University of Padua

DEPARTMENT OF INFORMATION ENGINEERING

Master Degree in Computer Engineering

Object recognition for an autonomous wheelchair equipped with a RGB-D camera

Candidate:

Alberto Bertan

Supervisor:

Enrico Pagello

Co-supervisor:

Chen Weidong
(SJTU - Shanghai)

Academic year 2011–2012

"Una volta che abbiate conosciuto il volo, camminerete sulla terra guardando il cielo, perché là siete stati e là desidererete tornare."

Leonardo da Vinci

Abstract

This thesis has been carried out within a project at AR Lab (Autonomous Robot Laboratory) and IAS-Lab (Intelligent Autonomous Systems Lab) of Shanghai Jiao Tong University and University of Padua respectively. The project aims to create a system to recognize and localize multiple object classes for an autonomous wheelchair called JiaoLong, and in general for a mobile robot.

The thesis had as main objective the creation of an object recognition and localization system in an indoor environment through a RGB-D sensor. The approach we followed is based on the recognition of the object by using 2D algorithm and 3D informations to identify location and size of it. This will help to obtained robust performance for the recognition step and accurate estimation for the localization, thus changing the behaviour of the robot in accordance with the class and the location of the ojbect in the room.

This thesis is mainly based on two aspects:

- the creation of a 2D module to recognize and detect the object in a RGB image;
- the creation of a 3D module to filter point cloud and estimate pose and size of the object.

In this thesis we used the Bag of Features algorithm to perform the recognition of objects and faces and a variation of the Costellation Method algorithm for the detection; 3D data are computed with several filtering algorithms which leads to a 3D analysis of the object, then are used the intrinsic informations of point clouds for the pose and size estimation. We will also analyze the performance of the algorithm and propose some improvements aimed to increase the overall performance of the system besides research directions that this project could lead.

Contents

List of Figures

Chapter 1

Introduction

1.1 Summary

The purpose of this work is recognize the class of an object and extract useful information about it, through a RGB-D sensor. The main objective consists in build a robust system that can distinguish objects in different and changable environments, then estimate pose and size of them, to change the behaviour of the robot, this helps it to improve the performance and interact better with people, for example when recognize a desk, could estimate if there is enough space or not to dock in a wheelchair, or when it recognize a door if there is enough space to pass through it saftely. In order to obtain this, we have been used a more conservative approach for the recognition step, to take advantage of informations contained on point clouds, which allows more accurate spatial/geometric information. The decision of first part is due to the poor state of the arts and performance of algorithms working on only 3D information, which push us to choose on the 2D literature, which has decades of research and develop, then gives more robust and fast techniques for recognition and detection. Opposite consideration regards the second part, where the only 2D data does not allow accurate information about size and pose in the real world, conversely the 3D data gained with the camera used in this project (Microsoft Kinect) which shows a precision of $0.5mm$ at shortest distance of $50cm$ and $7cm$ at farest distance of $5m$ [1]. To do that has been developed a first module called `2d_processing` that recognize and detect the object on the 2D world and a second module called `3d_processing` that extract and estimate informations. Finally the results of the work are shown.

1.2 State of the art

Many works exist about object recognition and localization by using RGB cameras only [2][3][4][5][6][7] or 3D sensors only [8][9][10][11].

Works which face only with 2D informations lead to good results in term of performance or time processing, but they tend to focus on single view objects, small variations of position or different views but for a single classe of object. These lead to a lose of spatial and geometric informations, useful for a better reppresentation of the environment, and potential tasks. Different situation regards the 3D works where many approaches have been put forth, but many of these methods require that the target be segmented from the background, which makes them difficult to apply to real-subtle shape variations, especially with large parts of the shape missing from the scene. Moreover in several cases, no informations are given about the computational time needed by the algorithms and results are reported for some sequences acquired from a static platform.

However, when dealing with mobile robots, the need for robustness and real time capabilities usually led researchers to tackle these problems by combining appearance and depth information, which is the direction followed in our project.

