Object Recognition using the Kinect

FAVIO SAPONARA IRIARTE PANIAGUA

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2011

Object Recognition using the Kinect

F A V I O S A P O N A R A I R I A R T E P A N I A G U A

Master's Thesis in Computer Science (30 ECTS credits)
at the Degree Program in Information and Communication Technology
Royal Institute of Technology year 2011
Supervisor at CSC was John Folkesson
Examiner was Danica Kragic

TRITA-CSC-E 2011:115
ISRN-KTH/CSC/E--11/115--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

We show results on object recognition using the Kinect 3D camera. Several algorithms are tested along with automatic segmentation and model learning. The analysis of the environment is of great importance for autonomous robots. Depth of objects in the scene (is therefore) [then represents a] very useful data.

As 3D cameras previously existed only as very expensive devices, the research into algorithms for analysis of their data is limited. This has changed since the Kinect has been introduced into the market last year as a low priced 3D camera and now there is a sharp increase in research into such algorithms. The Robot Operating System (ROS) software the Point Cloud Library (PCL) has integrated the Kinect into several of its applications. On this library some state-of-the-art algorithms have been created. We will use both the ROS and the PCL library, evaluating the results and introducing some improvements and hints for future work. We will introduce also some state-of-the-art algorithms in order to apply the conversion from Computer-Aided Design (CAD) into Point Cloud Data (PCD) models which represent the format used by the PCL.

Acknowledgements

First of all I would like to express gratitude to my Swedish supervisor John Folkesson and my Italian supervisor Marina Indri for the guidance and help they gave me in terms of advice, reviewing, and improving this work.

A special thanks to all my friends I met, who have accompanied me in this wonderful journey of professional and personal growth started in Torino and ended in Stockholm. Thanks for supporting, putting up with me, and sharing with me unforgettable times.

Finally, I want to dedicate this thesis to my family, specifically to my parents Loida and Rocco, and my brother Luis. Thank you for supporting all my decisions and for your love. I hope I did everything possible so that you can be proud of me.

Contents

List of Figures

Acronyms and Abbreviations

CAD	Computer-Aided Design
CVFH	Clustered Viewpoint Feature Histograms
FPFH	Fast Point Feature Histograms
HSV	Hue Saturation Value
ICP	Iterative Closest Point
NARF	Normal Aligned Radial Feature
OBJ	Waveform Object
PCD	Point Cloud Data
PCL	Point Cloud Library
PFH	Point Feature Histograms
PLY	Polygon File Format
PNG	Portable Network Graph
RGB	Red Green Blue
RGBD	Red Green Blue Depth
ROS	Robot Operating System
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded Up Robust Features
VFH	Viewpoint Feature Histograms

Chapter 1

Introduction

This thesis project evaluates the use of two 3D object recognition feature descriptors with the use of the **Kinect** [2] and describes some improvements and hints for future work on them. Some new algorithms for the conversion of Computer-Aided Design *CAD* [3] into Point Cloud Data *PCD* [31] models are also introduced in order to have a wide range of models which can be used for the experimentations.

This master thesis represents the final project in my academic studies for the Master of Science in Computer Science Engineering, which has been conducted initially at Politecnico di Torino (Italy) and afterwards at Kungliga Tekniska Högskolan (KTH) in Stockholm (Sweden) as an exchange student in an Erasmus project.

The project has been carried out at the Computer Vision & Active Perception laboratory [4] which is part of the School of Computer Science and Communications (*CSC*) at KTH. This department has provided the necessary material for the development of the thesis.

The people I worked with during this project were:

- **John Folkesson** [5]: He has been the reference person at KTH. He is a Research Scientist working in the CAS department and his field involves the mobile robotics, with particularly attention for the navigation.

- **Marina Indri** [6]: She has been the reference person at my own University in Italy - Politecnito di Torino. She is an Associate Professor of Automatic Controls and Robotics and Automation in the Italian University. Industrial and mobile robotics are two of the fields involving her research.

- **Aitor Aldomà** and **Bastian Steder** [7]. They are the creators of the two descriptors I used during this work. Aldomà is currently doing his second year PhD at the Vienna University of Technology. His supervisor is Markus Vincze and his research interests are basically perception for robots including object recognition, pose estimation, grasping and modeling affordances. Steder is a PhD student in the Department of Computer Science of the Albert-Ludwigs-University of Freiburg. When some issues were found during the development

of the project, those researchers were really available in order to help the resolution of those problems.

1.1 Object recognition

The creation of mobile autonomous robots requires development in fields like control, estimation and localization, vision and programming. The main application of this project is object recognition within the mobile robotics field via. One of the challenges in mobile robots is that its environment is variable in terms of objects, colors and illumination. The ability of mobile robots, to recognize objects is of primary importance, because a robot has to be able to carry out tasks with the objects.

The object recognition consists in identifying an object in an image scene while using the object model is given as input. The recognition should be possible in conditions where the object has been rotated/translated or when different point of views of the object are considered.

The first step for doing object recognition consists in finding features from the image. A feature is defined as a piece of information deriving from a specific point of an image. The set of features of the global image defines the image itself. The feature descriptors are the algorithms which extract this information from the image. Depending on the feature descriptor used, the information is extracted in a different way. Then the information extracted is saved into vectors called feature vectors [1] or descriptors.

There are several methods for object recognition in a 2D dimension space where the information is extract taking into account the pixel color information (RGB). But, since the development of the 3D dimension techniques, we can add a new entry to the recognition in order to get better results, because we can immediately know the distance of the object from the robot without the need of elaborating data to get this new information like some object recognition methods do. This is essential if the robot needs, for example, to pick up or avoid objects. Furthermore the scalea is inherent to the distance from the objects, because it is possible to determine the size of the object just evaluating its relative size in an image by multiplying this value by the depth of the object from the camera.

The device used to acquire scenes with the depth information is the Kinect, which is a new revolutionary video-game tool integrated with the Microsoft XBOX 360. With this tool it is possible to capture 3D images plus depth. This last parameter is the most important information we need in order to use a 3D model in an object descriptor.

1.2 Object models to build and recognize

The models we worked with during the project were:

- A kitchen cup (figure 1.1(a)).

- The Kinect box (figure 1.1(b)).
- A small red-black box (figure 1.1(c)).
- The green cup present in the "KIT ObjectModels Web UI" [33] (figure 1.1(d)).
- The spray flask present in the "KIT ObjectModels Web UI" [33] (figure 1.1(e)).

(a) Kinect box

(b) Kinect box

(c) small red-black box

(d) Green Cup

(e) Spray flask

Figure 1.1. Object models used during the experimentations

We have created the 3D model representation of the kitchen cup, the Kinect box and the small red-black box objects. How this has been done will be explained in the Segmentation section. The 3D model representation of the green cup and the spray flask was already available [33]. The features descriptors have been used with those models and the object recognition results will be presented in the Experimentation section of this report.

1.3 Contributions

Contributions during the development of this project were done on different aspects. First of all segmentation algorithms had been introduced in order to create 3D Point Cloud Data (*PCD*) object models with methods that will be described in the next sections.

Available open-source code has been taken and integrated with new code in order to get improvements in the results. A "reverse" segmentation has been introduced in one of the descriptors used to get the object recognition. The aim of this segmentation is to remove planar surfaces on which the objects lie in order to remove irrelevant information from the scene. This will be discussed in section 5.1.2.

In addition testing of the Clustered Viewpoint Feature Histogram [19] descriptor algorithm, before its publication, has been done in collaboration with its creator, Aitor Aldomà.

A Simultaneous Localization and Mapping (*SLAM*) software has been used for the acquisition of the PCD model and some useful suggestions for better results will be given in the Discussion section of the Experimental Results section.

During the progress of this work a good knowledge of some libraries has been acquired and issues regarding the use and installation have been solved. A guide [8] describing the procedure of installation and execution has been done in order to reproduce the results and facilitate a future work.

I have also created an algorithm which is able to convert Waveform Object (*obj*) [9] into Point Cloud Data files maintaining the color information. This code has been sent to **Radu Bogdan Rusu** who is the main responsible of the Point Cloud Library(*PCL* [21]). He will then integrate this code into this library in order to add a tool able to do this conversion considering that a parser that makes the conversion maintaining the color information does not exist yet. Another algorithm for the conversion from Waveform Object to Polygon File Format(*ply*) [30] has also been implemented.

1.4 Outline

The thesis is organized as follows. After discussing related work in Chapter II, I will present the environment tools used, including the Kinect, in Chapter III. I will then describe in Chapter IV the segmentation algorithms created in order to reconstruct the model and the parsers created in order to convert models in different formats. In Chapter V I will present the Normal Aligned Radial Feature and the Clustered Viewpoint Feature Histogram descriptors with analysis of the code present in some available libraries. Subsequently I will give the description of the experimental results in Chapter VI and then some final conclusions in Chapter VII.

Chapter 2

Related work

Images contain interesting points which represent information about the scene. The way in which this information is extracted and how the description of those items is done distinguish the variety of available object recognition methods. Then these methods use the extracted data in order to identify the object in an image which contains the specific object.

A brief explanation is provided about some descriptors: the Scale Invariant Feature Transform (*SIFT*) [10], the Speeded Up Robust Features (*SURF*) [12], the Point Feature Histogram (*PFH*) [15], the Fast Point Feature Histogram (*FPFH*) [14] and the Viewpoint Feature Histogram (*VFH*) [18] descriptors.

2.1 SIFT and SURF

The Scale Invariant Feature Transform (*SIFT*), created by David Lowe in 1999, can be applied to different fields of computer vision such as object recognition, face recognition and 3D modeling. According to [10], these features are "invariant to image scaling, translation, and rotation, and partially invariant to illumination changes".

The main method used by SIFT to get the object recognition and described in [10] can be summarized into these steps:

1. Extraction of a set of key points from different images and store them into a database

2. The recognition of the object is done by comparing each feature of an image with the features of the image saved into the database . The main method used to compare features is to evaluate the Euclidean distance between the key points feature vectors which are saved in hash tables

3. When three or more matching features of an object are found with corresponding locations, further detailed verifications of the models are done

4. The probability of the recognition of an object, due to the presence of those features, is calculated considering the accuracy and the possibility of the existence of false positives

5. Key points that pass those tests then represent correspondences with objects memorized in the database.

In order to reduce errors caused by local variations, SIFT is able to detect and use a larger number of features from the images. An example of feature matching using SIFT is shown in figure 2.1.

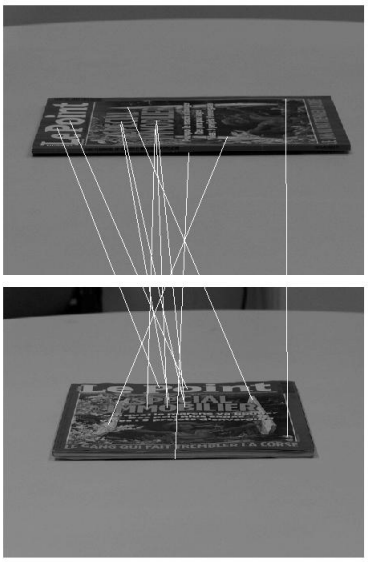


Figure 2.1. Example of a matching using the SIFT feature descriptor [11]

The Speeded Up Robust Features (*SURF*) has been presented by Herbert Bay et al.(2006) [12] . This method, like SIFT, is invariant to image scaling, translation and rotation, but it gains in performance because the computation and the comparison between features are faster and it is "more robust against different image transformation".

A public code for object recognition with SURF is available on a website source [13]. This can be summarized into the following steps:

1. Creation of an integral image, known also as *Summed area table* [35] which has been created by Franklin C. Crow. He defines this integral image like "an algorithm for quickly and efficiently generating the sum of values in a

rectangular subset of a grid". This different structure allows a fast object detection

2. Finding and extraction of interest points using the Fast-Hessian method

3. Initialize the SURF descriptor, compute the orientation and fill the descriptor for every interest point

4. Proceed with the matching comparison once an image is given in input.

2.2 Point Feature Histograms and Fast Point Feature Histograms

The Point Feature Histogram (*PFH*) is a descriptor created and described by Radu Bogdan Rusu in his PhD Thesis "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments" [15].

The main idea of this descriptor is to analyze the surrounding information of every point present in a Point Cloud Data structure in order to get a geometrical representation using a "Multidimensional histogram of values" which is based on the estimated surface normals of the point "k-neighborhood" as defined by the author. The **k** value indicates the number of neighbors which are situated in a position inside a specific radius **r** of a sphere around each point to be analyzed, and so which neighbors to consider during the normal estimations, as shown in figure 2.2.

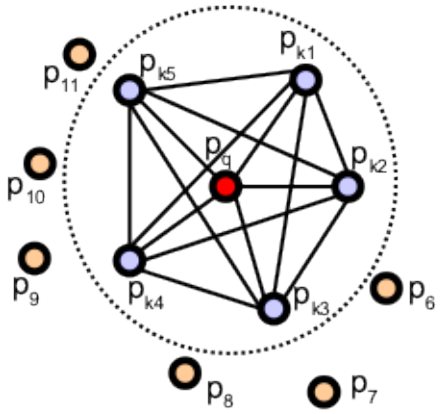


Figure 2.2. Neighbor range analysis using the Point Feature Histogram [16]

An implementation of this descriptor has been done using the Point Cloud Library [21]. In the Point Cloud Website [16] the steps to use this descriptor are summarized as follows:

- the first step consists in taking the closest neighbors of each point into the Point Cloud Data model

- after that, it is necessary to evaluate the euclidean distance and to compute the "three angular values" algorithm for each couple of neighbors

- results of this comparison need to be put into the output feature histogram.

