

Warsaw University of Technology

Faculty of Mechatronics

Institute of Automation and Robotics



**Adam Kosiorek**

# **3D Object Classification based on RGBD images**

bachelor thesis written under supervision of

prof. dr. hab. Barbara Siemiątkowska

Warsaw 2014



*Degree:* Automatic Control Engineering and Robotics

*Major:* Robotics

*Birth date:* 28 February 1991

*Study starting date:* 1 October 2010 r.

## Biography

I was born on 28 February 1991 in Olsztyn, Poland. I attended 5<sup>th</sup> Secondary School by the name of “The Common Europe” in Olsztyn. I started a Bachelor of Engineering degree on the Faculty of Mechatronics of Warsaw University of Technology on 1 October 2010. I did an internship in Faurecia R&D S.A. in 2012. In 2013 I started working in IBM Poland Sp. z o.o. where I stayed for 6 months. Then I started working full-time in Samsung R&D Centre Poland.

.....

signature

# Streszczenie

W pracy przedstawiono podejście Bag of Words do klasyfikacji obiektów na podstawie trójwymiarowych chmur punktów. Wykorzystano metodę uczenia nadzorowanego, dzięki czemu można rozpoznawać dowolne kategorie obiektów. Opisano metodologię BoW, jej znaczenie w wizji komputerowej, a także algorytmy znajdujące zastosowanie w poszczególnych stadiach procesu. Następnie przedstawiono problem klasyfikacji. Określono wymaganą funkcjonalność, zaprojektowano konfigurowalną i łatwo rozszerzalną aplikację oraz zaimplementowano ją w C++. Zbadano algorytmy pod kątem przydatności do BoW oraz sprawdzono skuteczność aplikacji na dwóch powszechnie uznawanych zbiorach danych.

# Abstract

This paper introduces a Bag of Words semantic object classification framework based on three dimensional point clouds. The framework uses machine learning algorithms and a learning by example approach, thus an arbitrary set of categories can be used. Basic concepts of BoW and its role in computer vision are introduced. Then, the most popular algorithms for different steps of BoW are described and a problem of classification is explained. After a requirement analysis was performed a configurable and extensible application was designed and implemented in C++. Finally, algorithms for different steps of BoW classification are evaluated and results on two benchmark datasets are discussed.

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>Introduction</b>		<b>6</b>
1.1	scope and purpose of this work . . . . .	7
1.2	Bow of Words . . . . .	8
1.2.1	Processing Pipeline . . . . .	9
1.2.2	Applications in Computer Vision . . . . .	12
1.3	Problem of Classification . . . . .	14
1.4	Libraries . . . . .	15
1.4.1	Boost . . . . .	16
1.4.2	OpenCV . . . . .	16
1.4.3	PointCloud Library . . . . .	17
1.4.4	libsvm . . . . .	17
1.4.5	Apache log4cxx . . . . .	17
1.5	Datasets . . . . .	17
1.5.1	Berkeley 3D Object Dataset . . . . .	18
1.5.2	The University of Tokyo Dataset . . . . .	20
<b>2</b>	<b>Design</b>	<b>21</b>
<b>Design</b>		<b>21</b>
2.1	Software Functionality and Architecture . . . . .	21
2.1.1	Required functionality . . . . .	21
2.1.2	Architecture . . . . .	23
<b>3</b>	<b>Experiments and Results</b>	<b>26</b>
<b>Experiments and results</b>		<b>26</b>
3.1	Experimental Setup . . . . .	26
3.2	Experiments . . . . .	26
3.2.1	Detectors . . . . .	27
3.2.2	Descriptors . . . . .	27

3.2.3	Codebook	27
3.2.4	B3DO	28
3.2.5	Tokyo	29
3.3	Conclusion	30

# Chapter 1

## Introduction

It has never been easier to capture visual information. Popular storage services grow lighting fast due to terabytes of photographs and movies we upload every day. The growth is so fast that hand-tagging and description, the traditional means of annotation, cease to suffice. They are ambiguous, emotional and rarely optimal. With no better solution at hand, databases are becoming increasingly harder to browse. Another but quite similar setting occurs in the field of mobile robotics. A mobile robot is there to interact with its environment. If so, it would be desirable for the robot to know what kind of surroundings it is in. The problem can be treated as a scene or object classification problem, one of the most popular computer vision issues in the last decade. As Forsyth and Ponce [15] have it:

computer vision is (...) an enterprise that uses statistical methods to disentangle data using models constructed with the aid of geometry, physics, and learning theory.

The area of computer vision is not new and have initially dealt with industrial inspection or mobile robot navigation. Recent applications include human-machine interfaces, medical diagnostics and image retrieval.

This paper tackles two issues. Firstly, the Bag of Words (or BoW for short) approach to image classification is reviewed. The Bag of Words method has originated from the natural language processing domain [ref](#). It makes a strong assumption that the word occurrence in a text document define the meaning of the document. It were Csurka *et al* who have first used BoW for image classification in 2004 [3]. Since we cannot simply

translate an image into natural language a pipeline that help us achieve this goal is needed. It usually consist of: keypoint detection, feature extraction, vector quantization and classification, steps that will be discussed later.

Secondly, a configurable and flexible application for point cloud classification is designed and implemented in C++. Many algorithms are suitable for the bag of words pipeline. To choose an optimal combination for a specific problem we have to perform extensive evaluation of those algorithms. The process can be greatly simplified by a programme that would be: (1) easily extensible, that is adding new algorithms wold be painless and (2) configurable, with every parameter (and algorithm) adjustable at run-time.

The rest of this work is organised as follows: Section 1 summarises purpose and scope of this work. Section 2 describes the Bag of Words technique in detail. Construction of an image model is discussed and the most popular algorithms are outlined. Section 3 provides basic information on the problem of classification, introduces basic concepts and methods of classification. Section 4 names C++ computer vision and general purpose libraries used in this project. Finally, in Section 6 we describe two renown RGBD image datasets we have chosen to test our application at. The following chapters are titled “Design” and “Experiments and Results”. The former analyses functionality of a programme for BoW object classification, its design and implementation details. The latter addresses the topics of experimental setup, performed experiments and a summary of results.

## 1.1 scope and purpose of this work

The goal of this work is to design a flexible and configurable application for point cloud classification. The point cloud should be registered by the Microsoft Kinect RGBD camera. The scope of the work is as follows:

- literature review in the area of bag of words image classification
- design of a pipeline for bag of words classification of point clouds
- design of a configurable programme for point cloud classification
- implementation of the programme in C++ using OpenCV and/or PointCloud Library

- evaluation of algorithms for the bag of words classification
- evaluation of the programme on two scientific RGBD object databases

## 1.2 Bow of Words

The Bag of Words model is a an intermediate representation used in natural language processing, information retrieval and data mining [pub](#). The model can be seen as a simplification, for it obliterates any grammatical information and even word order contained within an original text document. What is left is an unstructured group of words, usually in a form of a vector or a histogram of word incidence.