1.3 Content

This document is structured as follows:

- Chapter 2 describes the main hardware and software that form the system.
- Chapter 3 describes the approach we followed to set up the whole system.
- Chapter 4 and 5 describe in details the two main phases namely 2D and 3D module.
- Chapter 6 shows both analytical results about our approach and a real-world application of the system.
- Chapter 7 contains the final conclusions about the work.

Chapter 2

Work Environment

2.1 Microsoft Kinect

Microsoft Kinect[1] (Figure 2.1) is an accessory for the Microsoft Xbox 360 video game platform. It is considered a "controller-free gaming and entertainment experience". In fact, it can interpret 3D scene information using a RGB camera, a depth sensor and a multi-array microphone. The depth sensor consists of an infrared laser projector (which creates a grid of points) combined with a monochrome CMOS sensor, which interprets the infrared light captured creating a depth map. It is also equipped with a motorized pivot that permits the sensor to tilt up to 27° either up or down and has an angular field of view of 57° horizontally and 43° vertically.

Video stream. Kinect sensor outputs video at a frame rate of $30Hz$ as a combination of two different streams. The first one is provided by the RGB camera which uses 8-bit VGA resolution (640×480 pixels) with a Bayer color filter, the other one, instead, is produced by the monochrome depth sensor with the same resolution and 2,048 levels of sensitivity (11 bits).

Software technology. The software technology provided to the Xbox developers enables advanced gesture recognition, facial recognition and voice recognition as well as tracking up to six people simultaneously (declared but probably it is able to track how many people can fit in the field-of-view of the camera), including two active players for motion analysis with a feature extraction of 20 joints per player.

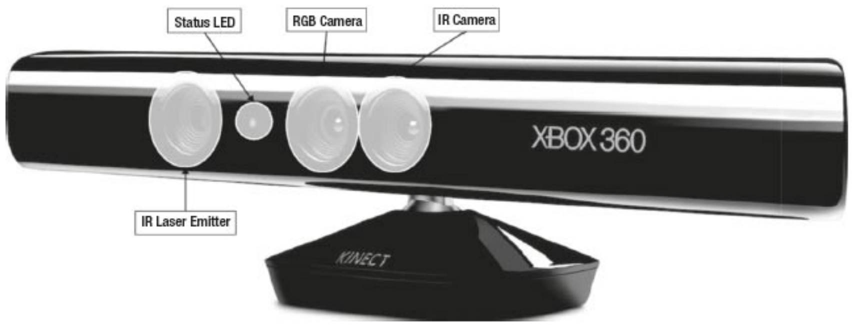


Figure 2.1: *Microsoft Kinect sensor.*

[1]http://www.xbox.com/en-US/kinect

2.2 JiaoLong

JiaoLong[12][13] is an intelligent wheelchair developed by the Autonomous Robot Lab[2] of Shainghai Jiao Tong University. Its design intention is based on human-guided method which means users can give some simple guidance during application. Figure 2.2 shows that there are three interactive ways to input the human guidance. One is microphone, another is joystick and the other one is touch screen. That means users can interact with JiaoLong through voice, hands or screen touch. Also users can get real-time information, such as the global map and current velocity from touch screen.

Hardware and Software technology. Its length, width and height are respectively 110cm, 90cm and 105cm. It is powered by a group of 12V batteries which can supply power long as 6-8 hours continuously. The back wheels adopt DC motors which are controlled by a motor controlling board(DSP 2407) adopting two wheeled differential driving. It has an Industrial Computer (WinXP, Intel Core2 Duo @ 2.0GHz CPU) for management of interactive information, and a Laptop (Linux, Intel ATOM N270 @ 1.6GHz CPU) for real-time execution of navigation algorithm. The sensors are encoder-based odometer, laser range finder (LRF, SICK LMS100) and a RGB-D camera (Microsoft Kinect).

