the front of Microsoft Kinect V2.

By taking this average error in the X-axis and offsetting the location by it, we can see that the minimum distance of 1.46 cm and a maximum distance of 2.62 cm. In line with the precision of Microsoft Kinect V2 that is in [6]. Although we know that it is not like this, in the algorithm, we decided to use the location relative to the RGB camera, since this is a well-known position, making it is easy to implement in any robotics application, as opposed to the exact location of the depth sensor which is unknown.

As a note, since with aligned rectangular prisms, the fitting can be only the part of an object facing the camera and not much more this can lead the location to have a large error.

4. CONCLUSIONS

The objective of this dissertation was to present a novel method of object classification using RGB-D data from Microsoft Kinect V2, that was able to classify 20 objects from the YCB object and model set [34]. In doing so, a novel unsupervised feature extraction algorithm from RGB-D data was also introduced, which was used together with FFANNs for object classification.

The unsupervised feature extraction algorithm works by first acquiring and merging frames from Microsoft Kinect V2. After this a first trimming procedure is done, that removes useless data, beyond the reach of the manipulator, for example a robot. Then rotations are applied to the point cloud so that, Microsoft Kinect V2 is able to compensate when its pose is changed and in order to have the table in the horizontal position. Following this, a fine trimming operation and a table plane removal are applied in order separate the objects from the background.

Then object segmentation is applied to separate each object into its own points cloud, with the objects separated denoise and downsampling are applied, in order to give more stable data to the feature extraction algorithm. This algorithm works by extracting the main colours of an object, the primitive shapes and score that best fits an object and an estimation of the volume based on that fitting.

In the classification stage of the algorithm, FFANNs are used to classify the objects based on these inputs. Results show, that with the introduction of virtual samples the accuracy of classification can be greatly improved, without the need to take a large number of real samples, which is a very time-consuming task. Also in this dissertation, it is shown the importance of careful study of the best conditions of training, taking into account that the number of samples in training and the number of hidden neurons in the FFANNs, can have a great impact on the accuracy of classification.

With the tests that were presented in this dissertation, we can see that an accuracy of 96% can be achieved with multiple objects on the scene and in an unstructured environment, which is better than most state of the art methods which achieve similar accuracies but with a single object on the scene and in a structured environment.

It is demonstrated that by doing the training with features instead of raw RGB-D data, cataloguing was much faster, since this allows for the simple creation of virtual samples, based on means and standard deviations from cataloguing, to be used for training, which will have more samples and will lead to a higher accuracy of recognition.

Also with this dissertation, a method of locating objects using RGB-D data was introduced, this method considers the location of an object to be the geometric centre of the model, that was created during the fitting stage of the algorithm. It is shown that maximum error of this method is 5.75 cm and that this error can be lessened with careful calibration.

Overall the results indicate that, we can extract features of objects from RGB-D data, approximate them by 3D primitive shapes and use this information to classify them, with a reduced amount of training data.

4.1. Future Work

Taking into account the work that has been done in this dissertation, future work will include:

- Cataloguing more images, in order to improve the accuracy of the algorithm and to include more objects in classification;
- Perfecting the object segmentation algorithm possibly by using a 3D grid or a smaller grid size in order to see the effect that this has on clustering;
- Getting more primitive shapes or a combination of primitive shapes, in order to classify more complex objects;
- Lessen the acquiring parameters in order to speed up the algorithm and see what would be the effect on the classification, namely taking out the third rotation and some denoise operations;
- Implementation of the algorithm in grasping applications, in order to aid robots in detecting and classifying obstacles and graspable objects.

BIBLIOGRAPHY

[1] A. Y. N. Ashutosh Saxena, Justin Driemeyer, "Robotic Grasping of Novel Objects using Vision," *J. Int. J. Robot. Res.*, 2008.

[2] Yun Jiang, S. Moseson, and A. Saxena, "Efficient grasping from RGBD images: Learning using a new rectangle representation," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3304–3311.

[3] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen, "Automatic grasp planning using shape primitives," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, pp. 1824–1829.

[4] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[5] W. P. Chan, K. Nagahama, H. Yaguchi, Y. Kakiuchi, K. Okada, and M. Inaba, "Implementation of a framework for learning handover grasp configurations through observation during human-robot object handovers," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 1115–1120.