The main problem of this algorithm is due to its computational complexity. With thick clouds of data around one point, real-time applications can not be built using this descriptor. This is the reason why Rusu et al. (2009) had improved this method creating the Fast Point Feature Histograms which has been described in their paper "Fast Point Feature Histograms (FPFH) for 3D Registration" [14].

This descriptor allows to lower the computation times in order to get an algorithm complexity suitable for real-time applications. Like PFH, this descriptor has been introduced into the Point Cloud Library and a tutorial for its use is available into the Point Cloud Website [22]. This improvement is realized thanks to the introduction of a "weighting scheme" between one point and its neighborhood. The first step consists in apply the PFH in order to get the estimations between a point and its k-neighborhood and so on for each point. After this step the weighting scheme is introduced by recalculating the estimations considering not only the points which are present into the k-neighborhood, but a wide range which allows to have "extra FPFH connections", as shown in figure 2.3.

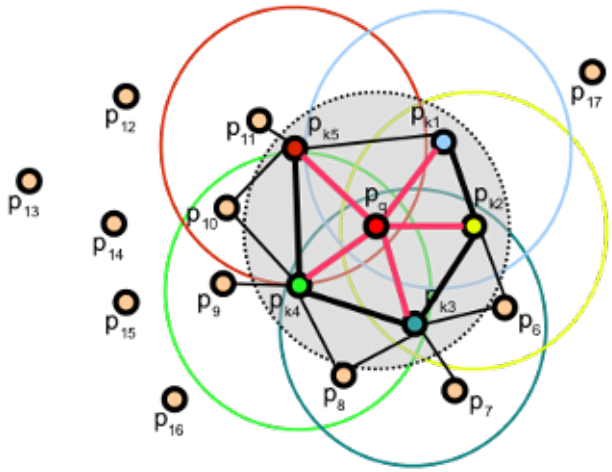


Figure 2.3. Neighbor range analysis using the Fast Point Feature Histogram [17]

2.3 VFH

The Viewpoint Feature Histogram (*VFH*) feature descriptor has been created by Radu Bogdan Rusu et al. (2010). According to its creators "focus is perception for mobile manipulation", that means this descriptor will give reliable results in terms

of object recognition, and the pose estimation of the models should be accurate enough because the robot has to know how to grab the object.

This descriptor works on a set of "clusters", where a cluster is defined as a "collection of 3D points" which can represent either an object or a scene. The main goal of the creators working on this descriptor was to get a set of "candidates" which could be the target objects which represents the input data. The pose position is also given because of the main purpose of grasping applications.

Two different steps involve the use of this algorithm: a "training" and a "testing" stage. In the first step a procedure is used to create a set of objects that will be recognized. From this set, in the second step, the candidates will be matched to the objects in the set.

The creation of the object set involves some phases according to [18]:

- The object should be placed on a structure which is easy to separate as an isolated cluster.
- Some methods are used to get the pose estimation for the object placed above the structure.
- This view is saved and the camera used to acquire the view or the object are rotated.
- This new view is saved into the dataset and so on.

The second stage instead, according to [18], involves algorithmic procedures that can be summarized as follows:

- Extraction of the clusters in a scene where the objects can be easily separated
- Considering the camera position, where the picture has been taken from, a VFH descriptor is "computed" for each cluster
- The VFH descriptor is then used in order to to match the clusters to objects in our dataset.

The training step is a significant difficulty. During the experimentations the creators of this descriptor have used a camera sensor called "PR2 Sensor Head" [23]. This is an expensive sensor, so the training phase implies high costs although the results are very accurate. This means that it is difficult to use this descriptor in order to do object recognition with own models although we have either CAD or PCD models.

Chapter 3

The Kinect and the work environment

In this chapter the tools used during this project are introduced. First an overview of the Kinect is presented, then the environmental work is described. After that, there is an explanation of the Robot Operating System (*ROS*), the Point Cloud Library (*PCL*), the RGBD 6D Simultaneous Localization and Mapping software and the Meshlab system.

3.1 The Kinect

The Kinect is a tool that provides full 3D images comprised of 640x480 RGB plus depth [27]. It is made of a RGB camera, an infra-red depth sensor, a multi-array microphone and a 3 axis (XYZ) accelerometer (see figure 3.1).

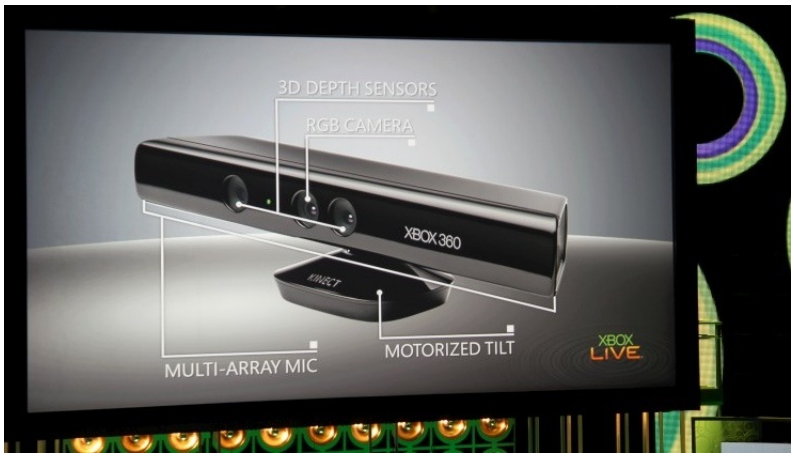


Figure 3.1. The Kinect

The main reason why this device is being used in computer vision projects all around the world is because of its relation between price and performance. Usually

devices that can provide the same information are much more expensive than the Kinect. The Kinect costs around $150 USD.

3.2 The work environment

The whole work has been performed in a MacBook Pro, 2.4GHz Intel Core 2 with 4GB of RAM although Ubuntu 10.04 has been the Operating System used during the project.

3.3 The Robot Operating System

3D perception is one of the most important parts in the robotics field, considering the availability of new hardware that researchers can use. This creates an increasingly need for software in order to work with this new hardware.

The Robot Operating System is an innovative open-source software that tries to follow hardware's footsteps which is daily being updated in most of its libraries (figure 3.2).

Figure 3.2. The ROS logo [28]

This is an Operating System that can be installed on a Robot. There are some experimental versions for other Operating System as OSX and Windows, but this software is not yet full-supported. This system, as described in its wiki page [28], is organized as a set of "nodes that are combined together into a graph". Node is defined as *a process that performs computation*. This is done in order to "reduce the code complexity and the fault tolerance" and to have an better overview of the whole system.

In order to install this software the simple use of "sudo apt-get install" is needed. Then to use the applications inside the software it is necessary to run commands. The most important are:

- **roscd**: With this command it is possible to navigate into the ROS packages. So this command is followed by the name of the package we want to visualize. If we type *roscd pcl* we move into the pcl package directory.
- **rosmake**: This command creates the binaries of the ROS tools from the source code available in the ROS system.
- **rosrun**: With this command it possible to run the binaries created in the step before. It needs as input the name of the package where the binary is stored and the name of the binary. For example *rosrun pcl pcd_viewer* runs the binary pcd_viewer which is situated in the pcl package.

- **roslaunch**: This command is able to execute *nodes* in ROS. For example, in order to launch the Kinect driver installed inside ROS, the follow command is executed: *roslaunch openni_camera openni_node.launch*. This launches the node **openni** inside the openni_camera package.

ROS contains the driver that allows the connections between the computer and the Kinect, and it contains the tools that were used in this project: The Point Cloud Library to recognize the object using 3D model descriptors, and the RGBD 6D SLAM software necessary to reconstruct the object model.

3.4 The Point Cloud Library

The Point Cloud Library (*PCL*, figure 3.3) [21] is a library developed byRusu et al. (2010) who define this library as "one of our most recent initiatives in the areas of point cloud perception" and they say that the goal of their work is to "provide support for all the common 3D building blocks that applications need".

During this project PCL has been full exploited. First of all because this library uses the Point Cloud Data file format (*PCD*) that contains the XYZRGB information. Moreover some 3D model object descriptors are already available in this library so they can be easily used in order to get the final result.

A lot of modifications and updates are involving this library everyday. *Pcl v.1* has been released only at the middle of May 2011 and its version has been defined *unstable* because of those continuos changes that should expand and improve the existent code.

During the experiments both this version and the previous one (*Pcl v. 0.9*) have been used. The previous version is the default version installed in ROS. Update completely the PCL version in ROS means to change a lot of the current ROS code. Obviously this is not suitable for now considering that *Pcl v.1* is unstable, as defined by its creators. This is the reason why the ROS creators did not overwrite the Pcl version in their repository.

Although they introduced the possibility to install the new version because the new tools that are being created in ROS are using the current version keeping in mind that old code is not going to work. This is done by overlying the previous Pcl package with the new one. I managed to achieve their coexistence in order to work with both of them.

Figure 3.3. The PCL logo [21]

PCL does not offer possibilities to work with CAD models, although we are going to use both CAD and PCD models in our experiments. In order to use CAD models, we will convert them into Point Cloud Data files with some methods and

then deploy PCL in order to work with those models too. A parser converter will be introduced and discussed in the next chapter.

PCD files, as said before, contain the information of the position in the space (XYZ) of the pixels in one scene. One constraint that we need to respect is to place the objects, we are interested in, at least 0.5 meters away from the Kinect. So pixels in the Kinect image scene which are closer are not rendered because they are not part of the working range of the Kinect.

The Point Cloud Library mailing list, available at *<pclusers@code.ros.org>* has been a useful tool in order to solve problems deriving from the use of this library. Developers were really available to help in the resolution of compiling and linking issues too. Then I would strongly recommend the use of this tool for future projects and work.

3.5 The RGBD 6D Simultaneous Localization And Mapping software

This software is a tool that has been the winner of a ROS contest on the February 2011. Then it has been improved in the middle of the April 2011 and this is the version of the software we used in this project.

Mainly RGBD 6D Simultaneous Localization And Mapping (*RGBD 6D SLAM*) allows to generate a colored 3D model of an environment. This software is based on the use of features descriptors like SURF or SIFT in order to "match pairs of acquired images" and then uses a "RANdom SAmple Consensus" (*RANSAC*) which is an "iterative method to estimate parameters of a mathematical model from a set of observed data" in order to "estimate the 3D transformation" between those acquired images. After those estimations a graph is built. Nodes of this graph are represented by the "camera views", the edges "correspond to the estimated 3D transformations". After that some mathematical optimizations are applied.

This software has been used to create the object model. In order to build this model, the object has been placed in the middle of an indoor environment, so the whole environment has been acquired and the object has been segmented with a method that will be described in section 4.2.

3.6 The Meshlab System

Meshlab [34] (figure 3.4), as defined in its wiki page, is an "advanced mesh processing system" which is able to manage different kinds of CAD models like Waveform object and Polygon File Format models. Removing the Point Cloud Data header, and saving the file into an *asc* format, using this open source tool, it is possible to visualize Point Cloud Data files. This can be done in order to have a graphical view of the model because in this way it is easy to evaluate the values and the distances where an object is situated using the Meshlab XYZ axis with values above.

Figure 3.4. The Meshlab logo [34]

This software then has been used to evaluate the position of the object we need to extract in order to create the 3D model. This tool is also able to change the format of CAD models, for example from an *obj* format into a *ply* format. The way in which this is done will be replicated into a C++ code and it will be improved in order to help future improvements. This is going to be discussed in sections 4.3 and 6.5.

Chapter 4

Segmentation and Parser Conversion

The first step consists of the creation of the object model that needs to be recognized with a 3D model object descriptor in the next step. This step is not necessary when we are going to consider the CAD models because we already have the files describing those models.

Two different segmentation approaches have been followed during the development of this project. The first one is a basic approach that has as output a single viewpoint of the whole model. In contrast the second approach, that we call complete segmentation, is able to extract the whole 3D model. These two approaches are going to be discussed in the next two sections.

As said before, CAD models are part of our experiments. In order to use those models a parser able to convert this kind of model into a Point Cloud Data model has been implemented. This is going to be discussed in section 4.3.

4.1 Basic Segmentation

This kind of segmentation is made by overlapping two different images captured by the Kinect: the background image and the image containing the object we want to segment. So once the background image has been acquired the object can be placed in the background and another image can be acquired. Then, considering that we know the background, we can easily extract the object from the global scene.

The basic idea of this process is to compare the depth information of the two images taking into account first the points which are the same or similar. This comparison is done by comparing the Z-value (or depth) of each pixel. So if they are equal (or at least similar) the point is not saved in the final PCD model file. This "similarity" can be evaluated by putting a threshold on the depth information between the two images.

We are aware that two different images of the same scene will not have exactly the same XYZ information due to the uncertainty of the Kinect device. But the XYZ information is quite similar, and it is why including a threshold into the segmentation algorithm, we can evaluate where the two images differ.