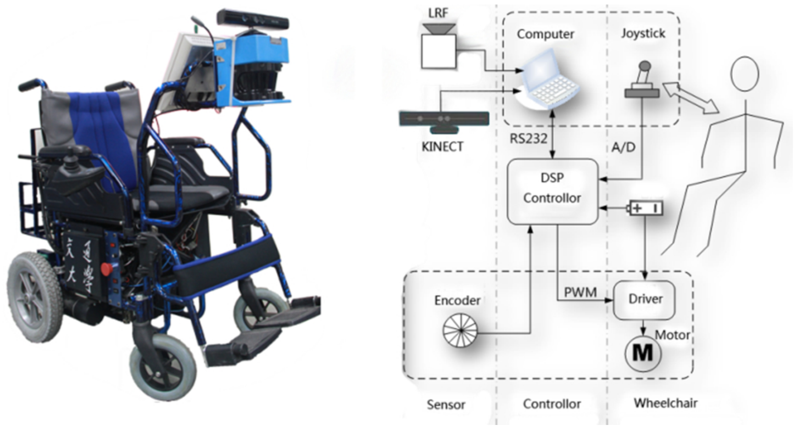


Figure 2.2: *Autonomous wheelchair JiaoLong.*

2.2.1 Shared control

The share control has two key parts: the reactive control and the weight optimization. The reactive control provides basic obstacle avoidance using MVFHVFF methods [14][15]. The weight optimal algorithm optimizes three indicators which will be discussed in the following section to obtain weight of reactive control and user.

Weight optimization. In the previous work [12], indicators of wheelchair's performance were proposed: *safety*, *comfort* and *obedience*. *Safety* measures the probability of collision. *Comfort* measures the variation of angular velocity. *Obedience* measures the degree of obedience to the user's control intention. These indicators are defined as:

$$\begin{aligned} safety &= 1 - exp(-\alpha \, . \, dis); \\ comfort &= exp(-\beta|\omega - \omega_0|); \\ obedience &= exp(-\gamma|\xi - \xi^*|). \end{aligned}$$

where, *dis* measured the distance between the wheelchair and the nearest obstacle in its path; ω and ω_0 are the desired and current angular velocity; ξ^* is the orientation of user command

[2]http://robotics.sjtu.edu.cn/English/

calculated from the user's input ν_{mach} and ω_{mach}; ξ is the orientation of final command determined by ν and ω; α , β and γ are constants.

The aim of weight optimization is to maximize all three indicators. However, these indicators are usually contradictory to each other. Therefore, there is no absolute optimum solution for maximize the three indicators at the same time. So we proposed of solving this problem is: always improve the smallest indicator among the three. In accordance with this principle we choose the minimax method to simplify this multi-objective optimization problem to a single objective one.

$$\begin{cases} max_k(min(safety, comfort, obedience)) \\ s.t \\ \nu(t) = \nu_{user}(t) \\ \omega(t) = k\omega_{user}(t) + (1-k)\omega_{mach}(t) \\ 1 \geq k \geq 0 \end{cases}$$

where, k and $(1-k)$ is the user weight and the reactive control weight; $\nu(t)$ and $\omega(t)$ is the linear and angular velocity to be sent to the wheelchair. This equation means that finding the user weight is equivalent to finding the proper k to maximize the objective function $min(safety, comfort, obedience)$ under the restrictions stated after $s.t.$. As the linear velocity in MVFH is equal to $\nu_{user}(t)$ as long as there is no possible collision, we restrict $\nu(t)$ to be equal to $\nu_{user}(t)$.

Previous equation as a linearly constrained nonlinear programming problem, there is generally no analytical solution, since we use one-dimensional search algorithm to solve the optimization: First, use rough search algorithm to determine the interval that contain the maximum of the objective function $min(safety, comfort, obedience)$; Second, implement Golden section search algorithm in the interval mentioned above to find the k at maximum of the objective function.