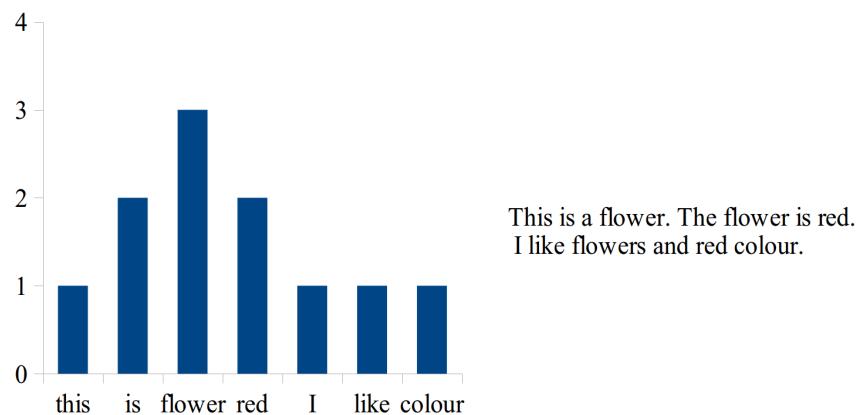


Figure 1.1: Bag of Words histogram

The figure 1.1 shows a piece of text and its BoW model, where articles and punctuation marks were left out. In order to compare different documents a global dictionary must be built. The dictionary (codebook) is constructed by taking every word from all available documents and removing duplicates. One can image that if a dictionary is of any considerable size resulting representation of especially small documents will be very sparse. Classification algorithms can be optimised for sparse data in order to boost performance.

In natural language processing BoW is used to infer semantic meaning of documents [pub](#). If we could translate an image into a text document we might be able to employ similar methods. A question arises: How does one make a text document from an image?

### 1.2.1 Processing Pipeline

Tsai describes a well established pipeline that allows translation of images into text documents [23]. It consists of the following steps: (1) region detection, (2) feature extraction, (3) vector quantization and (4) BoW histogram formation as can be seen in the figure 1.2. The author summarises the most commonly used methods for performing these tasks.

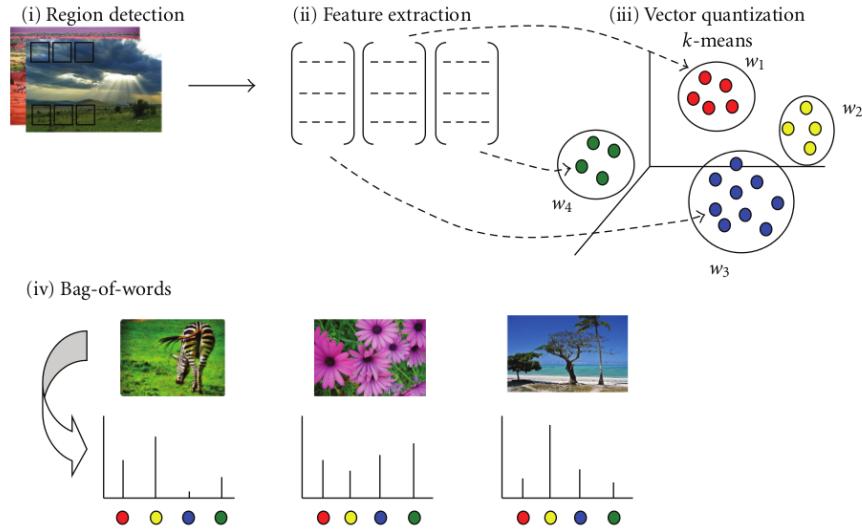


Figure 1.2: Bag of Words pipeline. The figure comes from [23]

### Region Detection

Characteristic region detection is the first step in any Bag of Words framework. Numerous detection methods have been developed, but choosing the right one for any particular case might prove tricky.

**describe fundamentals of keypoint detection in greater detail** The most common detectors make use of a Harris corner detector or image's first or second derivatives. A Harris-Laplace detector is an example of a Harris-based detector. The Harris function is scale adapted and its outcome is a subject to a Laplacian-of-Gaussian (LoG) operator, which selects relevant points in scale space. Images' regions' 2<sup>nd</sup> derivatives, namely the regions' Hessians, can be combined with a LoG operator as well. This combination allows selection of points significant in the two spaces: the scale space and the Hessian's determinant space. The latter entails the speed at which pixel intensities change in the neighbourhood of a point.

A number of more advanced recipes for salient region localisation have been developed

and implemented. These include for example SIFT [11], SUSAN [20] and Intrinistic Shape Signatures [25]. The majority of keypoint detection formulas is being developed for the 2D domain. A number of them have been adapted to 3D, however. A comparative evaluation of detection algorithms available in the PointCloud Library (PCL, discussed below) can be find in [5]. Another comprehensive study is [19]. All these formulas, called sparse feature detectors, resort to selection of maxima in specific state spaces. An entirely different scheme is to use a dense feature detector. That is, take a uniformly sampled grid of points. Dense detectors have an advantage in that they sample slow changing regions in terms of gradient or hessian of an image. Examples of such regions would be a clear sky or a calm ocean. Li *et al* showed that dense detectors generally outperform sparse ones [4].

## Feature Extraction

Suppose only coordinates of a keypoint were known. In case of even the smallest rotation or translation they would change and the keypoint would be lost. Keypoints should be described in a way that makes them invariant to affine transformations, changes in light intensity or colour saturation. It is hard to achieve all of these properties simultaneously, but methods that meet some of them exist.

Keypoint description takes form of coordinates in a high-dimensional space. One of the most precise and repeatable algorithm is SIFT [12], which is a 3D histogram of gradients structured as a 128-dimensional vector of floating point values. It is the most often extracted descriptor in BoW pipelines [ref](#). Other methods include various colour descriptors, binary descriptors such as 512-dimensional GIST [15]. There are techniques designed for 3D exclusively. Among them are Persistent Point Feature Histogram (PFH) [18], its faster alternative FPFH [16] and PFHRGB, which takes into account colour information, all implemented in PCL.

## Vector Quantization

When features are extracted they have to be normalised. Vector quantization step have two main phases. The first one is a codebook construction from a training dataset. The second phase is responsible for parsing (or translation) raw image descriptors into a form compatible with the newly constructed codebook. The simplest way of building a dictio-

nary is to find patterns or regions within the descriptors computed on a training dataset. Then, the parsing phase is about assigning a descriptor to one of the codebook elements (e.g. to one of the regions). Suppose we use centroids or medoids of the discovered regions as our codebook’s entries. Then, we can match a descriptor to one of the centroids performing a nearest neighbour search. The descriptor parsed this way is called a *visual word*. Finally, we build a histogram of visual word occurrence. The histogram have a fixed size equal to the codebook size. Each bin of the histogram tells us how many visual words of a particular kind were in the description of the corresponding image.

The kMeans is the single most popular vector quantization algorithm used in the BoW pipelines [23]. Developed in 1950’s, it is well known and simple to implement. kMeans divides all data into a predefined amount of clusters and computes the clusters’ centroids. Many modifications and alternative versions have emerged [7]. Some of them are: faster than the original *approximate kmeans*, *hierarchical kmeans*, which automatically chooses the final number of clusters and a *soft kmeans* — a variation of the algorithm that allows a fuzzy alignment. The fuzzy alignment mean that each point can belong to several clusters with different weights. A weight can be proportional for instance to the inverse square of the distance to a centroid.