In this way we find where there is a considerable difference between the two images, and we can "extract" those points that correspond with the model.

Once the first process is computed we have a first model which includes some noise as result of pixels which have not been removed. So, after this initial processing, a reduction of noise is applied to improve the quality of the result. This is done by analyzing the neighboring pixels and eliminating the pixels which are not part of the main group because they have a few neighbors or they have no neighbors at all. For a better understanding of this method the pseudo-code has been written in Algorithm 1.

Algorithm 1 Basic segmentation pseudo-code

```
START main
Open input file BackgroundPcd
Open input file BackgroundObjectPcd

Read from BackgroundPcd
Read from BackgroundObjectPcd
for eachrow do
  diffDepth ⇐ Z_BgValue − Z_BgObjValue
  if diffDepth > depthThreshold then
    addPoint[numberPoints] ⇐ currentPoint
    numberPoints + +
  end if
end for
applyNoiseReduction(pointToValidate, numberPoints)
STOP main


START applyNoiseReduction
vector neighborpoints //contains the offset of the 8 neighbors of each point
totalNeighbors ⇐ 8
for i ⇐ 0 → numberPoints do
  countNeighbors ⇐ 0
  for j ⇐ 0 → totalNeighbors do
    if exists → pointToValidate ⇐ currentNeighbor then
      currentNeighbors + +
    end if
    if countNeighbor ⇐ thresholdNeighbors then
      insert → modelPoints ⇐ currentPoint
    end if
  end for
end for
STOP applyNoiseReduction
```

4.2 Complete segmentation

This second approach is based on using the RGBD 6D SLAM software described in section 3.5. The steps followed to reconstruct the 3D model are listed here after:

1. The object has been placed in the middle of an indoor environment.

2. The Kinect has been placed on the top of a shelf with wheels.

3. Then, maintaining a distance of at least 0.5 meters from the target object, the shelf has been moved around the object.

4. More or less, every 10 centimeters new data has been acquired with the Kinect using the software mentioned before.

5. Once we have moved the shelf all around the object, we can save the whole scene in a PCD file.

This file then has been processed according to two methods in order to extract the object model from the whole scene.

4.2.1 Extraction using the distance

The depth between the Kinect and the position where the target object has been placed is known a priori. Furthermore we know also the X and Y position with respect the Kinect camera.

With this information then it has been possible to remove most of the points which are not part of the target object. The result is not so accurate because in the vicinity of the object we have still some noise coming from the floor where the object has been placed.

4.2.2 Removing the noise using HSV and RGB thresholds

The distance gives us a very coarse result, so in order to obtain a better result the use of HSV and RGB thresholds has been introduced.

First the color of the kitchen floor has been taking into account. Considering that it is almost uniform it has been possible to remove those points which are in that range. Obviously the target objects should not contain those colors otherwise some information from the object itself will be removed too.

Then, if the object has a uniform color, we can improve the previous noise reduction by adding a second step in which we keep only the point information of those points which are in the object color range.

A pseudo-code of this implementation has been written in algorithm 2.

Algorithm 2 Complete segmentation pseudo-code

```
START main
Open input file wholeEnvironmentPcd
Open output file wholeModelPcd

Read from wholeEnvironmentPcd
for eachrow do
  //save x, y, z and rgb values in local variables
  stringstream(row) ⇒ x ⇒ y ⇒ z ⇒ rgbInteger;
  //convert rgbInteger into 3 different values: r, g, b
  rgb ⇐ *reinterpretCast < int* > (rgbInteger);
  r ⇐ (rgb ⇒ 16)
  g ⇐ (rgb ⇒ 8)
  b ⇐ (rgb)
  //positionThresholds, rgbThreshodls and hsvThresholds have been chosen
  //considering the distance where the object has been placed and its colors
  if x and y and z ∈ positionThresholds then
    if (r and g and b ∈ rgbThresholds) and (h and s and v ∈ hsvThresholds)
    then
      write → wholeModelPcd ⇐ currentLine
    end if
  end if
end for
STOP main
```

4.3 Parser Conversion

In this section we will descrive the *Parsers* we have created. We call them *Parsers* because they are able to read from a source file and create a file with a different organization, but similar content, by analyzing the structures of both files.

4.3.1 Obj to Pcd Parser

With this tool we can convert Waveform Object CAD models into Point Cloud Data models. The most important feature of this tool is that we can maintain the color information inside the Point Cloud Data file after the conversion from the *obj* file.

In algorithm 3 a pseudo-code of the tool has been implemented.

The code is structured in this way:

1. First of all two files are needed in input. The *.obj* CAD model and the image file (e.g. a Portable Network Graph (*png*) picture) describing the object model. An example of png is shown in figure 4.1. Wavefront objects and image files, which were used for the conversions, are part of the KIT ObjectModels Web UI [33].

Algorithm 3 Obj to Pcd converter pseudo-code

```
START main
Open input file fileNameModel.obj
Open input file FileNameModel.png

textureImage ⇐ cvLoadImage(FileNameModel.png);
Read from fileNameModel.obj
for eachrow do
    if Header ==′ v′ then
        v[numberVertices] ⇐ currentVertex
        numberVertices + +
    else if Header ==′ vt′ then
        v[numberTextureVer] ⇐ currentTextureVertex
        numberTextureVer + +
    else if Header ==′ f′ then
        for foreachTuple v/vt ∈ line do
            rgb ⇐ getTextureRGBValue → vt
            rgb[vertex] ⇐ rgb
        end for
    end if
end for
fileNameModel.pcd ⇐ v ⇐ rgb
STOP main
```

2. The image file is loaded into an OpenCV [32] structure. Instead the *.obj* file is loaded using the standard C++ syntax.

3. The *.obj* file, as described in Appendix A.1 contains the information about the vertices, the textured-vertices and the faces. The first two information are saved in two different structures because when the faces will be read we need to get the values inside the listed vertex-index values.

4. After this, for each vertex/textured-vertex tuple, the mapping between the textured-vertex and the image is analyzed, and as result the RGB data is given.

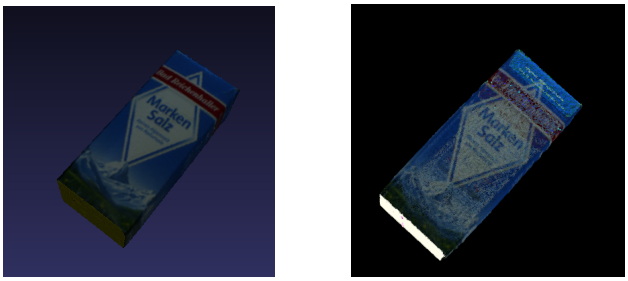
5. Then the 3 values (r,g,b) are converted in only one float value (as described in Appendix A.3 and then this value is saved in another structured whose index correspond to the current vertex position.

6. Then the Point Cloud Data is created in another file and it is saved with the Obj name, although the extension is changed into *.pcd*.

The main problem of this conversion was the mapping between the textured information and the image given in input which was achieved using the Bresenham's

Figure 4.1. Blue Salt Cube Image, available in the *KIT ObjectModels Web UI*

line algorithm. In order to get the conversion some internet material has been analyzed [24] and Andrea Bottino, who is a researcher at Politecnico di Torino, gave a good hint to use this bilinear interpolation and to get the desired result. Both Waveform object and PCD models can be seen in figure 4.2.



(a) Meshlab CAD model visualization

(b) PCL Point Cloud Data visualization

Figure 4.2. Visualization of both the CAD and the PCD models

4.3.2 Obj to Ply Parser

This tool introduces a new utility in the transition between two different CAD models: *obj* and *ply*. Software that is able to make this conversion is already available (e.g. Meshlab), but this conversion works in only one direction. *ply* files can have the information of the color in two different sections. It can be saved inside the vertex section or inside the face section. Software like Meshlab makes the conversion using the second way, but the first one is not implemented, or at least, it is hard to find a tool that converts and saves the information into the first section.

The second procedure has been created in our tool, that is able to save the information in both ways. A pseudo-code of the tool is available in algorithm 4.

The explanation of the algorithm can be summarized in the following steps.

- The code gets in input both the *.obj* and the image file, and requires to add an option that stands for the conversion to apply. 0 as option means to apply a color conversion *per face*, instead the option defined as 1 means a conversion per vertex.
- If the conversion is per vertex, the image is loaded and the Waveform Object file is read. Vertices and texture-vertices are saved into two structures, then the information from the faces is acquired and the color is calculated and saved in another structure. In this case the values r, g and b are stored separately inside the *ply* file.
- If the conversion is per face, instead, the image is not loaded, because the texture-color information is already saved as in the *.obj* file, that is to say, considering the values u and v that map the image information. So, the information is acquired from the obj file, and it is saved into the ply file, changing only the structure, in order to follow the ply syntax.

This parser has been mainly developed in order to introduce a color-factor into the Clustered Viewpoint Feature Histogram descriptor for future work and improvements.

Algorithm 4 Obj to Ply converter pseudo-code

START main
Open input file fileNameModel.obj
Open input file FileNameModel.png
Option: 0 - color information saved per face, 1 - color information saved per vertex
Read from fileNameModel.obj

if $option == 1$ **then**
  $textureImage \Leftarrow cvLoadImage(FileNameModel.png)$;
  for *eachrow* **do**
    if $Header ==' v'$ **then**
      $v[numberVertices] \Leftarrow currentVertex$
      $numberVertices++$
    else if $Header ==' vt'$ **then**
      $v[numberTextureVer] \Leftarrow currentTextureVertex$
      $numberTextureVer++$
    else if $Header ==' f'$ **then**
      for $foreachTuple\ v/vt \in line$ **do**
        $rgb \Leftarrow getTextureRGBValue \rightarrow vt$
        $rgb[vertex] \Leftarrow rgb$
      end for
    end if
  end for
  $fileNameModel.ply \Leftarrow v \Leftarrow rgb$
else if $option == 0$ **then**
  for *eachrow* **do**
    if $Header ==' v'$ **then**
      $fileNameModel.ply \Leftarrow currentPoint$
    else if $Header ==' vt'$ **then**
      $v[numberTextureVer] \Leftarrow currentTextureVertex$
      $numberTextureVer++$
    else if $Header ==' f'$ **then**
      for $foreachTuple\ v/vt \in line$ **do**
        $fileNameModel.ply \Leftarrow vertexPoint \Leftarrow textureVertexPoint \in vt$
      end for
    end if
  end for

end if
STOP main

Chapter 5

The Feature Descriptors

The most difficult task during object recognition is to find point correspondences between two images of the same scene or object. The feature descriptors approach this problem by selecting interest points from a scene or an object. Information on the vicinity around those key points is summarized in a descriptor which contains a set of values that allows the corresponding feature identification.

During this project we use two different descriptors that contain the features such as those required for the development of this thesis: the Normal Aligned Radial Feature ($NARF$) [26] and the Clustered Viewpoint Feature Histogram ($CVFH$) [19] descriptor.

5.1 The Normal aligned radial feature descriptor

The NARF descriptor, as described in [26], Steder et al. (2010), allows identifying key points and this information can be used for object recognition of a 3D model component. The NARF method is suitable for our purpose because this descriptor has been created with the intention of working with incomplete data depending on a particular viewpoint, and this is what the Kinect provides.

To find interest points with this method three requirements are needed:

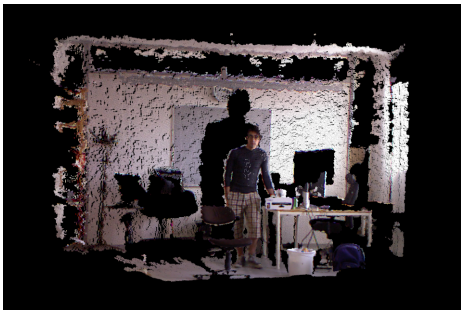
1. The surface in proximity of the point needs to be as "stable" as possible.

2. Around the point an adequate number of changes are needed in order to have information about borders.

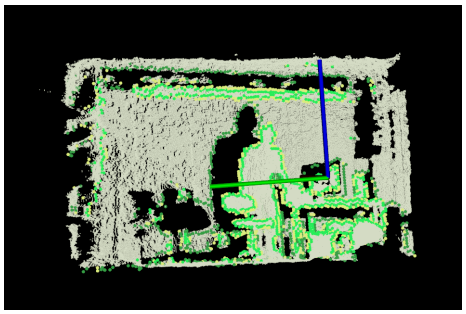
3. The selected points should not change even if the viewpoint changes.

The NARF key points depend on changes from foreground to background that can be easily evaluated with the 3D information. It is only necessary to compare the depth values of the pixels with its neighbors. If the difference between the distance exceeds a specific threshold the border has been found. Obviously this depends on the specific environment which is being observed.

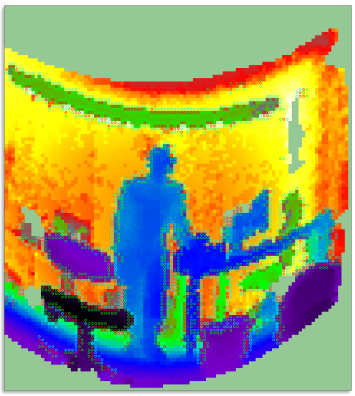
An example of border extraction using the NARF descriptor is shown in figure 5.1.