[6] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications," *Sensors*, vol. 12, no. 12, pp. 1437–1454, Feb. 2012.

[7] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view RGB-D object dataset," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011, pp. 1817–1824.

[8] A. Johnson, "Spin-images: a representation for 3-D surface matching," *Technology*, no. CMU-RI-TR-97-47, p. 138, 1997.

[9] H. S. Koppula, A. Anand, T. Joachims, and A. Saxena, "Semantic Labeling of 3D Point Clouds for Indoor Scenes," *Neural Inf. Process. Syst.*, pp. 1–9, 2011.

[10] L. Bo, X. Ren, and D. Fox, "Depth kernel descriptors for object recognition," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 821–826.

[11] M. Krainin, B. Curless, and D. Fox, "Autonomous generation of complete 3D object

models using next best view manipulation planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011, pp. 5031–5037.

[12] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Int. J. Rob. Res.*, vol. 34, no. 4–5, pp. 705–724, Apr. 2015.

[13] M. Blum, J. T. Jost Tobias Springenberg, J. Wulfing, and M. Riedmiller, "A learned feature descriptor for object recognition in RGB-D data," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1298–1303.

[14] A. Coates, H. Lee, and A. Y. Ng, "An Analysis of Single-Layer Networks in Unsupervised Feature Learning," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[15] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, 2008.

[16] L. Bo, X. Ren, and D. Fox, "Unsupervised Feature Learning for RGB-D Based Object Recognition," Springer International Publishing, 2013, pp. 387–402.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2012, pp. 1097–1105.

[18] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Scene parsing with multiscale feature learning, purity trees, and optimal covers," *29th International Conference on Machine Learning, ICML 2012*. 2012.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[20] M. Schwarz, H. Schulz, and S. Behnke, "RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1329–1335.

[21] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3D scene labeling," *2014 IEEE Int. Conf. Robot. Autom.*, pp. 3050–3057, 2014.

[22] R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng, "Convolutional-Recursive Deep Learning for 3D Object Classification," *Adv. Neural Inf. Process. Syst. 25*, 2012.

[23] Y. Cheng *et al.*, "Query Adaptive Similarity Measure for RGB-D Object Recognition," in *2015 IEEE International Conference on Computer Vision (ICCV)*,

2015, pp. 145–153.

[24] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust RGB-D object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 681–687.

[25] Y. Jia *et al.*, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the ACM International Conference on Multimedia*, 2014, pp. 675–678.

[26] J. Wang, J. Lu, W. Chen, and X. Wu, "Convolutional neural network for 3D object recognition based on RGB-D dataset," in *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, 2015, pp. 34–39.

[27] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1316–1322.

[28] F. Li, H. Liu, X. Xu, and F. Sun, "Multi-Modal Local Receptive Field Extreme Learning Machine for object recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 1696–1701.

[29] U. Asif, M. Bennamoun, and F. Sohel, "Efficient RGB-D object categorization using cascaded ensembles of randomized decision trees," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1295–1302.

[30] V. Kramarev, S. Zurek, J. L. Wyatt, and A. Leonardis, "Object Categorization from Range Images Using a Hierarchical Compositional Representation," in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 586–591.

[31] B. Dellen and I.A. Rojas, "Volume measurement with a consumer depth camera based on structured infrared light," in *16th Catalan Conference on Artificial Intelligence*, 2013, pp. 1–10.

[32] B. Q. Ferreira, M. Griné, D. Gameiro, J. P. Costeira, and B. S. Santos, "VOLUMNECT: measuring volumes with Kinect," in *Three-Dimensional Image Processing, Measurement (3DIPM), and Applications 2014*, 2014, pp. 5–10.

[33] D. Schiebener, A. Schmidt, N. Vahrenkamp, and T. Asfour, "Heuristic 3D object shape completion based on symmetry and scene context," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 74–81.

[34] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar,

"Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set," *IEEE Robot. Autom. Mag.*, vol. 22, no. 3, pp. 36–52, Sep. 2015.

[35] A. Broad and B. Argall, "Geometry-Based Region Proposals for Accelerated Image-Based Detection of 3D Objects." pp. 1–6, 2016.