If computational cost is of no concern, or if required precision is of the utmost priority, a Gaussian Mixture Model (GMM) can be used. The GMM partitions data into a set of clusters, finds their means and covariances. In the parsing step we compute a probability distribution of a descriptor over all the clusters. We can then build a fuzzy aligned histogram taking the probability distribution as weights. The GMM can be thought of as a generalisation of the kMeans algorithm. The drawback of the GMM is its massive computational cost in comparison with still expensive kMeans. Recently, Perronnin *et al* proposed GMM based fisher-kernel vector quantization step with superior results [13, 14].

After we choose a vector quantization algorithm, we have to decide what size of a dictionary to use. In a simplest case, when the kMeans is used, we can treat each clusters’ centroid as a visual word. Therefore, the dictionary size defines how much information we retain in a BoW histogram. A big training set containing many classes with large inter-class and inter-class variance is likely to require a huge codebook. On the other hand, too big a dictionary might introduce quantization artefacts. One has to bear in mind that the vector quantization step is the most time-consuming part of the BoW pipeline. The

highest computational complexity is associated with the codebook construction step and it is  $O(n^3)$ , where  $n$  is the codebook size.

It is possible to combine several feature extraction algorithms before creating the codebook (Early Fusion) or create many codebooks and concatenate resulting histograms (Late Fusion). Both schemes are as simple as vector concatenation. The Early Fusion concatenates two feature vectors if they have been extracted from the same keypoint. The Late Fusion joins histograms, outputted by separate quantization steps.

### 1.2.2 Applications in Computer Vision

The Bag of Words approach is a tool than enables us to represent an image by a single feature vector of a chosen size. Its most popular applications in computer vision are Content Based Image Retrieval (CBIR) and Scene/Object Classification.

#### Content Based Image Retrieval

CBIR is a computer vision approach to image retrieval from large image databases. Tangledler *et al* provides an overview of techniques applicable to 3D objects [21]. The task of image retrieval is to find a database entry that fulfil certain conditions. If we wanted to perform the task efficiently, we would have to meet the following criteria [22]: (1) All entries should be indexed in a concise way, (2) a (dis)similarity measure should be provided and (3) an efficient search algorithm should be available.

Database entries (images) should be indexed beforehand. Otherwise we would have to compare all the entries explicitly. An index must be compact as well as discriminative. Suitable algorithms can be divided into three general categories: (1) feature based methods, (2) graph based methods and (3) other methods.

Feature based methods can be further divided into global features and local features. The global features take form of a single vector (or a point in a  $d$  dimensional space). They are usually associated with object's mass or volume or distributions of these. Being easy to compute and straightforward to implement, their discriminative power is rather low — they cannot be used for partial matching, but are well suited for an early preprocessing.

Local features, on the other hand, describe object's characteristic regions. Their shape is similar to that of the global features, but instead of a single point in space there are multiple ones — one for each considered region. Tangledler argues that local features

based approaches are inefficient and lead to a complex indexing problem [21]. At the same time Toldo *et al* and Li *et al* show that Bag of Words local feature based approach has no such drawbacks. On the contrary — it is easy to implement, efficient and provides state-of-the-art results.

Feature based methods can be either local or global. Local features describe characteristic regions of an object, which have to be identified first. We have to carry out similar computation for each salient region. An outcome is a set of vectors, one vector for each region in  $d$  dimensional space, where  $d$  is dimensionality of the descriptor. Global features are generally easier (faster) to compute and more compact — they take form of a single  $d$  dimensional vector. Often global features are more computationally efficient and can be easily used for object indexing . Before the advent of BoW it was not clear how to use local features for indexing, nor there was any straight-forward dissimilarity measure available [21]. The Bag of Words approach solved these issues. What is more, it is easy to implement, efficient and provides state-of-the-art results [10].

## Scene and Object Classification

Scene and object classification are among the most popular issues of computer vision nowadays [ref](#). We would like to perform those tasks automatically for the sake of surveillance, navigation or automatic image tagging. The very first attempt to use BoW in scene classification was made by Csurka *et al* in 2004 [3]. They were inspired by an idea of a “texton”, a building block of texture introduced earlier in pattern recognition and texture classification. The authors suggested a BoW pipeline described above. Then, they fed the resulting BoW histograms into two classifiers: Support Vector Machine (SVM) and Nàve Bayes (both discussed below). Fei-Fei *et al* refined the original approach by examining several keypoint detectors and descriptors [4]. Moreover, they developed a novel probabilistic graphical model for classification. They achieved accuracy of 76% on a large 13 category dataset.

One of the best entries in an ILSVRC2013<sup>1</sup> challenge, attempting to classify millions of images into 1000 categories, was submitted by a fisher vector based BoW approach as well. The top-5<sup>2</sup> accuracy rate was close to 85%.

---

<sup>1</sup><http://www.image-net.org/challenges/LSVRC/2013/results.php>

<sup>2</sup>top-5 — the classification result is successful if the ground-truth category is in one of the five most probable categories predicted

## 1.3 Problem of Classification

To classify means to, given a set of categories, produce a category label for a given set of features [15]. Image classification fits this description perfectly. The only difficulty lies in complexity of a raw image. Suppose we have a 320 by 240 three channel colour image. This amounts to a total of 230400 dimensions, far too many to feed into any classifier directly. Fortunately, a Bag of Words histogram is a compact and discriminative intermediate representation that solves this issue. It can be used with any classifier. Classifiers can be two-class classifiers (e.g. binary) or multi-class classifiers. The latter are often a combination of several binary classifiers. The most popular classifiers are: k-Nearest Neighbours, Logistic Regression, Softmax Regression, Naïve Bayes and Support Vector Machine.

### Naïve Bayes

Assume that we have a  $n$ -dimensional feature vector  $\mathbf{X}$  such that  $\mathbf{X} = \{x_1, x_2, \dots, x_n\}$  and an unknown category label  $C$ . The probability of the category  $C$  given the feature vector  $\mathbf{X}$  is  $P(C|\mathbf{X}) = P(C|\{x_1, x_2, \dots, x_n\})$ . The strong (or naïve) Bayes assumption says that features are independent of each other, that is  $P(\mathbf{X}) = P(x_1) * P(x_2) * \dots * P(x_n)$ . If so, we can compute probability distributions of every class in our training set given every feature. Then, for any new feature vector the joint probability distribution over a set of features can be calculated and an appropriate class label can be assigned.

### k-Nearest Neighbours (kNN)

The nearest neighbours classifier assumes that if there are labelled points in a neighbourhood of a newly added point, then the new point can be classified based on it's neighbours' labels. There is an issue of choosing the correct number of nearest neighbours (or  $k$ ) to consider.