(a) Image captured by the Kinect



(b) Border extraction 3D visualization



(c) Border extraction color-depth 2D visualization

Figure 5.1. Example of the border extraction considering the depth values

The main approach of this descriptor can be divided into three points. First of all common features between the model and the scene are found in order to proceed with the matching task. Then some potential object poses from the matches are chosen. Then in the third phase those candidates that can correspond to suitable object matches are filtered by thresholds involving border and surface estimations.

During the first step, the descriptor is computed by saving the information of the vicinity of the key points that are found during the detection of the interest points. This last procedure involves mainly two methods. First of all for each point in the image a score is calculated considering the surface changes in the vicinity and the stability of the surface itself. Then only some points satisfying specific thresholds are chosen because they contain interesting information.

During the third step then multiple instances of the same object that had been found in the same position are removed and, after that, the Iterative Closest Point (*ICP*) [25] is apply in order to improve the object pose and get a 3D visualization of the object in the scene.

An implementation of object recognition has been done by one of the NARF creators, Bastian Steder [7]. During the course of this thesis, this implementation has been used and has been modified in order to improve the results which have been obtained by Steder. A pseudo-code has been created in order to allow an easy fast approach with the C++ code (see algorithm 5).

5.1.1 Description of the NARF object recognition code

In this section the code written by Steder and available in the Robot Operating System will be described. Some changes have been made to improve the matching results and this will be discussed in the next section.

When the code is called using the Robot Operating System (section 3.3), there is the possibility to set some options which are really important in order to get better results. Options that can be defined are:

- -v: "min validation point score". This is the main option that we can set and defines which is the minimum threshold that allows the recognition of the object. The default value is 0.4. If the object has not been found with this threshold, we can lower this value in order to get the recognition, although this can introduce the recognition of false positives in the scene.

- -m: "set unseen pixels to max range". Setting this option the borders of the objects are found even if the area around the object was unseen. Considering that NARF key-points depend on changes from foreground to background, if the area around an object is marked as unobserved those areas are totally ignored. In order to keep key-points on those areas, this option can be defined, in this way these areas are marked as "far ranges", so the interest points will be extracted.

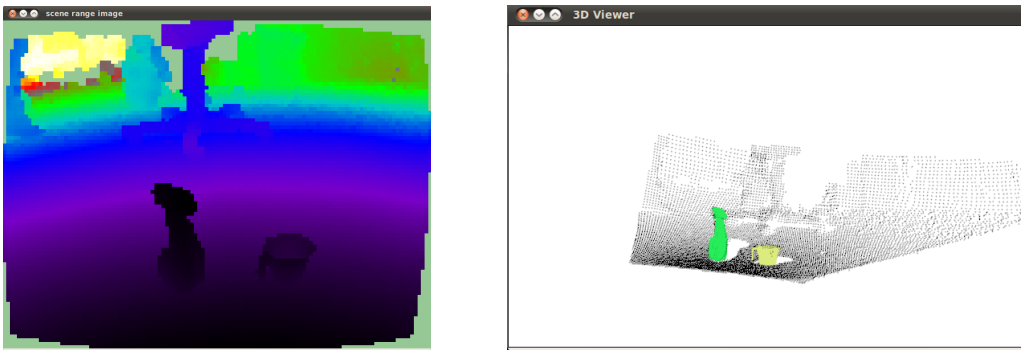
- -o: "use orientation invariant version". With this option the descriptor is invariant regarding the rotation around the normal. Without it for example, a top left corner will not be matched with a top right corner. Setting this option the search space gets bigger but this mean there might be more false positives.

- -b: "activate single view model". This option is useful when the model in input is not composed entirely by points, but only single views of it are available.

- -s: "scene used to create the range image". With this option it is possible to set the origin of the scene model. The default is 0, that means a PCD file. If The value is defined as 1, then the Kinect is chosen as scene input data, and this has to be done by using: "input:=/camera/depth/image input2:=/camera/depth/camera_info".

- -h: "help". Useful to visualize all the available options.

As said before, the input model can be originated in two different ways setting an option to distinguish which way is going to be applied. In both cases two different windows are launched to have a graphical view of the current range scene, which allow also a *visual* recognition of the object into the scene. The first window (see figure 5.2(a)) is called "scene range image" and it visualizes the scene in a 2D color depth dimension. The second window (see figure 5.2(b)) instead is called "3D viewer" and it visualizes the scene in a 3D pointed structure, and, if the model is found in the scene, it is colored in a random way.



(a) The color depth image viewer (b) The 3D viewer with colored models

Figure 5.2. Viewers using the NARF descriptor code

The number of models that are given in input is arbitrary. What we need to consider is that the size of those objects should be similar. This is due to the fact that the radius of the support size value, important to define the search wide for each interesting point found, is defined taking into account just the first model given in input.

After the models have been acquired, key points are extracting using the NARF descriptor. Then key points are extracted also for the scene image and they are compared in order to match the objects.

5.1.2 Method to improve the results

Optimizations of the NARF descriptor had been done by taking into account that objects we are considering during our experiments lie usually on planar surfaces like a table or the floor. So, when the data is acquired by the Kinect, before using the NARF descriptor for the matching of the object, we remove the planar surface, in order to delete points in the scene that do not contain interesting information.

This integration is done by using a code written by Radu Bodgan Rusu and available in the Point Cloud Library. This code is able to segment planar objects and then gives as result only the segmented surface removing all the other information. In figure 5.3 an image captured by the Kinect is shown. This image contains a table with some objects on it. A result of the previous surface segmentation can be seen in figure 5.4(a).

Algorithm 5 Complete segmentation pseudo-code

START main
Open input file(s) ModelPcd
Get Scene input from ScenePcdFile or from the Kinect
Get the list of the options and set them
$supportSize \Leftarrow firstModel.getRadius()$
$NarfKeyPointinterestPointDetectorModel(borderExtractorModel)$
$objectModels.extractNARFsForInterestPoints$
//compare the main threshold with the matching border score
if $minScoreValidation > borderScore$ **then**
  $continue$
end if
//compare the main threshold with the matching surface score
if $minScoreValidation > surfaceScore$ **then**
  $continue$
end if
$addModelMatching$
$drawModelInSceneVisualization$
STOP main

Our purpose is instead to remove the surface information where the objects lie. In order to get this different type of segmentation we have created a code in the main object recognition C++ algorithm described above.

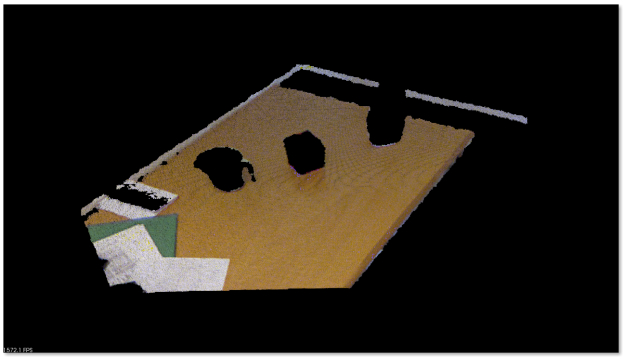
Figure 5.3. Sample image with some objects on the table

The main changes introduced in Steder's code regarded the introduction of an input option in order to decide if the input scene will use the planar segmentation or not. The option introduced has been called -*t* and we will refer to this option as "reverse planar segmentation" .

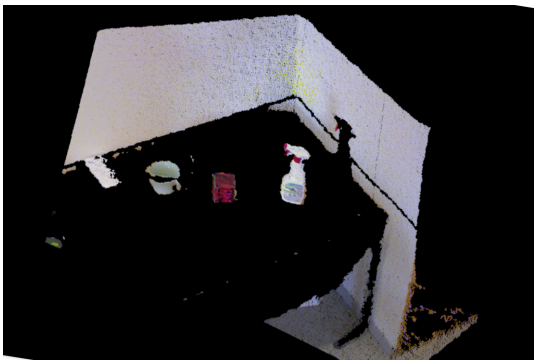
A result of our remotion of the planar surface can be seen in figure 5.4(b) where

the planar surface is represented by a table.

Results of the matching improvements will be shown in section 6.5.



(a) Image containing only the table



(b) Image where the table has been removed

Figure 5.4. Different planar surface segmentation

5.2 Clustered Viewpoint Feature Histogram descriptor

This descriptor has been used in order to recognize CAD models available in the KIT Web UI [33]. The main idea of its creator, Aitor Aldomà et al.(2011) takes into consideration the properties of the viewpoints over the object surfaces.

CVFH works better than the Viewpoint Feature Histogram feature descriptor (*VFH*) when the training data sources and the recognition data source are different. So in order to train the classifier only the CAD 3D models are needed. The concept of the descriptor is similar to the VFH, but according to its author this descriptor has been improved by the use of the CVFH considering the execution and processing time and the results.

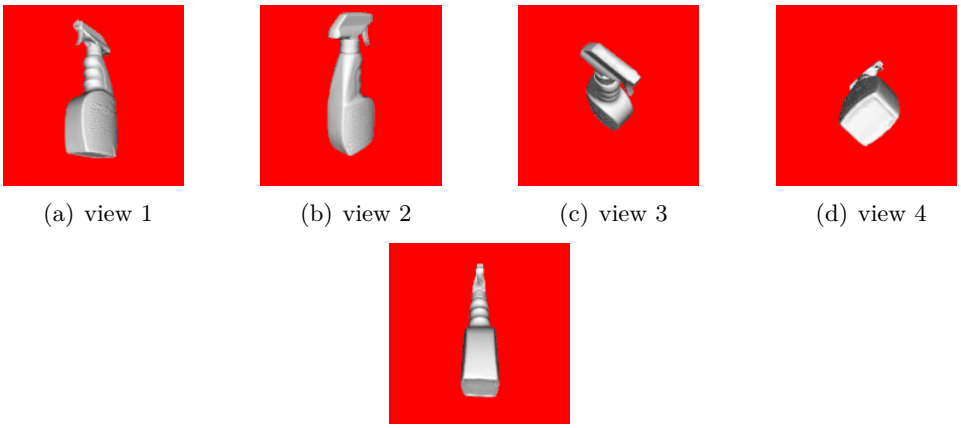
The C++ code of this object recognition is also available in the ROS package since the beginning of May when it has been released by its creator, Aitor Aldomà.

The explanation of this descriptor is given by considering this available ROS code because the paper has not been released yet. The main approach of this descriptor will be discuss in the next section.

5.2.1 Description of the CVFH object recognition code

There are different steps involving the object recognition using this descriptor. The first step consists of a training process of the input data as in the VFH descriptor described in section 2.3. The difference is that this descriptor needs as input CAD models (*ply* format) than will be analyzed and converted in a set of PCD files and there is no need of a particular camera sensor in order to acquire models as explained in the VFH description section.

Each of the PCD file, which has been created, contains only a view of the global input model. An example of this conversion is shown in figure 5.5 where 5 of the spray flask model viewpoints are shown. A Meshlab view of the whole spray flask model, which is available in [33], is instead shown in figure 5.6.



(a) view 1 (b) view 2 (c) view 3 (d) view 4

(e) view 5

Figure 5.5. 5 different views of the total 20 for the spray flask model

Once the PCD view models have been created and the classifier has been trained, the second step consists of the object recognition matching. In order to achieve this goal we can summarize the code as follows:

- Computation of the cluster through a segmentation procedure which substantially filters the scene with some bounds on the depth and performs segmentation extracting the planar information in order to consider objects the elements situated only on planar surfaces.

- Some parameters are defined in order to distinguish different clusters like a minimum number of points per cluster and a minimum distance between candidate objects which can stay in the same cluster.

- Once the clusters have been calculated the matching step is computed in order to find similar surfaces of the input dataset into the clusters which have been calculated.

A limitation of this feature descriptor is that it only uses surface properties. This leads to false positives for objects that have parts with similar surfaces as parts of our dataset objects. This would require using more information such as colour to disambiguate. We also need to consider that, in order to use this descriptor, the objects should lie on planar surfaces like tables or floors.

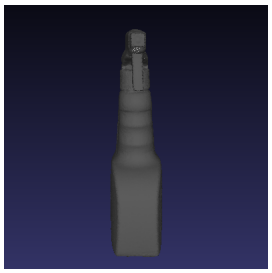


Figure 5.6. Whole spray flask CAD model - Meshlab view

Chapter 6

Experimental Results

In this chapter some experimental results of this project are presented.Results regarding the use of the segmentation algorithms (section 6.1), the NARF (section 6.2) and the CSVF (section 6.3) descriptors will be shown.

6.1 Segmentation Results

Two procedures have been presented in order to segment the models. In the next two sections some results with relative image documentation are presented.