[36] A. Maligo and S. Lacroix, "Classification of Outdoor 3D Lidar Data Based on Unsupervised Gaussian Mixture Models," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 1, pp. 5–16, Jan. 2017.

[37] A. Teichman, J. Levinson, and S. Thrun, "Towards 3D object recognition via classification of arbitrary object tracks," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 4034–4041.

[38] W. J. Beksi and N. Papanikolopoulos, "Object classification using dictionary learning and RGB-D covariance descriptors," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1880–1885.

[39] D. Fehr, W. J. Beksi, D. Zermas, and N. Papanikolopoulos, "RGB-D object classification using covariance descriptors," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5467–5472.

[40] E. Martinez-Martin and A. P. del Pobil, "Object Detection and Recognition for Assistive Robots," *IEEE Robot. Autom. Mag.*, pp. 2–17, 2017.

[41] H. Pan, S. I. Olsen, and Y. Zhu, "Object classification from RGB-D images using depth context kernel descriptors," in *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 512–516.

[42] H. F. M. Zaki, F. Shafait, and A. Mian, "Convolutional hypercube pyramid for accurate RGB-D object category and instance recognition," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1685–1692.

[43] H. F. M. Zaki, F. Shafait, and A. Mian, "Localized Deep Extreme Learning Machines for Efficient RGB-D Object Recognition," in *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2015, pp. 1–8.

[44] E. Santana, K. Dockendorf, and J. C. Principe, "Learning joint features for color and depth images with Convolutional Neural Networks for object classification," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 1320–1323.

[45] S. Boubou, T. Narikiyo, and M. Kawanishi, "Differential Histogram of Normal

Vectors for Object Recognition with Depth Sensors," in *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2016, pp. 162–167.

[46] V. Seib, R. Memmesheimer, and D. Paulus, "Ensemble classifier for joint object instance and category recognition on RGB-D data," in *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 143–147.

[47] L. Peppoloni, M. Satler, E. Luchetti, C. A. Avizzano, and P. Tripicchio, "Stacked generalization for scene analysis and object recognition," in *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, 2014, pp. 215–220.

[48] J. Stuckler and S. Behnke, "Benchmarking mobile manipulation in everyday environments," in *2012 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2012, pp. 1–6.

[49] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Graph. Image Process.*, vol. 24, no. 6, pp. 381–395, 1981.

APPENDIX A (ALGORITHM SCHEME)

In this appendix, we will show in an index form of how the algorithm works, showing what functions does it call, when the main function SmartDemonstrator.m is called. The algorithm contains four parts which are:

- Part 1: Obtaining data from Microsoft Kinect V2;
- Part 2: Point Cloud Processing;
- Part 3: Feature Extraction;
- Part 4: Classification.

When functions are called, an index will appear that indicates the order in which the function was called and what it does once it's called, as shown next:

Part 1: Obtaining data from Microsoft Kinect V2.

1. Acquisition of point cloud using fKinectAcquireM;
  1.1. Acquire point clouds from Kinect;
  1.2. Merging of point clouds into one;
2. Remove invalid points from resulting point cloud using ptCloud.removeInvalidPoints;
3. First trimming operation based on the range of grasping using PtTrimLoc;
  3.1. Define shape, dimensions, centre and angles;
  3.2. Create an affine transformation matrix from Angles and then rotates the input point cloud using PtRotTrams;
  3.3. Point cloud analysis – defines region of interest, useful points are inside this region, based on a shape and its limits;
    3.3.1. Define cube limits;
    3.3.2. Define sphere limits;
    3.3.3. Define rectangular prism limits;
    3.3.4. Define ellipsoid limits;
  3.4. Output creation – creates output by selecting only useful points and then plotting the output point cloud on a figure;
4. Denoise and show the image before and after denoise.

Part 2: Point Cloud Processing.