### Support Vector Machine (SVM)

Assume that a set of pairs  $\{\{\mathbf{x}_1, y_1\}, \{\mathbf{x}_2, y_2\}, \dots\}$  where  $x_i$  are features and  $y_i \in \{-1, 1\}$  are labels is given. Further assume that points with different labels are two linearly-separable datasets as shown in the figure 1.3, where dots and circles are points with

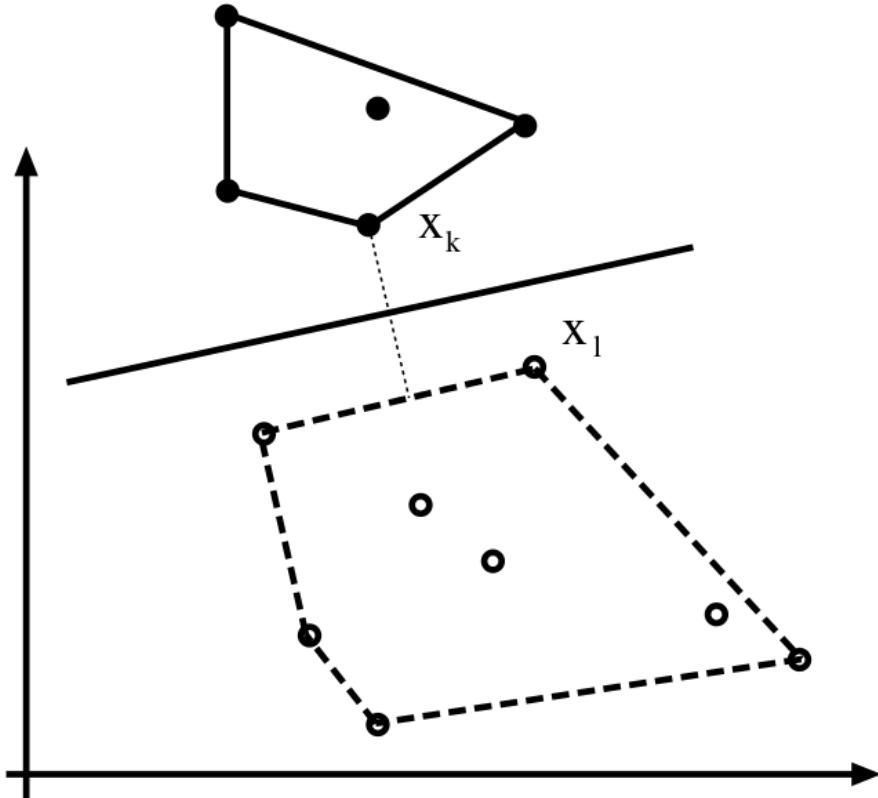


Figure 1.3: Classification by a Support Vector Machine. The distance of the separating plane from both datasets is maximal. The figure was originally published in [15]

different labels. Then, parameters  $\mathbf{w}$  and  $b$  exist such that

$$y_i (\mathbf{w} * \mathbf{x}_i + b) > 0 \quad (1.1)$$

for every  $\mathbf{x}_i$  or a data point. This equation specifies constraints on a plane (or a hyperplane in general) that separates different classes. A Support Vector Machine finds the parameters  $\mathbf{w}$  and  $b$  so as to maximise the distance of the plane from members of both classes. We can combine several SVM classifiers in a 1-versus-all or an all-versus-all scheme.

## 1.4 Libraries

We use the following third party C++ libraries:

- Boost<sup>3</sup> — of the biggest general purpose libraries

---

<sup>3</sup><http://www.boost.org/>

- OpenCV<sup>4</sup> — a general computer vision library
- PointCloud Library<sup>5</sup> [17] — a library for point cloud processing
- libsvm<sup>6</sup> [2] — a renown and very efficient SVM classifier implementation
- Apache log4cxx<sup>7</sup> — Easy to use and intuitive logger

### 1.4.1 Boost

Boost is a general purpose library available under the Boost Software License<sup>8</sup>. It is free for both commercial and non-commercial use. Boost comprises of more than 50 separate libraries. 12 of the most popular libraries have been incorporated into the new C++11 standard. Because of that, Boost may become redundant in many applications. The original library had contained conveniences such as smart pointers, multi-threading utilities and meta-programming tools. We excessively use smart pointers and meta-programming utilities; but the only package from the trimmed down Boost used in this project is the Boost.program\_options library. It enables convenient parsing of configuration files and command line arguments.

### 1.4.2 OpenCV

OpenCV is a general computer vision library available under the BSD License<sup>9</sup>. It is free for commercial and non-commercial use. It was designed for computational efficiency and real-time applications. OpenCV takes advantage of the underlying hardware employs hardware-specific optimisation e.g. SSE4. Some algorithms are written in nVidia CUDA and can use GPU for faster execution. Others use Threading Building Blocks (TBB, Intel's multithreading library) to enable concurrency. OpenCV is available for the most popular desktop and mobile platforms. It is written in optimised C and C++ and provides two interfaces: the original OpenCV 1.x C interface and a newer OpenCV 2.x C++ interface. This single library allows production of fully functional computer vision programmes.

---

<sup>4</sup><http://opencv.org/>

<sup>5</sup><http://pointclouds.org/>

<sup>6</sup><http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

<sup>7</sup><http://logging.apache.org/log4cxx/>

<sup>8</sup>[http://www.boost.org/LICENSE\\_1\\_0.txt](http://www.boost.org/LICENSE_1_0.txt)

<sup>9</sup><http://opensource.org/licenses/BSD-3-Clause>

Many machine learning routines are available, too. Among them are: kMeans, Gaussian Mixture Model, Naïve Bayes, SVM, Random Forests and Boosting. We use OpenCV for input/output operations, image processing and vector quantization.

### 1.4.3 PointCloud Library

PointCloud Library (PCL) is a computer vision library suitable for point clouds processing, manipulation and visualisation. It is released under the BSD license and is free for both commercial and non-commercial use. PCL contains numerous state-of-the-art algorithms for feature detection, surface reconstruction, segmentation to name just a few areas. It is well documented and easy to use. Recently, a machine learning module containing SVM, kMeans and other algorithms. Some routines are multi-threaded thanks to the OpenMP library. A GPU module makes use of CUDA to lessen computation time. Unfortunately, the PCL is relatively new and its stability could be better. Currently available 1.7.1 version is going to be replaced by a soon-to-be-announced 2.0 version with incompatible API. In this project we try to build an application for point cloud applications. We extensively use PCL for operations like downsampling, keypoint detection, feature extraction.

### 1.4.4 libsvm

**describe**

### 1.4.5 Apache log4cxx

**describe**

## 1.5 Datasets

Machine Learning algorithms provide a way of learning a function that maps input to the desired output. Classification algorithms require supervised training and labelled data. If we wanted to compare any algorithms we would have to do so on a common dataset. That

is why multiple scientific datasets have been created: ImageNet<sup>10</sup>, PascalVoc<sup>11</sup>, LabelMe<sup>12</sup> and SUN<sup>13</sup> among others. Unfortunately, they contain RGB and grayscale images without any depth information. Therefore they cannot be used in this paper. On the other hand, the vast majority of RGBD datasets are focused on tracking or instance-level recognition. Others have very few categories or insufficient number of examples per category. We managed to find only two datasets suited to our needs. They are: the Berkeley 3D Object (B3DO) dataset [8] and a dataset compiled by Zhang *et al* at the University of Tokyo [24]. The latter will be abbreviated Tokyo dataset from now on.

An interesting dataset is provided by the University of Washington [9]. It consists of video sequences that capture objects' round views in RGBD. Kinect Fusion<sup>14</sup> technology could be used to prepare excellent point clouds from the videos. We have decided against it, for a huge computational cost associated with this way of point cloud construction. Furthermore, in a real setting a mobile robot would have to circle an object in order to gather enough data.