6.1.1 Basic segmentation

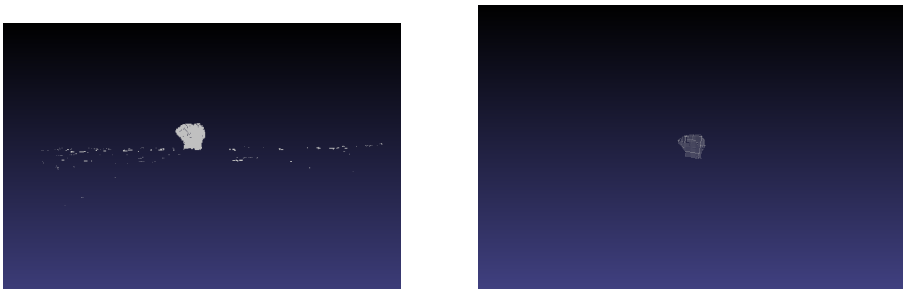
The kitchen cup has been used to show the results of this type of segmentation. As explained in section 4.1, this segmentation is based on image comparison and involves substantially the main zone of the images where they differ. After this comparison a noise removal algorithm has been applied.

In figure 6.1(a) is shown the image without the object model. Instead figure 6.1(b) shows the model inside the scene image.

(a) The background (b) The background with the object

Figure 6.1. Example images used in the basic segmentation

After the first comparison the result we have got can be seen in figure 6.2(a) , and then a view of the model without noise is shown in figure 6.2(b).



(a) The object before the noise removal function

(b) One viewpoint of the object after the noise removal function

Figure 6.2. Noise removal function

6.1.2 Complete segmentation

To demonstrate this method, we used three different models: the kitchen cup, the Kinect box and the small red-black box. In figure 6.3 there is one example about the phase of reconstruction of the Kinect box model using the RGBD 6D SLAM ROS software.

Those models are going to be part of the results we are going to show using the NARF descriptor.

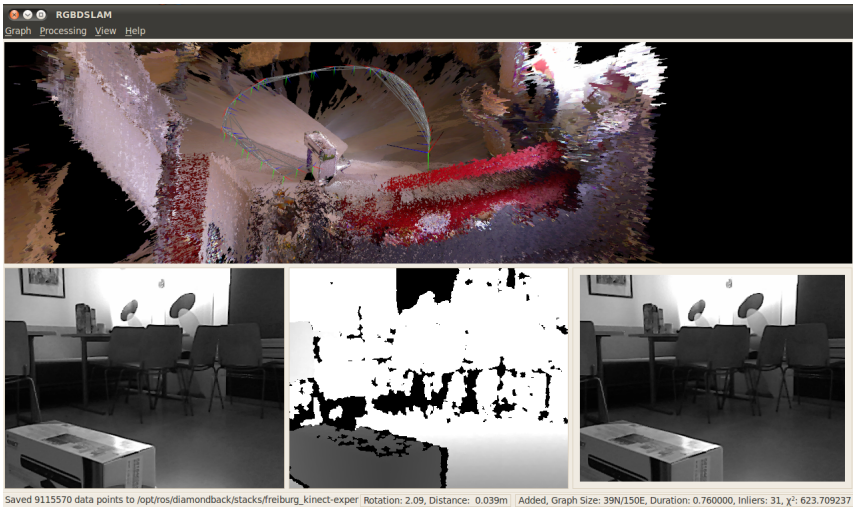
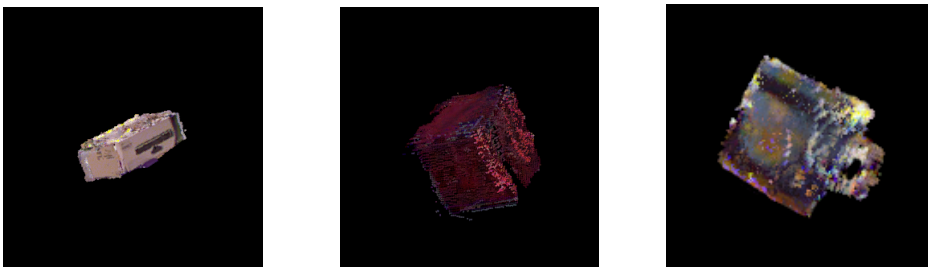


Figure 6.3. The whole Kinect box model inside the kitchen environment

6.2 Object recognition with NARF descriptor - Results

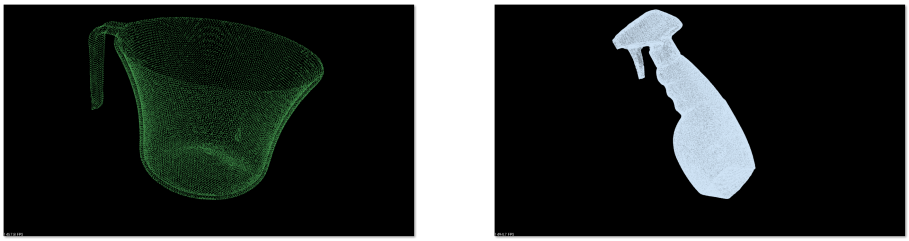
How this descriptor works has been presented in section 5.1. In this section we want to focus on the results of the recognition using the NARF descriptor.This descriptor is already used in the PCL libraries to recognize objects. The main problem was obtaining the model, but we solved that problem using our segmentation algorithm. So experiments will refer both to the models we have acquired in the previous section (the kitchen cup (6.4(c)), the Kinect box(6.4(a)) and the small red-black box (6.4(b))) and to the CAD models converted into PCD files (the green cup (6.5(a)) and the spray flask (6.5(b))).

Those CAD models have been converted into PCD files by a simple parser code whose implementation has been reproduced in a pseudo-code (see algorithm 6). This is a simple conversion because it does not consider the color information stored inside the CAD models because the current NARF descriptor code does not consider this data.



(a) The Kinect box (b) The small black-red box (c) The kitchen cup

Figure 6.4. 3D model views



(a) The Green Cup (b) The Spray Flask

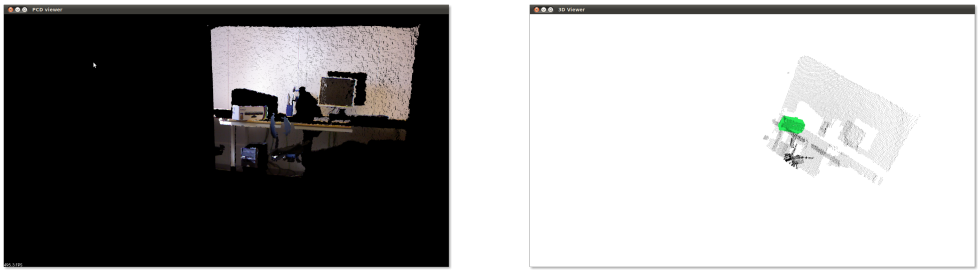
Figure 6.5. CAD models transformed in full PCD 3D models

Then, moving into the recognition part, figure 6.6(a) shows a scene acquired with the Kinect with one of the previous mentioned objects, and in figure 6.6(b) it is possible to observe the image recognized by coloring the part where the object is situated.

Algorithm 6 Simple parser converter from obj to pcd pseudo-code

```
START main
Open input file modelObj
Open output file modelPcd
READ modelObj
for eachline do
  if line[0] ==' v' then
    write → modelPcd ⇐ line[2 → end]
  end if
end for
STOP main
```

These models had been given as input to the object function described before. In figure 6.6(b) it is possible to see a 3D environment where the models, which have been found in the scene, are painted by the visualizer. The images are captured by the Kinect, with a frequency of 1Hz.



(a) One scene from the Kinect

(b) Object successfully recognized

Figure 6.6. Kinect box recognition

We considered a set of 4 different scenarios where the objects have been placed. For each scenario 2 different methods have been applied with different parameters. We will refer to those methods as method A and method B. The analysis has been done in the following way:

- Once an object, which is given as input to the NARF code descriptor, is recognized, in order to verify the matching, further acquisitions are done
- Those acquisitions are done close to the initial matching with a different point of view
- By using method A, each scene acquired can give a matching recognition or a false positive matching of the objects in the scene.
- By using method B the object has to be recognized for a total of 4 times in a row of 5 scene acquisitions. If the object which passes those verifications is

not the real object is considered like a false positive. Instead if the object is present in the scene but it is not considered for a total of 3 times in a row it is considered like a failure matching.

6.2.1 Recognition of the kitchen cup in a simple scene

The first scenario that is analyzed consists of a table where the kitchen cup and the spray flask lie (see figure 6.7). The object recognition is performed on the kitchen cup.

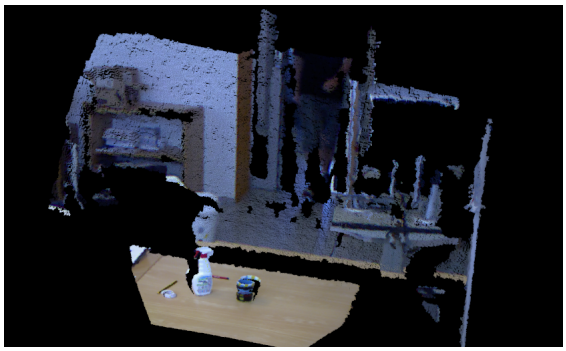
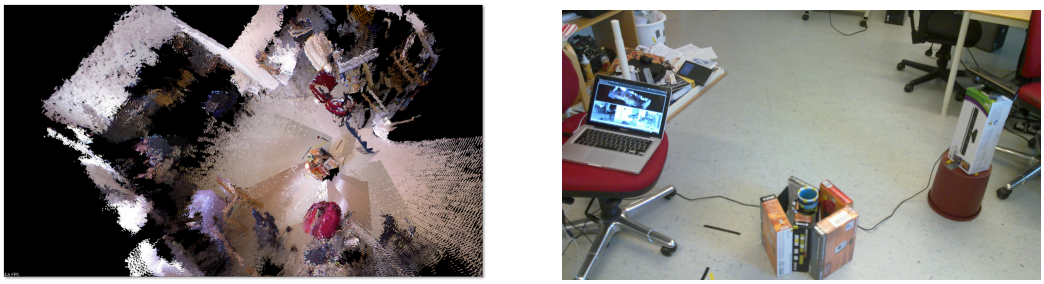


Figure 6.7. Simple scenario

As said before the cup itself did not exist as a 3D model, so the first step consisted in creating that model and then recognizing it. The model of the cup was created using the tool RGBD-SLAM which has been presented in section 3.5. After some experimentation using RGDB-SLAM, we were able to define the best conditions to build a 3D model using this software. We present these results in the Discussion section of this chapter.

The cup has been placed in the middle of the laboratory where the project has been developed. By using the RGBD-SLAM we could reconstruct the laboratory model with the cup inside (see figure 6.8).

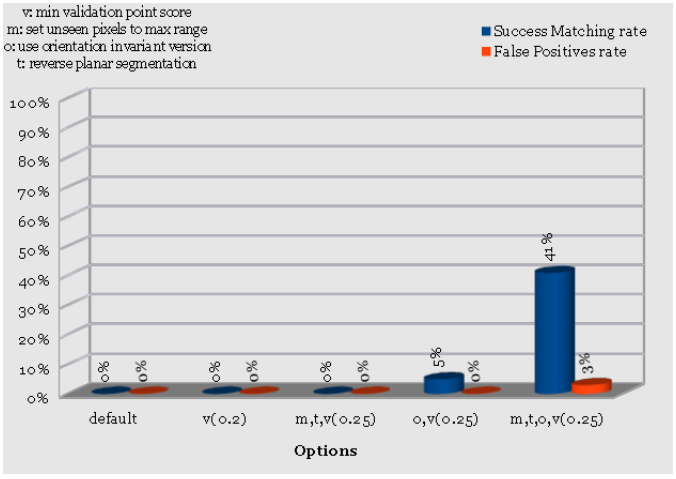


(a) 3D full environment

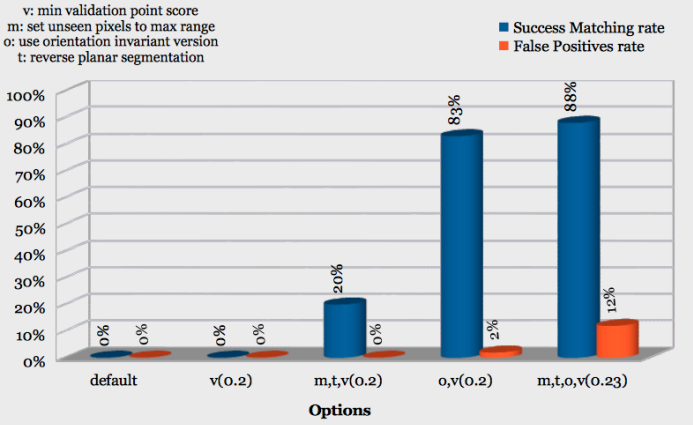
(b) Capture procedure

Figure 6.8. Creation of the 3D cup model

Then the next step regarded the segmentation of the cup using the procedure which has been already discussed in section 4.2. Once the model was acquired it needed to be recognized. In order to do some experiments in this scenario the Kinect streaming mode has been used. The results have been taken looking at the output of the 3D viewer. Different options have been used in order to evaluate different matching scores and to find the best solution in this scenario. Results and options by using boht method *A* and *B* are shown in figure 6.9.



(a) Recognition results by using method A



(b) Recognition results by using method B

Figure 6.9. Kitchen cup recognition in a simple scene

The option "reverse planar segmentation" introduced by the author of this thesis together with the option "set unseen pixels to max range" increased the object recognition matching when the "min validation score" has been lowered. We need to use the option *-m* due to the fact the surface in proximity of the point needs to be as "stable" as possible. We remove information which is irrelevant because

does not contain the object, but it could help to find more interesting points into the objects. This has been taking into account since the beginning because we can force the descriptor in order to find more key-points where there are not so much information as explained in section 5.1.