1. Point cloud rotation using FPtFastRot;

1.1. Fit plane to 3-D point cloud using fFitPlaneMatLabOptimize;

1.2. Creates first affine transformation matrix from vectors using RotFromVec;

1.3. First rotation to the vertical of the input point cloud using PtRotTrams;

1.4. Creates second affine transformation matrix from vectors using RotFromVec;

1.5. Second rotation to the horizontal of the input point cloud using PtRotTrams;

1.6. Fine-tune point cloud rotation by iteratively fitting a plane using PtFineTuneRotation;

1.6.1. Create a grid and get statistical data from it using FGrid;

1.6.1.1. Generate limits according to the max/min coordinates using Generate_Zone4Boundary;

1.6.1.2. Create a grid and count points in each cell using FGridCreateACount;

1.6.1.3. Get statistical data from grid using MATLAB functions and FStatGen;

1.6.2. Create a 3D scatter plot from data using F3DPointPlaneForFitting;

1.6.3. Fit a plane using FCreateFitPlaneX1Y1;

1.6.4. Iterative optimization using FFitPlaneRemoveOutliners to remove less similar points to plane and create a new fit plane;

1.6.4.1. Remove less similar points;

1.6.4.2. Fit a plane using FCreateFitPlaneX1Y1;

1.6.5. Rotate the point cloud in order to align the plane to the horizontal using FPTRotTra2Plane;

1.6.5.1. Creates affine transformation matrix from vectors using RotFromVec;

1.6.5.2. Translation of the input point cloud to the origin of the Cartesian axis using PtRotTrams;

1.6.5.3. Rotation to the horizontal of the point cloud using PtRotTrams;

1.6.5.4. Translation of the point cloud using PtRotTrams;

1.7. Show evolution during the optimization plus alignment;

1.8. Show final rotation (result of PtFineTuneRotation);

2. Trimming Section;

2.1. Find centre of point cloud using fPtFindCenter;

2.2. Fine trimming operation using PtTrimLoc;

2.2.1. Define shape, dimensions, centre and angles;

2.2.2. Create an affine transformation matrix from Angles and then rotates the input point cloud using PtRotTrams;

2.2.3. Point cloud analysis – defines region of interest, useful points are inside this region, based on a shape and its limits;

2.2.3.1. Define cube limits;

2.2.3.2. Define sphere limits;

2.2.3.3. Define rectangular prism limits;

2.2.3.4. Define ellipsoid limits;

2.2.4. Output creation – creates output by selecting only useful points and then plotting the output point cloud on a figure;

3. Segmentation of floor using FloorDepthSegmentation;

3.1. Creation and plotting of histograms of counts of points at each coordinate along the Z-axis of point cloud;

3.2. Segmentation of the table from point cloud based on maximum number of points bin and show of point cloud after table segmentation;

4. Point Cloud Clustering and Segmentation;

4.1. Denoise point cloud using pcdenoise;

4.2. Cluster and segment regions of interest from point cloud using ROIClusteringSegmentation;

4.2.1. Obtain grid count and indexes using FGrid;

4.2.1.1. Generate limits according to the max/min coordinates using Generate_Zone4Boundary;

4.2.1.2. Create a grid and count points in each cell using FGridCreateACount;

4.2.1.3. Get statistical data from grid using MATLAB functions and FStatGen;

4.2.2. Get clusters from a grid using FGridClustering;

4.2.3. Remove meaningless clusters;

4.2.3.1. Separate data according to clusters in grid using FProcessCluster;

4.2.3.2. Remove meaningless clusters with density validation;

4.2.3.3. Get final data and plot;

4.3. Remove useless point cloud clusters using fPtSegmentedClean.

Part 3: Feature Extraction.

1. Downsample and denoise point cloud using fPtDownDenoise;

2. Colour extraction by k-means clustering using FFExtractColour;

3. Get volumes, location, shape and score of objects using fPtCloudShapeVolLoc;

3.1. Fit primitive shapes using MATLAB functions by calling fPtCloudGeoMatchMatlab;

3.1.1. Get maximum radius and volume by using fPtCldSize;

3.1.2. Fitting of primitive shapes using MATLAB functions;

3.1.3. Find centre of point cloud using fPtFindCenter;

3.2. Select best shape and score, estimate volume and location using fPtCloudMatchVol;

3.2.1. If volume or score verification is not met use fPtCldSizeCentre to find volume and location of point cloud;

3.2.2. Calculate plane intersection for rectangular prism at the Z-coordinate of the point cloud's centre using fModelPlaneInter;

4. Write output for cataloguing in to Microsoft Excel file using WriteOutputExcel;

5. Offset location based on previous calibration using fOffsetLocation.

Part 4: Classification.