### 1.5.1 Berkeley 3D Object Dataset

The Berkeley 3D Object dataset was specifically designed for the purpose of object detection and classification. It consists of cluttered images in indoor environment. There are around 50 classes with more than 20 examples in each of them. RGBD data are provided as pairs of colour images (8 bit RGB jpeg files) and depth maps (16 bit png files). Images are densely labelled — for every pair of images there is an xml file with annotations. Each annotation states a category of an object as well as its bounding box location. Images are cluttered, with many labelled and random objects per image. Labelled objects are often partially occluded. We do not address image segmentation nor object detection; we had to extract objects from the images before we could use the dataset.

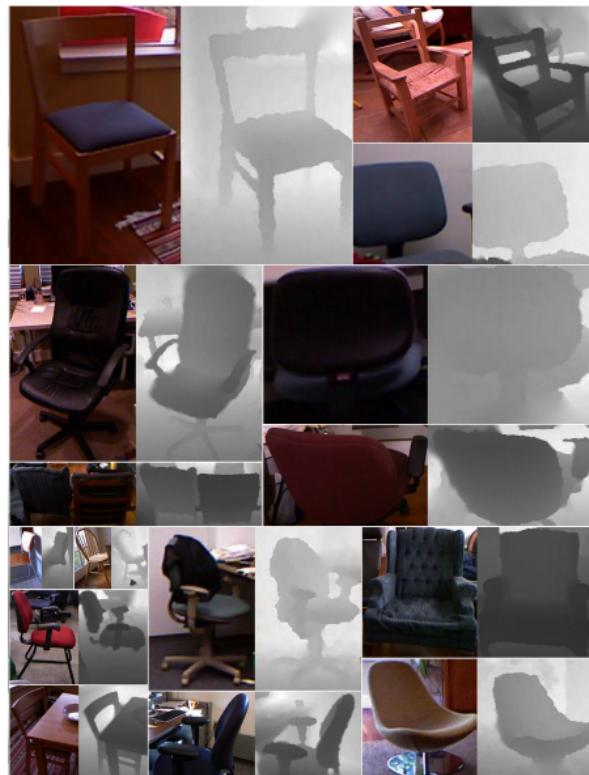


Figure 1.4: Berkeley 3D Object Dataset. The figure comes from [8]

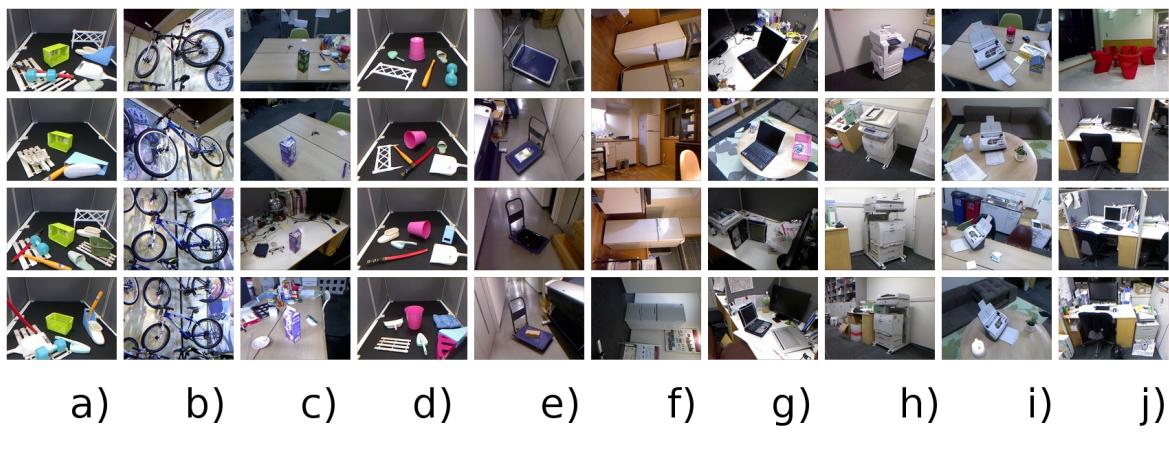


Figure 1.5: The Tokyo dataset: a) basket b) bicycle c) box d) bucket e) cart f) freezer g) notebook h) printer i) scanner j) scene

### 1.5.2 The University of Tokyo Dataset

The Tokyo dataset is comprised of high quality images of objects in different settings, with changing viewpoint, object orientation and texture. The dataset is presented as colour images (8 bit RGB jpeg files) and depth maps (csv files with XYZ coordinates of each pixel). This dataset is aimed at the task of object classification in casual images. There is one labelled object per image, without any bounding box given. Scenes are sometimes cluttered, with many random objects present at a scene.

---

<sup>10</sup><http://www.image-net.org/>

<sup>11</sup><http://pascallin.ecs.soton.ac.uk/challenges/VOC/>

<sup>12</sup><http://labelme.csail.mit.edu/Release3.0/browserTools/php/dataset.php>

<sup>13</sup><http://groups.csail.mit.edu/vision/SUN/>

<sup>14</sup><http://msdn.microsoft.com/en-us/library/dn188670.aspx>

# Chapter 2

## Design

### 2.1 Software Functionality and Architecture

We have decided to call our project “Tagger3D”. Since the purpose is to classify (or tag) point clouds, the name seems suitable. An optimal combination of algorithms for a bag of words pipeline is task and data specific. To find a combo of routines that yields superior performance we have to evaluate numerous algorithms. The process can be simplified; We are going to design an application that can be easily configured and extended. It should be possible to configure and choose an algorithm at run-time. Moreover, it should enable real time classification if it is to be used in any real setting.

#### 2.1.1 Required functionality

There are three major functions that have to be supported by the programme:

1. train — performs training
2. test — evaluates the previously trained classifier
3. infer — predicts a category of a single object

Before any of them can be implemented, we have to identify common features. Inference and testing is very similar. The only difference is that the latter should process a whole testing dataset and compute statistics. The inference and training processes looks as follows:

Inference:

1. RGBD image reading and preprocessing
2. Point cloud construction
3. Salient region identification
4. Feature Extraction
5. Vector Quantization
6. Classification

Training:

1. RGBD image reading and preprocessing
2. Point cloud construction
3. Salient region identification
4. Feature Extraction
5. Codebook Construction
6. Vector Quantization
7. Classifier training

The only differences is that training stage requires codebook construction and classifier training, whereas inference uses the classifier for prediction. If we compare training to testing it is clear that they both deal with large amounts of data. Inference is not that data-intensive and can be carried out for a single image. In order to simplify training we have decided to split the training and testing phases into several sub-routines. Such treatment let us change settings of an algorithm without the need to retrain the stages that lay before considered algorithm in the pipeline (*e.g.* there is no need to extract features once more only because we have changed the dictionary size). The final application has the following operational modes, accessible from the command line:

1. Feature Extraction
2. Codebook Generation
3. Classifier Training
4. Classifier Evaluation
5. Inference

In order to ensure that the application is configurable and easily extensible we used singleton and strategy design patterns and resorted to the Boost's program\_options package for configuration file parsing.