By using all the three options with a lower min_validation_score value than the default value, we obtained a good result in the object recognition matching, although this increased also the number of false positives which have been found in the scene.

6.2.2 Recognition of the Kinect box

In this scenario the Kinect box, the spray flask, the green cup, a wallet, a tape, the kitchen cup and the small red-black box lie on the table (see figure 6.10). We have acquired the Kinect box model as explained in the previous section for the cup.

Figure 6.10. Scenario for the Kinect box recognition

The results and options by using method A are shown in figure 6.11.

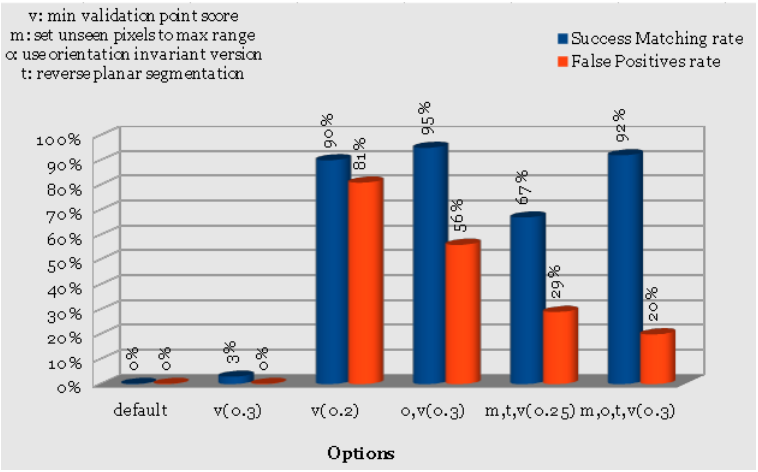
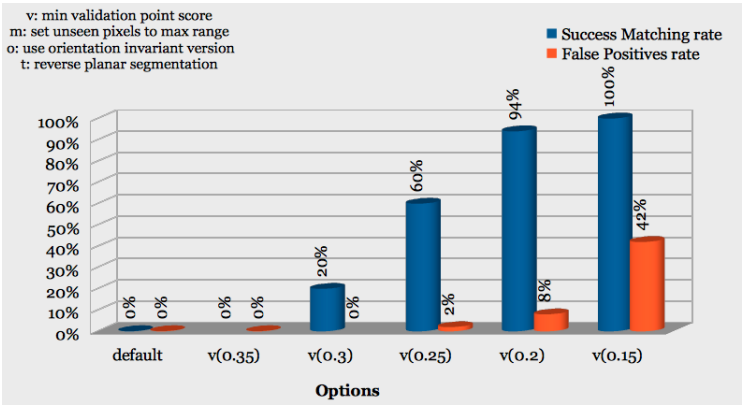


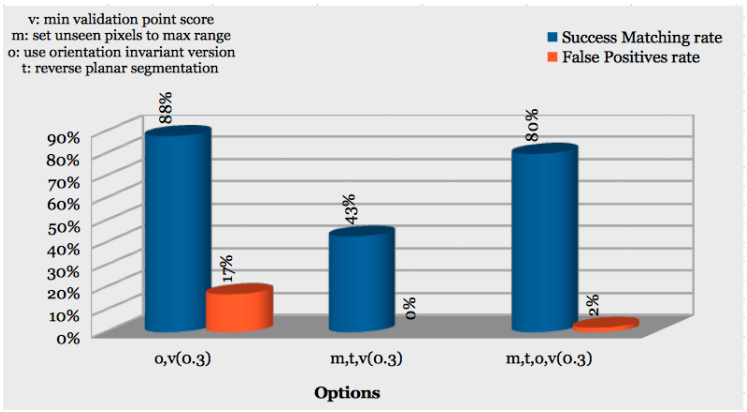
Figure 6.11. Kinect box recognition results - Method A

For method B we show two different approaches used during the experiments. The first one considers the variation of the main threshold using the option "min

validation point score". We can lower the threshold until we obtain a really perfect accuracy but this introduces a high number of false positives. By setting the v option we obtained the results shown in figure 6.12(a). The second approach consists in keeping the main threshold to 0.3 and combine the other options (figure 6.12(b)).



(a) Recognition results by setting the v option



(b) Recognition results by keeping the v option

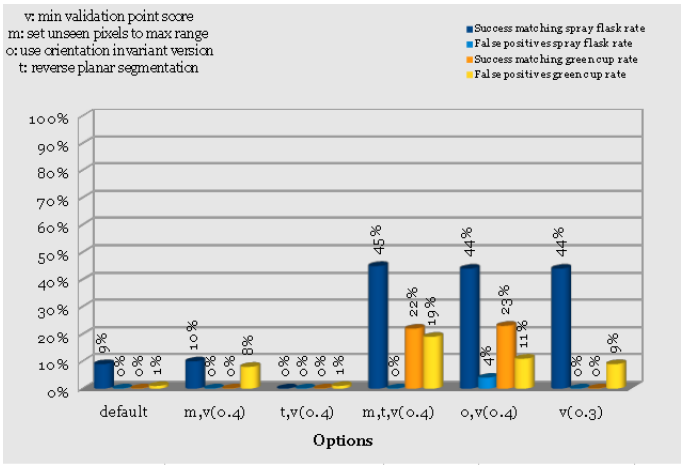
Figure 6.12. Kitchen cup recognition in a simple scene

Comparing the only use of the main threshold with combining this value with the other options, the results we obtain are better without the need to lower the main threshold too much like in the first approach in order to get high matching scores.

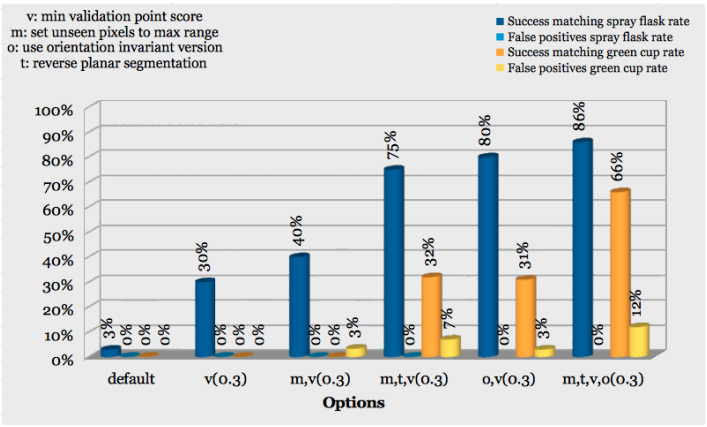
In order to evaluate those results we need to consider the Kinect position where the scene is acquired. Taking into account the shape of the box, if the Kinect captures more than one face of the box, this means that more interesting points are found and the matching score is higher than scenes where only one face is available.

6.2.3 Recognition of two objects

We used the green cup and the spray flask for the second experimentation using this descriptor. These objects have been placed on the laboratory floor where the work has been done. This has been done in order to test the planar segmentation introduced into the main code also in this conditions. Figure 6.13 shows results and options used during these tests by adopting both methods presented above.



(a) Recognition results by using method A



(b) Recognition results by using method B

Figure 6.13. Green cup and spray flask recognition results

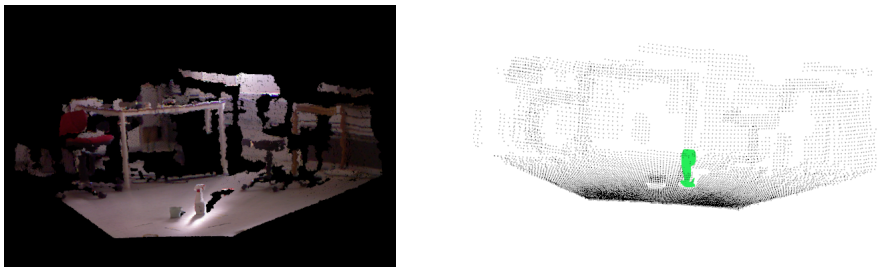
First consideration to do is about the order in which the models are put in input into the object recognizer code. As said before, the value of the size of the sphere, where to evaluate the neighbor area, is computed by considering the support_size. In Steder's code, this value is only equal to the support_size of the first model which is given in input, but as we have two different object with two different support_size values (see table 6.1), we need to put in input the object whose support_size is

Object	**Support Size**
Green cup	0.0393835
Spray flask	0.0634842

Table 6.1. Green cup and spray flask support size values

higher in order to have evaluated all the points of its size. In this way we will evaluate also the object which has a smaller support_size considering a big range. This could introduce more false positives, because increasing the research area it is possible to find more false positive matching.

Although the objects have been found, the pose estimation has not brought improvements to the initial pose which has been found, and which differs from the real pose. In figure 6.14(a) the 3D visualization of a test scene is shown, instead the wrong pose estimation can be seen in figure 6.14(b)).



(a) 3D test image visualization (b) Pose estimation of the object found

Figure 6.14. Wrong pose estimation

By lowering the main threshold (min_validation_point_score) from the default value (0.4) to 0.3, the rate of object recognition success has a good result when we combine this value with the three other options. The problem is that setting this value, more than one instance can been found in the same position during the object recognition process (see figure 6.15). This can be easily solved by considering only the value with the higher match score if we know that only one instance of the object is present into the scene.

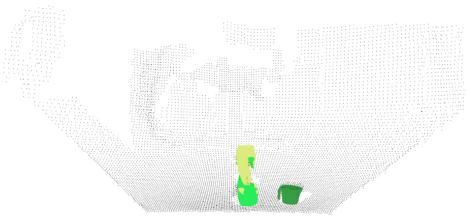


Figure 6.15. More instances of the same object

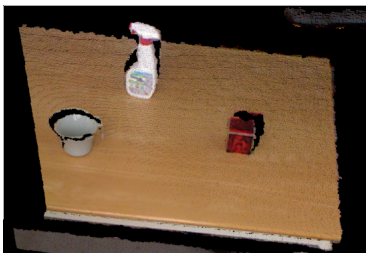
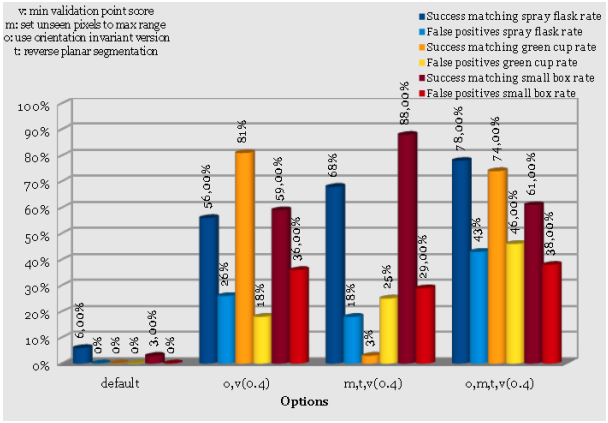
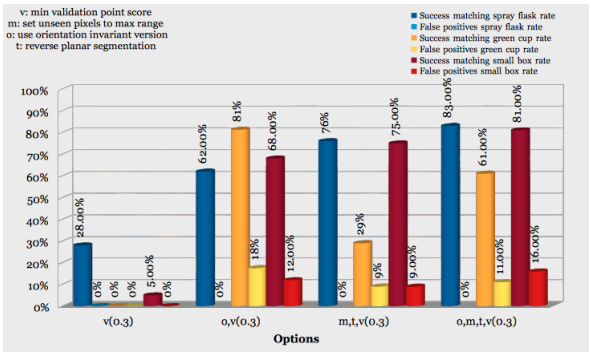


Figure 6.16. Scenario for the spray flask, Green cup and small red-black box



(a) Recognition results by using method A



(b) Recognition results by using method B

Figure 6.17. Green cup,spray flask and small red-black box recognition results

6.2.4 Recognition of three objects

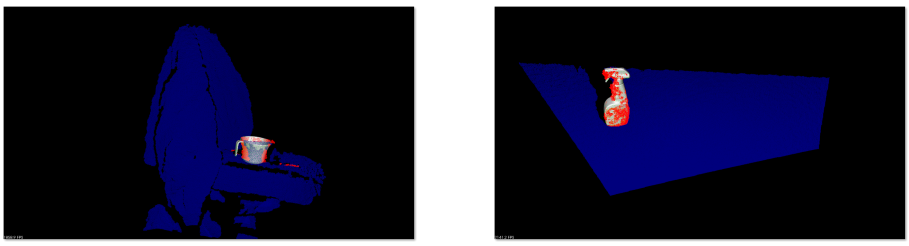
We then did experiments with three different models given as input using this descriptor and evaluate those results. The spray flask, the green cup and the small red-box have been put on a table as shown in figure 6.16. Results and options used are shown in figure 6.17.

Results show a higher matching for all of three objects when using all the three options (*-m, -t, -o*) lowering the main threshold to 0.3. One evaluation that we can do is about this last parameter which represents the most important value when we work with this descriptor because by changing its value we are able to increase significantly the number of object matching introducing however a high number of false positives.

6.3 Object recognition with CVFH - Results

In this section some results using the CVFH descriptor will be shown.