1. Classification and creation of output Microsoft Excel file using fClassification;

1.1. Creation of input matrix for neural network;

1.2. Classification using neural network

1.3. Writing of output in to Microsoft Excel file.

APPENDIX B (RECTANGULAR PRISM TEST)

Method 1 (Horizontal)			
Object	**Volume 1 [cm^3]**	**Volume 2 [cm^3]**	**Volume 3 [cm^3]**
Foam rick	152,0	423,0	420,0
Wood block	1281,0	1308,0	1304,0
Cracker box	1832,0	1875,0	1908,0

Object	**Real volume [cm^3]**	**Average Volume [cm^3]**	**Absolute value of error [cm^3]**	**Absolute percentage of error [%]**
Foam rick	187,5	331,7	144,2	76,9
Wood block	1536,7	1297,7	239,0	15,6
Cracker box	1990,8	1871,7	119,1	6,0
		Average absolute error [cm^3]	167,4	

Method 2 (Horizontal)			
Object	**Volume 1 [cm^3]**	**Volume 2 [cm^3]**	**Volume 3 [cm^3]**
Foam rick	402,0	439,0	482,0
Wood block	1437,0	1334,0	1344,0
Cracker box	1886,0	1975,0	1948,0

Object	**Real volume [cm^3]**	**Average Volume [cm^3]**	**Absolute value of error [cm^3]**	**Absolute percentage of error [%]**
Foam rick	187,5	441,0	253,5	135,2
Wood block	1536,7	1371,7	165,0	10,7
Cracker box	1990,8	1936,3	54,5	2,7
		Average absolute error [cm^3]	157,7	

Method 1 (Vertical)			
Object	**Volume 1 [cm^3]**	**Volume 2 [cm^3]**	**Volume 3 [cm^3]**
Foam rick	234,0	246,0	215,0
Wood block	1278,0	1253,0	925,0
Cracker box	2112,0	2392,0	2605,0

Object	**Real volume [cm^3]**	**Average Volume [cm^3]**	**Absolute value of error [cm^3]**	**Absolute percentage of error [%]**
Foam rick	187,5	231,7	44,2	23,6
Wood block	1536,7	1152,0	384,7	25,0
Cracker box	1990,8	2369,7	378,9	19,0
		Average absolute error [cm^3]	269,2	

Method 2 (Vertical)			
Object	**Volume 1 [cm^3]**	**Volume 2 [cm^3]**	**Volume 3 [cm^3]**
Foam rick	245,0	318,0	244,0
Wood block	986,0	1113,0	1216,0
Cracker box	2433,0	2691,0	2381,0

Object	**Real volume [cm^3]**	**Average Volume [cm^3]**	**Absolute value of error [cm^3]**	**Absolute percentage of error [%]**
Foam rick	187,5	269,0	81,5	43,5
Wood block	1536,7	1105,0	431,7	28,1
Cracker box	1990,8	2501,7	510,9	25,7
		Average absolute error [cm^3]	341,3	

APPENDIX C (CATALOGUING MEANS)

Object	Shape	Score	Measured volume [cm^3]	Global main colour R	Global main colour G	Global main colour B	Main colour 1 R	Main colour 1 G	Main colour 1 B	Main colour 2 R	Main colour 2 G	Main colour 2 B	Main colour 3 R	Main colour 3 G	Main colour 3 B
Sugar box	3,0	0,95	833,0	173,5	184,3	146,0	183,3	196,5	123,7	175,4	188,3	147,2	93,5	98,0	119,8
Potted meat can	2,4	0,86	396,5	125,6	124,6	88,6	142,7	137,2	90,2	84,6	86,4	71,1	153,4	156,2	111,4
Cracker box	3,0	0,94	1867,5	165,0	116,0	95,0	163,2	112,4	100,9	161,0	102,0	64,9	183,0	164,3	147,9
Wood block	3,0	0,97	1487,4	170,2	167,6	126,0	161,8	160,3	117,1	175,3	172,4	131,8	192,3	186,6	149,1
Gelatine box	3,0	0,97	201,4	174,8	153,2	150,6	185,5	180,0	179,6	168,5	119,4	114,1	161,0	122,0	118,4
Pitcher	1,0	0,77	2152,3	33,4	66,8	136,2	27,4	60,8	131,1	44,2	77,4	144,8	116,3	149,4	203,0
Mug	1,0	0,91	348,0	91,4	35,9	25,8	99,7	40,9	28,6	65,8	15,4	8,5	135,5	89,4	85,9
Master chef can	1,2	0,86	693,4	79,3	87,8	125,6	26,6	36,2	89,4	142,1	149,0	169,2	156,1	161,8	170,2
Tomato soup can	1,0	0,91	256,3	151,8	134,3	131,4	160,2	144,7	142,4	149,6	127,6	123,2	126,5	104,5	100,6
Chips can	1,4	0,90	710,8	157,4	100,8	81,1	153,3	51,9	35,8	171,1	157,7	132,9	143,5	128,3	106,7