## **Input/Output operations**

This work is extremely data dependent. Every operation requires external data. Both training and testing a classifier consumes hundreds of images. Datasets used in this projects have different and incompatible formats. Not only an object responsible for run-time input/output operations is needed, but also a set of scripts for data preprocessing. Additionally, some serialisation method have to be supplied. The training of the classifier is far too computationally expensive to be computed every time the application starts. Once the classifier is trained, all the models should be stored for further usage. A way of saving and loading codebook and svm model is required.

## **Configuration**

Every algorithm used in this work have several configuration parameters. If the parameters were to be compiled with the code, every change would require a rebuild of parts of the application. A run-time configuration functionality is essential.

## **Interchangeability of algorithms**

Multitude of available algorithms makes it time consuming and computationally expensive to evaluate every available option. It should be possible to build the programme with a number of algorithms suitable for performing the same task and decide which one should be used at run-time.

### **2.1.2 Architecture**

C++ is one of the most efficient languages on the market. It is perfect for Object Oriented Programming (OOP) — a paradigm that makes writing modular, flexible and extensible software easier. The excellent meta-programming tools provided by the language help to minimise the length of the hand-written code. Among OOP's best practices is to use design patterns - a set of well-known, documented and tested methods of solving certain architectural problems. Using design patterns leads to cleaner code which is easier to maintain and reuse.

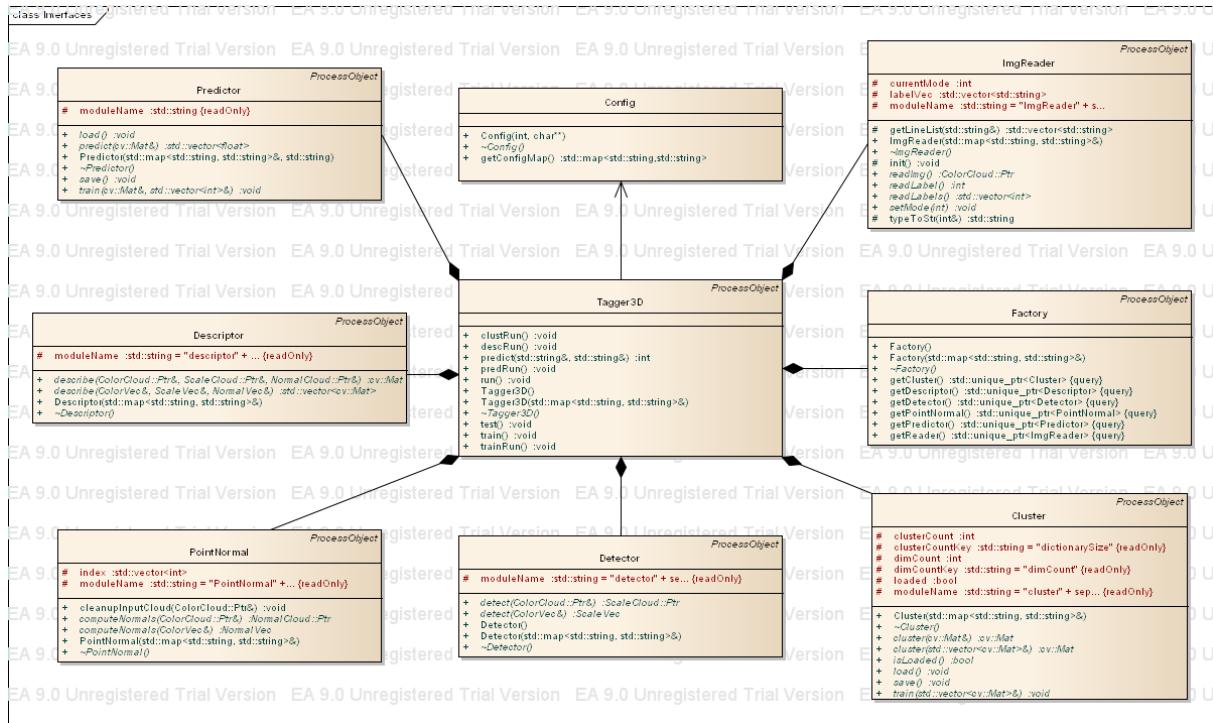


Figure 2.1: Tagger3D class diagram

## Configurability

The programme is configurable through a text file of a standard \*.ini format. It should be formatted as follows: param1 = 2 param2 = aha [section] param3 = no

On top of the file there is a global section. Parameters can be accessed by their names. We can define sections but putting a section's name in brackets. The param3 can be accessed as “section.param3”. An object “Config” was created for the purpose of configuration file parsing. It uses the Boost.program\_options module. Returned associative array of (parameter, value) pairs contains all parameters from a configuration file. We use a convention that if any object has to be configured by the configuration map it has to accept this map in its constructor.

## Strategy Design Pattern

Design patterns were for the very first time by Gamma *et al* in 1993 [6]. They are best practices that should be used to solve recurring problems in software design. One of the patterns is the strategy pattern. It enables implementation of multiple algorithms to handle one task. Then, a specific algorithm can be selected at run-time. It is the exact problem encountered here. It is stated that an interface should be defined for

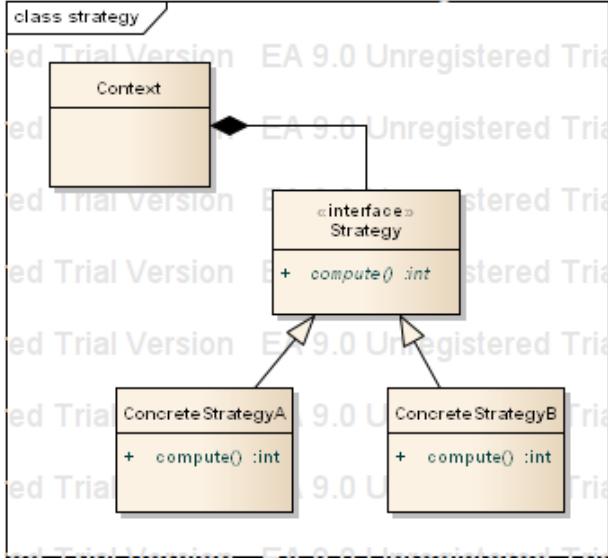


Figure 2.2: Strategy pattern

each task and multiple classes encapsulating different algorithms for handling the task should realise the interface. One modification has been done: Algorithms have often some common operations, thus an idea of an interface (a class without any implemented methods) was dropped in favour of an abstract class with some methods implemented.

The strategy pattern was used for every processing step of the pipeline so as to make it extensible. This property can be enhanced further by using a pattern of abstract factory. The idea was dropped in this case, however. The project is relatively small and the bulk of an abstract factory is beyond reason.

Six parts of the pipeline are a subject of strategy pattern. The interfaces representing them are: ImgReader, PointNormal, Detector, Descriptor, Cluster, Predictor. Each of them contains pure virtual computation methods — 'read' in case of the ImgReader or 'predict' in case of the Predictor. Additionally, they have some common methods defined — I/O methods in Cluster or batch processing methods (such that take a vector of elements instead of a single element as an input) for the Detector and the Descriptor. Every pure virtual class have its specialisations. For instance there is the PFHDescriptor class with a Point Feature Histogram descriptor or a KMeansCluster employing kMeans clustering algorithm. The Factory class chooses specialisations of the algorithms at run-time, depending on the configuration parameters.