As we have seen in section 5.2 where CVFH has been explained, this descriptor has been used in order to use the available CAD models from the KIT on-line database [33]. Two of those models have been used for the experiments using this 3D descriptor: the green cup and the spray flask.



(a) The Green Cup (b) The Spray Flask

Figure 6.18. CAD models recognized into the scene

We are going to use the main object recognition code inside the ROS package called "vfh_recognizer_test" and make some considerations about future improvements in the discussion section. Once objects are going to be recognized, we will get images like the ones shown in figure 6.18. Considering that this algorithm matches the objects using only the surface properties, it will be easy to have some false positives in the scene. An example of false positive can be seen in figure 6.19.

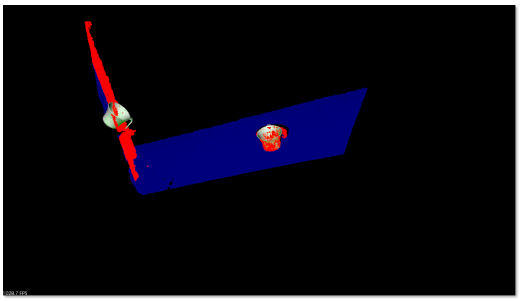
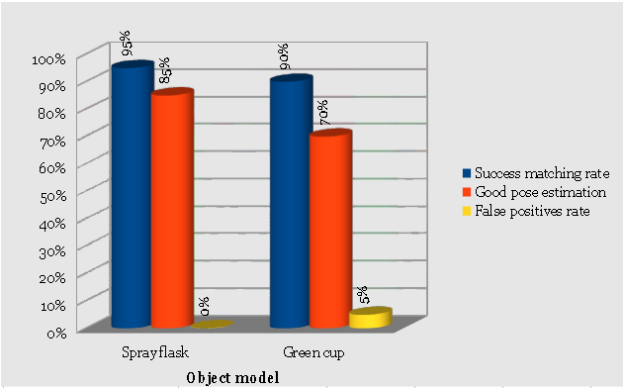
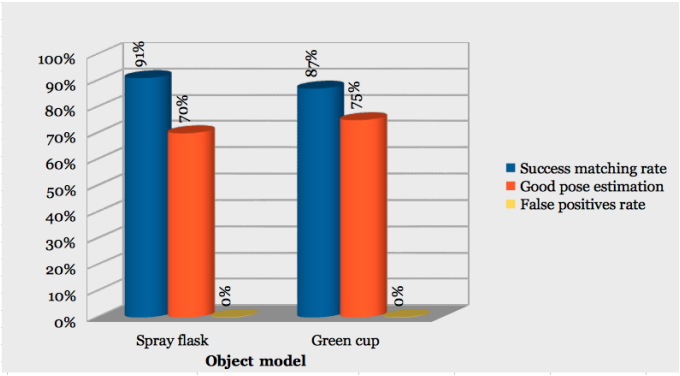


Figure 6.19. An example of false positive detected into the scene using CVFH

Figure 6.20. Spray flask and green cup table scenario



(a) Recognition results by using method A



(b) Recognition results by using method B

Figure 6.21. Green cup and spray flask recognition results in a simple scenario

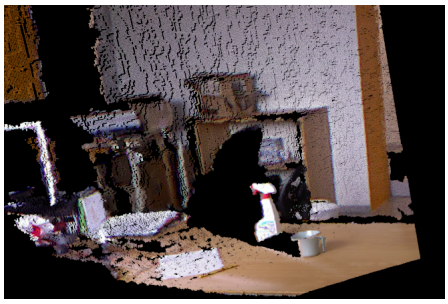
To test the matching rate for this descriptor we have done experimentations in two different scenarios. The acquisition scene procedure was the same used during the NARF descriptor experiments. In the first scenario, we placed only the green cup and the spray flask on an office table (see figure 6.20) removing all the other objects.

In figure 6.21 results of the object recognition in this condition are shown. The distance from the center of the table, when those scenes were acquired, was around

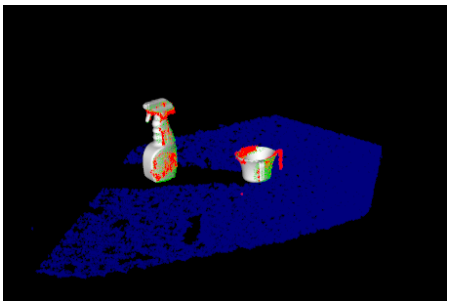
70 cm.

As we can see results of the object recognition matching are quite accurate with high matching rates. As explained in section 5.2, the background has been removed by filtering the depth and the table has been segmented. In this case only two clusters containing the model information have been found.

Occlusion conditions for those objects have been considered during this test. As we can see in figure 6.22 the spray flask has been occluded by the cup considering the position of the camera where the scene has been acquired. Although the spray flask does not contain part of its model information because of the occlusion, the object is found with a good pose estimation.



(a) Image capture by the Kinect - Occlusion on the spray flask

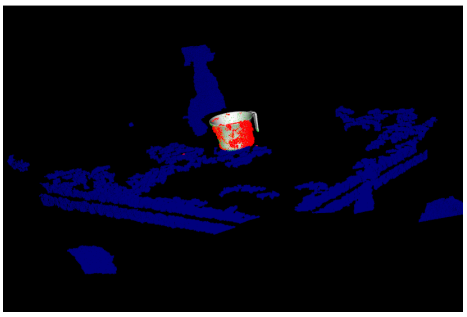


(b) Object recognition view

Figure 6.22. Success matching in occlusion conditions

However there can be scenes where, due to the occlusion, the object is not found as happened during the previous test. This is why, as you can see in figure 6.23, one failure was obtained during total acquisitions.

(a) Image capture by the Kinect - Occlusion on the spray flask



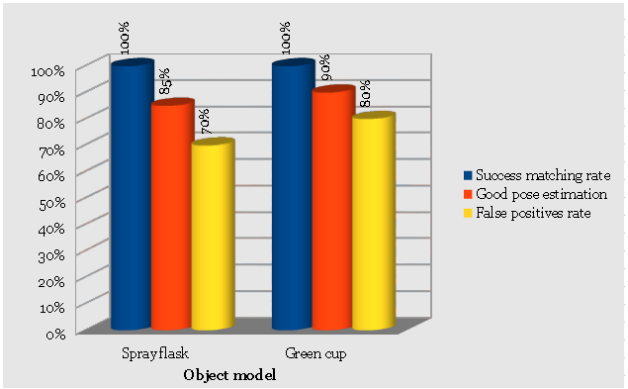
(b) View of the not successful Spray flask recognition

Figure 6.23. Failure matching in occlusion conditions

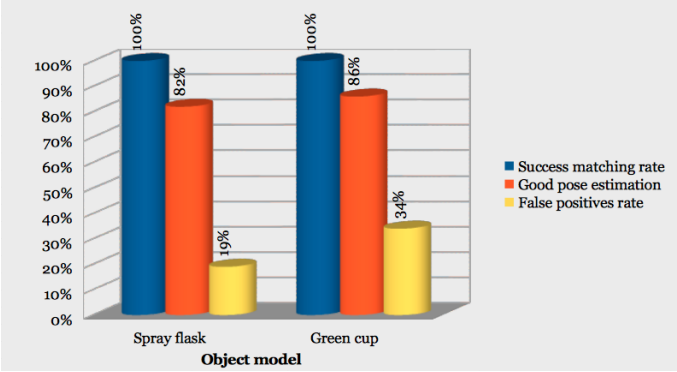
The second scenario instead consisted of the spray flask and the green cup as the previous one, but here we added two other objects: a wallet and an indian statue, as shown in figure 6.24.

Figure 6.24. Spray flask, green cup, wallet and indian statue table scenario

In figure 6.25 results of the object recognition in this scenario are shown.As in the previous example, the position of the Kinect from the center of the table was around 70 cm.



(a) Recognition results by using method *A*

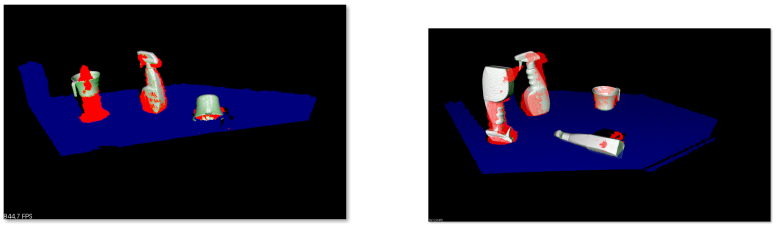


(b) Recognition results by using method *B*

Figure 6.25. Green Cup and Spray Flask recognition results in a more complex scenario

Occlusion conditions have not been considered, it is why the matching in this

experiment for both of objects is 100%. The weak spot of this algorithm is when objects which are not introduced as input lie in the same environment of the objects to recognize. Considering that for the match recognition only surfaces of single point of view are used, the surfaces of new objects can match with some of those point of views of the model. Considering that, in order to be recognized, an object needs to be matched at least 4 times in a row of 5 acquisitions, it happened, due to the surface properties of the objects in the scene, that the models given as input have been matched also in the wrong place. This is the reason of the false positive rate during the experimentations. An example of false positive obtained during the tests is shown in figure 6.26.



(a) Cup false positive detected on the indian statue surface

(b) Spray false positives detected on the wallet and indian statue surface

Figure 6.26. False positive matching using CVFH

6.4 Discussion

There is a substantial difference between the two methods which have been used during the experimentations. By using method B we are simulating the behavior of a robot which is not going to trust only one image but it is going to elaborate a set of images.

The acquisition of the 3D model using the software RGBD-6D-SLAM took some time considering that some conditions should hold in the environment to get a good result. Bright scenes, with many features (textures on the object and background) facilitate a fast acquisition of the model although sunlight is too bright for the infrared sensor, which means the environment should be illuminated artificially. We have noticed that the sensor error is lower for near objects, so textured foreground represents a good idea. This is the reason why our kitchen cup has been texturized.

The software can work either in continuous mode (acquiring data when the Kinect is moved) or just taking individual picture shots when a good position is reached. Although the continuous mode provides a good result if the number of features is increased in the "globaldefinitions" C++ code considering that the speed is not important for our acquisition because increasing the number of samples acquired, the speed of elaboration is slower and this does not allow a real-time application.

As regards the 3D object recognition descriptors evaluated during this project, considering the object recognition matching score, the CVFH offers excellent results, including also some scenarios where objects have in part been occluded by other objects. The object pose estimations is also really accurate. The problem of this descriptor regards the false positive rate matching when objects models are not given in input to the descriptor and can be found into the scene. The NARF descriptor instead has a different behavior depending on the scene and the objects to recognize. It depends strongly from the min_validation_score value, but, as we have seen, keeping the default value and changing the other options can give us a higher recognition matching with lower false positives that we could obtain by only lowering this value.

6.5 Hints for future improvements

In this section some suggestions that can improve the object recognition results will be given.

Working with Point Cloud Data files give us information about the color. This 2D information could be integrated in the NARF descriptor used above for those objects from which key points are extracted in the scene. The main idea could be to lower the min_validating_score to add another filter(besides the surface and border filter inside the code called **falsePositivesFilter::validationPoints()**) which concerns the color information. This filter could represent the global color of the object (if uniform) or a viewpoint of it. I would suggest the use of HSV, so the scenes are not conditioned so much by the light conditions. Furthermore a support size for each model given in input could be added to the main code.

In a similar way texture information coming from the CAD models can be considered. For models bringing this information a mapping between the object file and an image file is provided. So when CAD models are converted in Point Cloud Data files, this information needs to be converted too. The main problem here is to understand how the color information is stored in the CAD models and how to generate the training clusters into the Clustered Viewpoint Feature Histogram descriptor. The first point has been already solved introducing the Parser which is able to do this conversion. So the main problems is represented by the training of the data color and the use of this data in the testing step.

Chapter 7

Conclusions

The work done in this project is a demonstration of the importance of 3D data acquisition. The launch of the Kinect is revolutionary in the computer vision and robotic fields not only for the low cost of the device itself, compared with other technologies for the acquisition of three data as sterocameras, but also because the way in which these data can be captured and processed can be easy and straightforward. A big community is working on the processing of 3D data captured by the Kinect to manipulate them and develop new functionalities, such as segmentation, parser conversion and object recognition. This project is an example of how powerful these can be if combined together.

The objectives of this project were to reconstruct object models to give as input to 3D object recognition descriptors and to be able to handle those descriptors using the Kinect to get a streaming recognition.

The segmentation goal has been achieved using the RGBD-6D SLAM software. With this software we were able to reconstruct a full 3D environment model where the object has been placed in the middle, and then extracted. A simple segmentation has also been implemented which allows to segment simple views of the object.

Test trials in order to acquire the object have shown that, during the acquisition of the models using RGBD-6D SLAM, better results can be achieved if the models have texture properties.

CAD models (*ply* and *obj*) have also been part of the experimentations. This has been done to have a wider number of models whose matching object recognition results can be evaluated during the testing stage. Those CAD models have been converted into PCD files with parsers that have been created in this project for this purpose. The innovation of those parsers are the possibility to maintain the color information during the conversion. This has been done by applying a bilinear interpolation algorithm and the implementation created will be part of PCL.