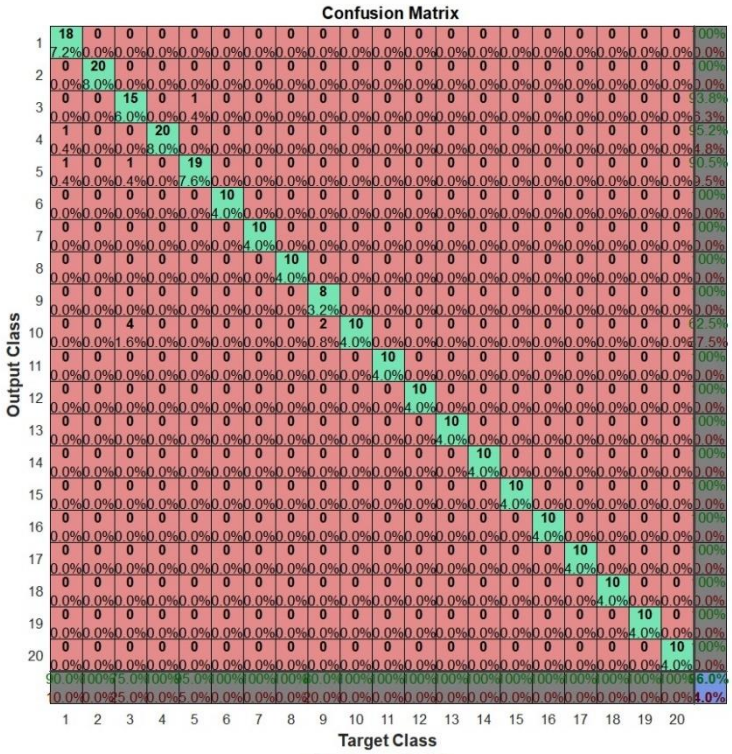
Object	Shape	Score	Measured volume [cm^3]	Global main colour R	Global main colour G	Global main colour B	Main colour 1 R	Main colour 1 G	Main colour 1 B	Main colour 2 R	Main colour 2 G	Main colour 2 B	Main colour 3 R	Main colour 3 G	Main colour 3 B
Apple	2,4	0,91	141,5	162,3	50,0	47,9	150,1	39,6	34,9	183,4	62,9	66,5	207,6	123,3	125,6
Peach	2,6	0,95	70,6	205,2	166,5	53,9	216,8	189,8	54,7	189,5	134,2	48,7	192,1	136,1	66,8
Plum	2,2	0,95	42,0	80,6	43,5	46,6	71,7	34,5	35,6	99,7	61,6	71,9	143,8	112,1	121,6
Lemon	2,5	0,96	50,7	229,7	232,5	62,1	223,9	225,6	56,7	236,4	240,9	65,0	229,1	232,0	85,0
Mini soccer ball	2,0	0,93	1016,6	145,8	99,3	54,7	123,1	62,0	35,8	155,9	119,3	59,7	206,1	190,3	120,3
Orange	2,1	0,93	127,7	198,9	117,7	50,2	191,5	107,2	37,9	211,1	131,7	63,1	211,0	151,6	101,2
Softball	2,0	0,96	342,7	191,6	230,1	84,1	208,6	247,1	91,5	177,7	216,6	78,7	96,6	130,9	41,1
Tennis ball	2,0	0,97	102,9	192,6	217,1	119,5	189,9	215,0	113,9	200,9	225,2	126,1	132,3	155,8	88,8
Baseball	2,0	0,98	145,8	199,8	205,3	207,7	204,1	210,2	213,0	203,2	209,2	211,8	161,6	164,2	165,0
Bowl	1,9	0,81	1090,3	153,5	49,7	35,7	147,5	41,0	25,3	168,8	71,8	65,1	213,4	162,5	162,2