# Chapter 3

## Experiments and Results

### 3.1 Experimental Setup

Thanks to the configurability of the programme we were able to run the majority of experiments in a batch processing mode. That is, we wrote a script in bash that allowed running multiple tests in series without any maintenance. In a single series of tests any parameters could be modified or any data processing algorithm could be exchanged. All experiments were run on a notebook with Intel Core i7 quad core 2.2 GHz CPU, 8 GB of RAM and a nVidia M540 GPU.

### 3.2 Experiments

As already stated, a BoW pipeline consists of: (1) keypoint detection (2) feature extraction (3) vector quantization and (4) classification. The number of possible combinations grows combinatorial with the amount of candidates for each part. We have to bear in mind that each algorithm contains a set of parameters that have to be adjusted for optimal performance. It would require a great deal of time and computational power to conduct exhaustive evaluation. Even if we consider only a single library — PCL, we would have far too many options with 12 keypoint detectors and more than 20 feature descriptors. A literature review can help perform an early selection of algorithms.

### 3.2.1 Detectors

run tests of SIFT, ISS, US Keypoint detection algorithms have been compared with respect to repeatability, invariance [5] as well as object retrieval and recognition [19]. 3D-SIFT and ISS achieve the highest robustness for invariance. ISS appears to be the best choice for object retrieval. Dense detector (or uniform sampling) achieved the best performance in 2d image classification [23]. Therefore, this paper compares SIFT, ISS and Uniform Sampling keypoint detection techniques.

### 3.2.2 Descriptors

run tests on all PFH and SHOT descriptors and USC PCL descriptors have been evaluated in terms of object retrieval performance [1]. PFHRGB and SHOTCOLOR are the most successful. Simpler versions of those, that is: PFH, FPFH, SHOT and USC scored a little worse. We evaluate all of them in terms of BoW performance.

Descriptor	Accuracy [%]
FPFH	65.22
PFH	59.32
PFHRGB	63.35

Table 3.1: Highest accuracy obtained with FPFH, PFH and PFHRGB descriptors on the B3DO dataset

The figure 3.1 shows highest accuracy obtained with the tested descriptors on the B3DO dataset. FPFH scored the best result, even though it is an approximate and theoretically the least precise of all evaluated descriptors. On the other hand, the PFHRGB scored better than PFH. It does not surprise, since the colour information is believed to have discriminative value.

### 3.2.3 Codebook

implement and run fisher KMeans is used in almost every implementation of the Bag of Words image processing pipeline [22, 23]. Csurka *et al* showed that the number of visual words has a significant impact on the final results [3]. We run a number of tests to determinate an optimal codebook size.

The results from the figure 3.1 partially confirm Csurka's findings. The performance raises with the increasing number of centroids up to the point of 1500 centroids and 65.22%. Then it starts to diminish. The discrepancy might be caused by the following

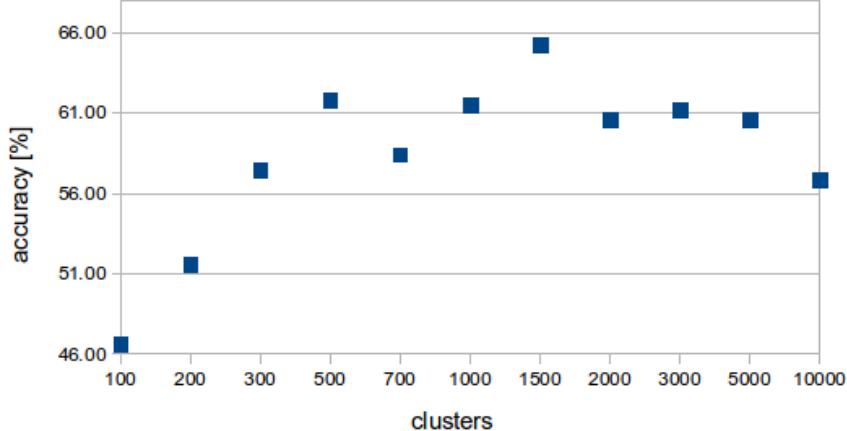


Figure 3.1: Influence of dictionary size on the overall accuracy. B3DO dataset with ISS keypoint detector and FPFH feature descriptor

factors: (1) ISS detector finds too many or irrelevant points, thus introducing noise or (2) The FPFH descriptor has too few dimensions (33) to be divided into more than 1500 regions in a meaningful way.

### 3.2.4 B3DO

Extraction of single objects yielded 78 separate categories with number of entries ranging from 1 (tape) to 299 (table). 8 categories were picked at random with a restriction that there should be at least 50 instances in each of them. Further, the objects were split into two sets (training and testing) with a proportion of 1:1. As names suggest, the estimation is performed on the test set and the evaluation on the test set.

The highest accuracy achieved for this dataset is 64%. Table 3.2 contains a confusion matrix, number of examples per category and accuracy. The confusion matrix depicts misclassification errors. Let  $m_{i,j}$  be an element of the confusion matrix at the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column. It shows how many elements from the  $i^{\text{th}}$  category was assigned to the  $j^{\text{th}}$  category. A high value of  $m_{i,j}$  such that  $i \neq j$  indicates that the classifier cannot distinguish between those two classes.

More than half of the bottles were assigned to the cup category and keyboards were often mistook for books. These two error types are easily explained by a high degree of similarity of objects (bottles and cups are usually round and tall, books and keyboards are flat and rectangular) Surprisingly, however, objects from half of the categories were

Table 3.2: Results on the B3DO dataset with ISS keypoint detector, FPFH features and a dictionary of 1500 words. **Overall accuracy is 65.22%**

Assigned to category \\ Category	cup	book	sofa	chair	table	bottle	monitor	keyboard	Entries per class	Accuracy [%]
cup	<b>59</b>	0	0	0	0	5	0	0	64	92.19
book	17	<b>14</b>	0	0	0	7	0	2	40	35.00
sofa	0	0	<b>17</b>	3	1	0	0	0	21	80.95
chair	0	0	2	<b>14</b>	4	0	2	0	22	63.64
table	0	0	0	0	<b>31</b>	0	0	0	31	100.00
bottle	34	0	0	0	0	<b>30</b>	2	0	66	45.45
monitor	7	1	0	2	0	10	<b>30</b>	1	51	58.82
keyboard	8	4	0	0	0	0	0	<b>15</b>	27	55.56