The object recognition has instead been achieved by employing two different 3D feature descriptors: the NARF and the CSVF descriptors. Their experimental code is part of the ROS package although they work with different Point Cloud Library versions. During this project we managed to handle those 3D object recognition

descriptors by testing and evaluating their results, introducing some improvements and giving some hints for future work.

Results of the NARF object recognition analysis have shown that for the object recognition matching both the "reverse" planar segmentation introduced in this project together with the option "set unseen pixels to max range" and the option "use orientation invariant version" give some good rate results. The main problem of the first option is the number of false positives introduced.

Considering the CVSF we have seen that its object matching success rate is quite high in conditions where only the models which are given in input are present into the scene. In different scenarios, considering that the recognition is doing considering the object surfaces, the rate of the false positives which are found into the scene has a high value. In order to decrease the number of false positives, some hints for both descriptors have been introduced to improve the rate matching.

Appendix A

File Formats

A.1 Wavefront Object Structure

According to the researcher Martin Reddy [9], a Wavefront Object model is made of different sections. In the first one there is the information of the vertices of the model. Three values are needed to create a vertex in the space (XYZ) and they are given in the file after the initial header *v*. After the first section there is an optional section where the texture-color information is stored. The header that introduces this section is *vt*. This information is given in two values defined as *u* and *v*. These values contain the necessary access to the image picture which is provided together with the textured CAD model. With this 2D coordinate representation, it is possible to have a mapping into the 3D model by using the *Bresenham's line algorithm* [24], which represents a bilinear interpolation of points.

Then there is another section which contains the faces. With this information vertices are linked together in order to build the small faces which the model is made of with. Usually each face is composed by a set of three different vertices, which means small *triangles meshes* are created in order to build the whole model (see figure A.1).

The lines in the file containing the faces start with the header *f*. If both the vertex and texture-vertex sections are available, the face section contains many tuples of vertices and texture-vertices information stored in this way: f *v0_index/vt0_index v1_index/vt1_index v2_index/vt2_index*. The values stored actually represent the *line* on the section (*v* or *vt*) where the vertex/texture-vertex has been stored. In this case, the reference to the image is needed, and it is introduced by the header *usemtl*.

If the Wavefront Object model does not contain the texture-vertex section, then the face section is made of only vertex indices (e.g. f *v0_index v1_index v2_index*). So in this case the reference to the image picture is not needed.

An example of Wavefront Object Structure is given in listing A.3.

We will consider the case in which the texture-vertex information is available,

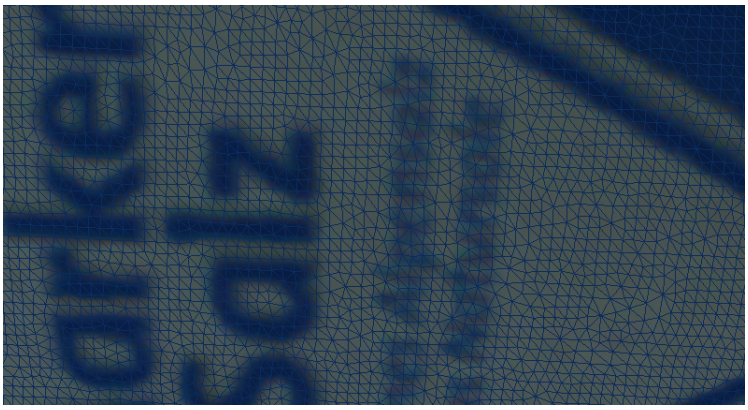


Figure A.1. A graphical example of the triangle meshes which a CAD model is made of with

because the final goal of our parser is to create models containing the color information.

Some Wavefront object models can also contain information about the normal, but for our purpose, the explanation given above is enough.

A.2 Polygon File Format Structure

According to the description given by Greg Turk[30], the content of a ply model is divided as follows:

- In the first part of the ply file, which is called *header*, there is the information of how the data is stored into the file. The number of vertices and faces is given in this header as well as the description of the properties for each element contained into the file.

- For each vertex the information of the X, Y and Z position is mandatory. Instead optional information like the RGB color data, density and luminosity can be stored into the file after the vertex position. We will consider the RGB data in the parser described afterwards, in order to have the color information stored *per vertex*.

- For each face a minimum of 3 values is required to create the structure. The color information can also be saved *per face* as well as it is stored into the *obj* file. In this case then the color information is stored into the 2D coordinates, u and v. Then, by using the *Bresenham's line algorithm* [24], it is possible to map this information into the 3D model. The number of vertexes and texture-vertexes is introduced before listing those elements. So, for example, when the face section starts, elements are listed in this way:

n_tot_vertexes *vertex_1 vertex_2 vertex_3* ***n_tot_textured_vertexes*** *u1 v1 u2 v2 u3 v3.*

An example of *ply* model taken from the Turk's website [30], and modified in order to show the explanation before, is given in listing A.5. Afterwards we will see we can create a *ply* model from an *obj* model, and we will be able to store the color information either *per vertex* and *per face*.

A.3 Point Cloud Data Structure

According to the description given by the creators of the Point Cloud Library, the Point Cloud Data file has the following structure:

- In the first part of the file, a *header* is defined. It contains the information about the properties of the file in terms of version, fields, size, type, width, height, number of points and the type of the data (ascii/binary).
- Fields that can be part of the PCD model are, for example, the X, Y and Z coordinates needed to identify the point in the space, the information about the color, which is actually stored in only one float value but its conversion into three different values (r, g and b) is easy to implement, (the instructions to achieve this conversion are showed in listing A.1). The reverse conversion (from (r, g and b) to a unique float value) is instead shown in listing A.2).
- Size is referred to the dimension of each field in bytes. Width can represent either the total length of the file (that means there is only one row with all the information saved in only one row, and from this derives that the value of the Height is equal to 1), or the length of the data memorized per line (if the width is for example 640 and the height is 480, this means that the information is stored like in a matrix structure).

```
uint8_t r, g, b;
//...values defined somewhere in the code
int32_t rgb_integer;
rgb_integer = (r << 16) | (g << 8) | b;
/*cast from integer to float.
This is the value which is stored into the PCD file*/
float rgb = *(float *)(&rgb_integer);
```

Listing A.1. Conversion from uint8_t r, uint8_t g and uint8_t b into float rgb

```
float rgb;
//rgb value defined somewhere..
uint32_t rgb_integer =  *reinterpret_cast<int*>(&rgb);
uint8_t r = ((rgb_integer >> 16) & 0xff);
uint8_t g = ((rgb_integer  >> 8) & 0xff);
```

```
  uint8_t b = (rgb_integer & 0xff);
```

Listing A.2. Conversion from float rgb into uint8_t r, uint8_t g and uint8_t values

After the header, all the information is considered as data. An example of PCD file is given in listing A.4.

```
v 0.1 0.2 0.2
v 0.2 0.3 0.4
v 0.5 0.6 0.7
v 1.2 1.6 1.7
....
vt 0 1
vt 1 0
vt 0.1 0.2
vt 0.3 0.5
....
usemtl textureImage.png
f 3/2 2/1 1/2
f 3/1 1/2 2/2
```

Listing A.3. Example of a obj file

```
# .PCD v.7 - Point Cloud Data file format
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 63815
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 63815
DATA ascii
-10.1025 -38.3424 -33.2597 2.03191e-39
-8.78072 -38.2784 -33.3098 2.30849e-39
-8.10687 -38.2322 -33.2899 1.20214e-39
-8.78072 -38.2784 -33.3098 2.30849e-39
-8.10687 -38.2322 -33.2899 1.20214e-39
-8.78072 -38.2784 -33.3098 2.30849e-39
-8.10687 -38.2322 -33.2899 1.20214e-39
.....
```

Listing A.4. Example of a pcd file

```
ply
format ascii 1.0
comment author: Greg Turk, modifiyed by faviuz
element vertex 4
property float x
property float y
property float z
property uchar red                  { start of vertex color }
property uchar green
property uchar blue
```

```
element face 4
property list uchar int vertex_index { number of vertices for each
    face }
property list uchar texture-vertex
end_header
0 0 0 255 0 0 #                          vertex 0
0 0 1 255 0 0 #                        vertex 1
0 1 1 255 0 0
0 1 0 255 0 0 #                          vertex 3
3 0 1 2 6 0.1 0.2 0.4 0.5 0.2 0.3 #      face 1
3 0 2 3 6 0 1 1 0 0 1 #                  face 2
3 0 1 3 6 0.54 0.31 0.34 0.53 0.23 0.32
3 2 1 2 6 0.12 0.421 0.42 0.21 0.12 0.32 #face 4
```

Listing A.5. Example of a ply file

Bibliography

[1] Canny J., *A Computational Approach To Edge Detection*, IEEE Trans Pattern Analysis and Machine Intelligence, 1986, pp679-714

[2] XBOX 360, *The Kinect*
http://www.xbox.com/en-US/kinect

[3] Duggal Vijay, *CADD primer, a general guide to computer aided design and drafting : CAD*, 2000

[4] *CVAP | The Computer Vision and Active Perception Lab at KTH in Stockholm*
http://www.kth.se/csc/forskning/cvap?l=en_UK

[5] John Folkesson, *Research Scientist at KTH*
http://www.csc.kth.se/ johnf/

[6] Marina Indri, *Associate Professor of Automatic Controls at Politecnico di Torino*
http://staff.polito.it/marina.indri/

[7] Bastian Steder, *Phd Student at Albert-Ludwigs-University of Freiburg*
http://www.informatik.uni-freiburg.de/ steder/

[8] Experiments and use guide of this project available at:
https://subversion.assembla.com/svn/thesis-kinect/

[9] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney, *Level of Detail for 3D Graphics*, Elsevier Science Inc. New York USA, 2002

[10] David G. Lowe, *Object recognition from local scale-invariant features*, International Conference on Computer Vision, Corfu, Greece, 1999, pp. 1150-1157

[11] Example of the sift method - Picture available at:
http://www.cmap.polytechnique.fr/ỹu/research/ASIFT/demo.html

[12] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, *Speeded-up robust features (SURF)*, Computer Vision and Image Understanding, 2008, vol. 110 pp. 346-359

[13] Surf Source available at:
http://www.vision.ee.ethz.ch/ surf/

[14] Radu Bogdan Rusu, Nico Blodow and Michael Beetz, *Fast Point Feature Histograms (FPFH) for 3D registration*, ICRA, 2009, pp3212-3217

[15] Radu Bogdan Rusu, *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*, 2009

[16] Radu Bogdan Rusu, Code and images of the Point Feature Histograms available at:
http://pointclouds.org/documentation/tutorials/pfh_estimation.php

[17] Radu Bogdan Rusu, Code and images of the Fast Point Feature Histograms available at:
http://pointclouds.org/documentation/tutorials/fpfh_estimation.php

[18] Radu Bogdan Rusu, Gary R. Bradski, Romain Thibaux, John Hsu - emphFast 3D recognition and pose using the Viewpoint Feature Histogram, IROS 2010, pp2155-2162

[19] Aitor Aldomà and Radu Bogdan Rusu, *Clustered Viewpoint Feature Histogram page*
http://www.ros.org/wiki/vfh_recognition

[20] Radu Bogdan Rusu, *Research Scientist at Willow Garage*
http://rbrusu.com/

[21] Radu Bogdan Rusu and Steve Cousins, *3D is here: Point Cloud Library (PCL)*, IEEE International Conference on Robotics and Automation, ICRA 2011

[22] Point Cloud Library Website
http://www.pointclouds.org

[23] Willow Garage, *PR2 Sensor Head*
http://www.willowgarage.com/pages/pr2/overview

[24] Colin Flanagan, *The Bresenham Line Algorithm*
http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html

[25] P.J. Besl, H.-D. McKay - *A method for registration of 3-D shapes* - Pattern Analysis and Machine Intelligence, 1992, pp239-256

[26] Bastian Steder,Radu Bogdan Rusu,Kurt Konolige, Wolfram Burgard -*Point Feature Extraction on 3D Range Scans Taking into Account Object Boundaries* - Robotics and Automation (ICRA), IEEE International Conference, 2010, pp2601 - 2608

[27] Kinect picture available at:
http://en.wikipedia.org/wiki/Kinect

[28] ROS, *nodes in ROS*
http://www.ros.org/wiki/Nodes

[29] Kinect box picture available at:
http://www.aeropause.com/2010/11/do-you-want-kinect-4-questions-to-ask-yourself/kinect-box/

[30] Greg Turk, *The PLY Polygon File Format*, the Board of Trustees of The Leland Stanford Junior University
http://people.sc.fsu.edu/~jburkardt/data/ply/ply.html

[31] Pcl creators, *Description of the pcd structure*
http://www.pointclouds.org/documentation/tutorials/pcd_file_format.php

[32] G. R. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O REILLY, 2008.

[33] Institute of Anthropomatics - *KIT ObjectModels Web Database*
http://i61p109.ira.uka.de/ObjectModelsWebUI/index.php?section=publications

[34] Visual Computing Lab, *Meshlab System*
http://meshlab.sourceforge.net

[35] Franklin C. Crow, *Summed-area tables for texture mapping*, Proceedings of the 11th annual conference on Computer graphics and interactive techniques, pp207-212, 1984

TRITA-CSC-E 2011:115
ISRN-KTH/CSC/E--11/115-SE
ISSN-1653-5715

www.kth.se