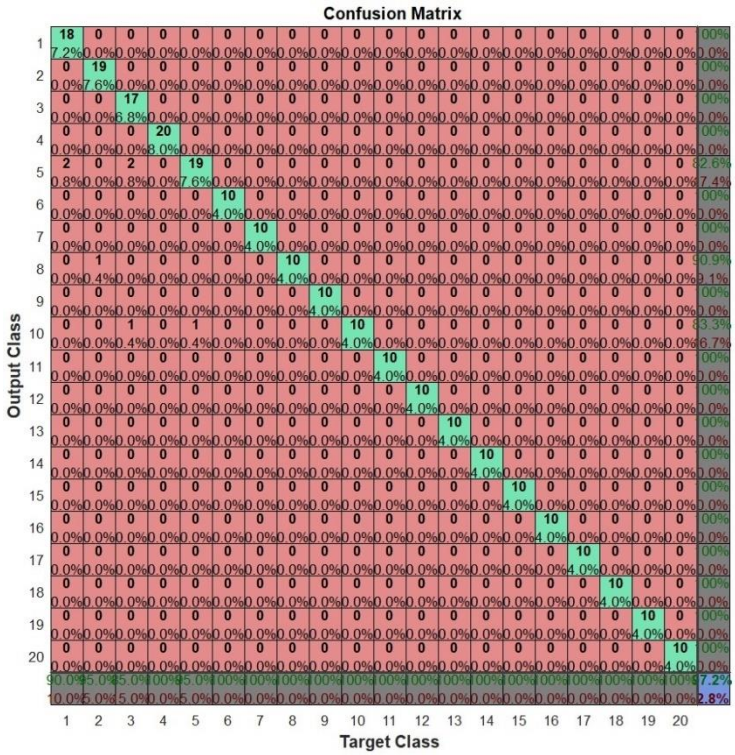
APPENDIX D (CATALOGUING STANDARD DEVIATIONS)