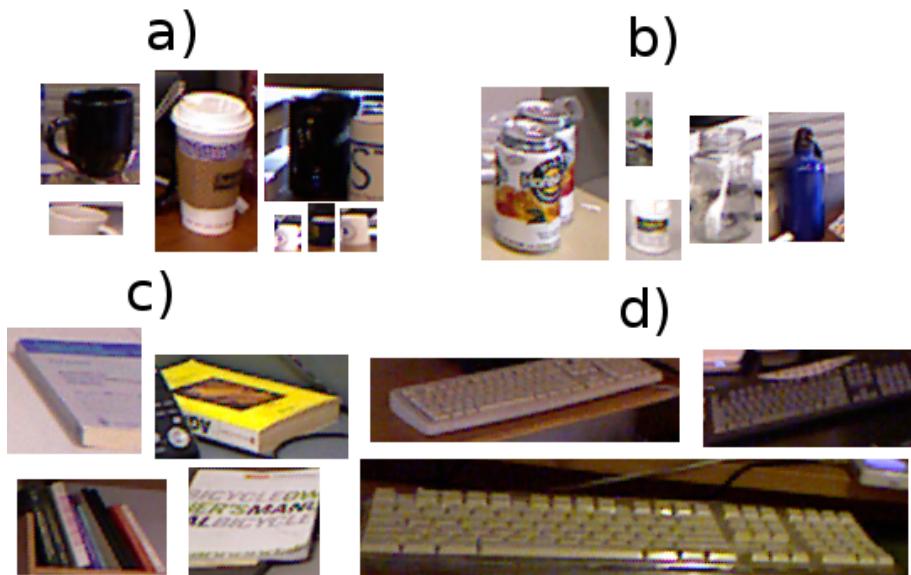


Figure 3.2: B3DO objects. Images are in their original sizes. a) cups b) bottles c) books d) keyboards

frequently marked as cups. Some of these misclassification errors might be caused by very poor quality of images. Many of them are occluded poorly lit low resolution images.

### 3.2.5 Tokyo

Every of the 343 images from Tokyo dataset was used. Data was split into test and train set in proportions 1:2 in order to provide more training examples due to low number of objects in some classes.

Even tough the overall accuracy achieved on Tokyo dataset is similar in value to that of the B3DO, the structure of the result is very different. It is clearly visible that there is

Table 3.3: Tokyo confusion matrix with ISS keypoint detector, PFH features and a dictionary of 3000 words

Assigned to category \\ Category	box	cart	notebook	scene	scanner	basket	bucket	freezer	bicycle	printer	Entries per class	Averages [%]
box	<b>11</b>	0	2	0	0	0	0	0	0	0	13	84.62
cart	0	<b>0</b>	1	0	0	0	3	0	0	0	4	0.00
notebook	6	0	<b>3</b>	1	0	0	1	0	1	0	12	25.00
scene	0	2	2	<b>4</b>	0	0	1	0	1	0	10	40.00
scanner	2	0	0	0	<b>0</b>	0	0	0	0	0	2	0.00
basket	1	0	0	1	0	<b>2</b>	7	1	1	0	13	15.38
bucket	3	0	0	0	0	0	<b>22</b>	1	0	0	26	84.62
freezer	0	0	2	0	0	0	1	<b>1</b>	0	1	5	20.00
bicycle	1	0	0	0	0	0	1	0	<b>33</b>	0	35	94.29
printer	0	0	0	0	0	0	1	0	1	<b>0</b>	2	0.00
<b>Average</b>											<b>62.30</b>	

a strong correlation between a per class accuracy and the number of entries in this class. The highest performance in the bicycle category is coupled with the largest number of entries. On the other end of the scale there are cart and printer categories with only 2 and 4 entries respectively. On top of that there are differences between some classes are marginal. The majority of carts and baskets were put into the bucket category. It does not surprise, for they are simply akin as can be seen in figure 1.5 on the page 19.

### 3.3 Conclusion

# List of Figures

1.1	Bag of Words histogram . . . . .	8
1.2	Bag of Words pipeline. The figure comes from [23] . . . . .	9
1.3	Classification by a Support Vector Machine. The distance of the separating plane from both datasets is maximal. The figure was originally published in [15] . . . . .	15
1.4	Berkeley 3D Object Dataset. The figure comes from [8] . . . . .	19
1.5	The Tokyo dataset: a) basket b) bicycle c) box d) bucket e) cart f) freezer g) notebook h) printer i) scanner j) scene . . . . .	19
2.1	Tagger3D class diagram . . . . .	24
2.2	Strategy pattern . . . . .	25
3.1	Influence of dictionary size on the overall accuracy. B3DO dataset with ISS keypoint detector and FPFH feature descriptor . . . . .	28
3.2	B3DO objects. Images are in their original sizes. a) cups b) bottles c) books d) keyboards . . . . .	29

# List of Tables

3.1	Highest accuracy obtained with FPFH, PFH and PFHRGB descriptors on the B3DO dataset . . . . .	27
3.2	Results on the B3DO dataset with ISS keypoint detector, FPFH features and a dictionary of 1500 words. <b>Overall accuracy is 65.22%</b> . . . . .	29
3.3	Tokyo confusion matrix with ISS keypoint detector, PFH features and a dictionary of 3000 words . . . . .	30

# Bibliography

- [1] L. A. Alexandre. 3d descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal*, 2012.
- [2] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [3] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, page 22, 2004.
- [4] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE, 2005.
- [5] S. Filipe and L. A. Alexandre. A comparative evaluation of 3d keypoint detectors.
- [6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: Abstraction and reuse of object-oriented design*. Springer, 1993.
- [7] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.
- [8] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. A category-level 3d object dataset: Putting the kinect to work. In *Consumer Depth Cameras for Computer Vision*, pages 141–165. Springer, 2013.
- [9] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgbd object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011.
- [10] X. Li and A. Godil. Investigating the bag-of-words method for 3d shape retrieval. *EURASIP Journal on Advances in Signal Processing*, 2010:5, 2010.

- [11] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.
- [12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
- [13] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*, pages 1–8. IEEE, 2007.
- [14] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *Computer Vision–ECCV 2010*, pages 143–156. Springer, 2010.
- [15] J. Ponce, D. Forsyth, E.-p. Willow, S. Antipolis-Méditerranée, R. d’activité RAwEB, L. Inria, and I. Alumni. Computer vision: a modern approach. *Computer*, 16:11, 2011.
- [16] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.
- [17] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.
- [18] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. Persistent point feature histograms for 3d point clouds. *Intelligent Autonomous Systems 10: Ias-10*, page 119, 2008.
- [19] S. Salti, F. Tombari, and L. D. Stefano. A performance evaluation of 3d keypoint detectors. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*, pages 236–243. IEEE, 2011.
- [20] S. M. Smith and J. M. Brady. Susan—a new approach to low level image processing. *International journal of computer vision*, 23(1):45–78, 1997.

- [21] J. W. Tangelder and R. C. Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia tools and applications*, 39(3):441–471, 2008.
- [22] R. Toldo, U. Castellani, and A. Fusiello. A bag of words approach for 3d object categorization. In *Computer Vision/Computer Graphics CollaborationTechniques*, pages 116–127. Springer, 2009.
- [23] C.-F. Tsai. Bag-of-words representation in image annotation: A review. *ISRN Artificial Intelligence*, 2012, 2012.
- [24] Q. Zhang, X. Song, X. Shao, R. Shibasaki, and H. Zhao. Category modeling from just a single labeling: Use depth information to guide the learning of 2d models. In *Computer Vision and Pattern Recognition, 2013. CVPR 2013. IEEE Computer Society Conference on*. IEEE, 2013.
- [25] Y. Zhong. Intrinsic shape signatures: A shape descriptor for 3d object recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 689–696. IEEE, 2009.