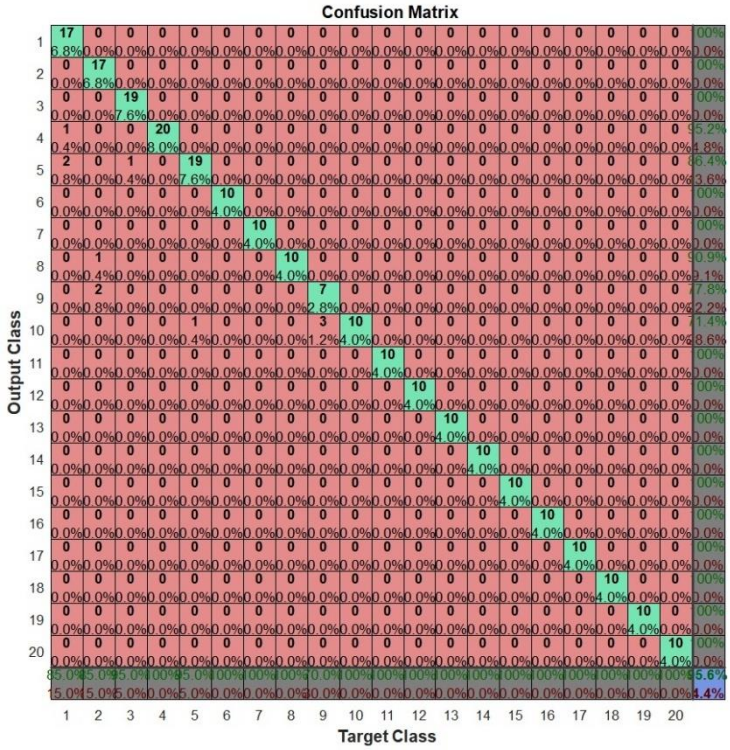
Object	Shape	Score	Measured volume [cm^3]	Global main colour R	Global main colour G	Global main colour B	Main colour 1 R	Main colour 1 G	Main colour 1 B	Main colour 2 R	Main colour 2 G	Main colour 2 B	Main colour 3 R	Main colour 3 G	Main colour 3 B
Sugar box	0,0	0,03	437,0	3,8	3,8	28,5	10,4	14,2	64,9	20,5	21,4	53,8	59,5	60,7	53,2
Potted meat can	0,9	0,08	206,8	15,2	15,5	9,4	28,9	32,0	15,9	66,9	66,8	38,6	68,8	70,9	44,7
Cracker box	0,0	0,02	1058,4	4,8	13,7	20,1	20,6	68,3	74,9	26,4	47,2	35,6	23,1	32,3	45,3
Wood block	0,0	0,04	257,4	9,5	10,4	12,3	8,6	11,8	14,9	33,3	34,2	40,6	37,8	38,8	44,9
Gelatine box	0,0	0,03	142,1	4,9	16,0	18,2	15,3	30,2	32,9	28,1	58,8	64,1	28,8	54,7	56,8
Pitcher	0,0	0,07	945,1	2,7	2,3	2,0	1,5	3,5	7,2	22,5	29,5	31,3	33,4	23,2	11,0
Mug	0,0	0,04	39,4	5,9	1,9	2,6	4,9	1,8	2,7	5,9	1,4	0,6	20,5	28,7	31,5
Master chef can	0,6	0,05	173,5	26,6	26,4	17,6	6,4	6,0	6,1	73,2	73,8	62,1	29,1	30,1	28,3
Tomato soup can	0,0	0,07	78,8	6,6	14,7	16,0	31,8	60,4	65,5	26,3	53,6	59,0	12,5	27,3	29,2
Chips can	0,8	0,05	335,1	10,3	15,8	19,9	14,9	4,6	6,7	14,7	27,8	32,8	50,0	58,5	56,7

Object	Shape	Score	Measured volume [cm^3]	Global main colour R	Global main colour G	Global main colour B	Main colour 1 R	Main colour 1 G	Main colour 1 B	Main colour 2 R	Main colour 2 G	Main colour 2 B	Main colour 3 R	Main colour 3 G	Main colour 3 B
Apple	1,0	0,05	34,8	3,8	4,4	5,3	5,0	0,9	2,1	11,0	12,5	14,0	5,1	37,7	35,6
Peach	0,5	0,02	21,8	10,8	17,5	4,4	8,2	13,4	6,5	32,2	56,6	16,7	35,5	53,8	37,4
Plum	0,9	0,04	13,5	2,5	2,3	2,5	2,5	1,2	2,2	12,3	13,3	13,3	15,5	18,9	18,8
Lemon	0,7	0,01	19,3	5,4	3,3	3,0	9,2	9,1	3,6	9,9	12,9	6,5	21,8	22,8	28,2
Mini soccer ball	0,0	0,03	142,1	4,0	7,5	3,4	13,8	27,8	9,0	11,7	28,9	9,3	9,9	12,7	21,8
Orange	0,3	0,02	25,1	1,9	1,7	2,2	3,1	2,6	1,6	12,5	12,7	11,5	46,2	38,5	43,2
Softball	0,0	0,02	29,8	7,4	6,7	1,2	9,5	4,7	8,7	29,8	25,5	16,5	45,7	47,3	15,1
Tennis ball	0,0	0,02	7,8	10,7	10,1	3,8	7,4	7,0	5,5	47,5	47,1	32,1	67,0	66,2	44,0
Baseball	0,0	0,01	27,3	4,4	4,7	4,7	12,3	12,9	13,9	23,8	25,0	26,7	49,7	50,0	51,2
Bowl	0,3	0,04	163,1	12,0	5,8	5,1	10,4	4,9	4,0	20,5	17,0	21,9	18,1	31,9	32,3

APPENDIX E (CONFUSION MATRICES)






Confusion Matrix
Output Class
Target Class
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

APPENDIX F (TRAINING RESULT)