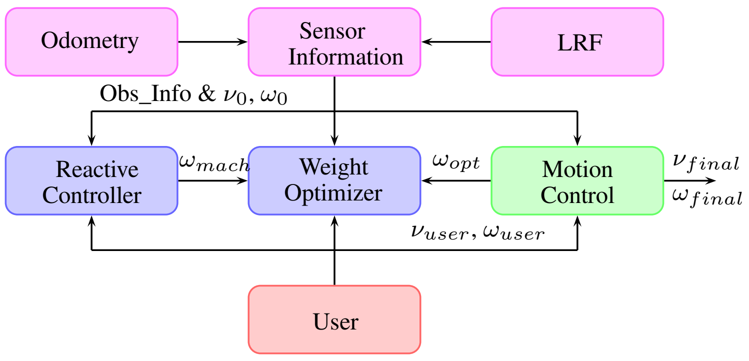


Figure 2.3: *Architecture of remote control system.*

2.3 ROS - Robot Operating System

ROS[3] (*Robot Operating System*) is a framework for robot software development, providing operating system-like functionality on top of a heterogeneous computer cluster. ROS was originally developed by the Stanford Artificial Intelligence Laboratory but, as of 2008, development continues primarily at Willow Garage[4] with more than twenty institutions collaborating in a federated development model.
ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly-used functionalities, message-passing between processes, and package management. It is based on a graph architecture where processing takes place in

[3]http://www.ros.org
[4]http://www.willowgarage.com

nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. The library is geared towards a Unix-like system (Ubuntu Linux is listed as "supported" while other variants such as Fedora and Mac OS X are considered "experimental"). ROS has two basic "sides": the operating system side called `ros` as described above and `ros-pkg`, a suite of user contributed packages that implement functionalities such as simultaneous localization and mapping, planning, perception, simulation etc. ROS is released under the terms of the BSD license, and is open source software. It is free for commercial and research use.

2.3.1 Computation graph architecture

As already mentioned, ROS is based on a graph architecture. From the computational point of view the graph is the peer-to-peer network of ROS processes that share data. So let's have a quick overview of these graph concepts.

Nodes. A node is substantially a process that performs computation: it is where all the main operations are done, the principal entity in the graph. Nodes communicate with each other by using streaming **topics**, **RPC services** or the **Parameter Server**.

Messages. In ROS, a message is intended as a simple data structure comprising typed fields. Standard primitive types are supported, as are arrays of primitive types or previously defined messages.

Topics. A topic is a named bus over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption and are intended for unidirectional, streaming communication. There can be multiple publishers and subscribers to a topic.

Services. The publish/subscribe model is a very flexible communication paradigm, but it is not appropriate for RPC request/reply interactions. Request/reply is done via a service which is defined by a pair of messages: one for the request and one for the reply.

Master. Name service for ROS. It provides name registration and lookup to the rest of the computation graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

Parameter Server. It is a shared, multi-variate dictionary that is accessible via network APIs. Nodes can use this server to store and retrieve parameters at runtime.

Bags. Bags are a format for saving and playing back ROS message data. They are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

An example of a graph created during a ROS session is visible in figure 2.4.

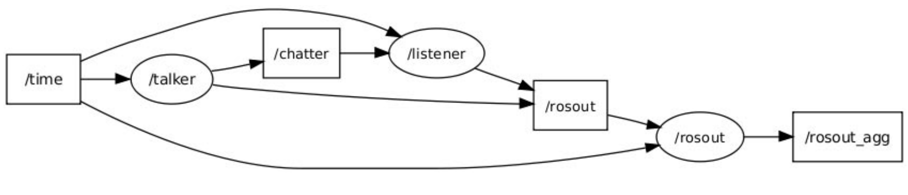


Figure 2.4: *Graphical representation of nodes (ellipses) and topics (rectangles).*

2.4 Libraries

2.4.1 PCL

The PCL[5] (*Point Cloud Library*) is a large scale, open project for point cloud processing [16]. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them. It is well integrated into ROS which also provides some functionalities as ready to use nodes.

PCL is released under the terms of the BSD license and is open source software. It is free for commercial and research use and it is supported by companies such as Google, NVidia and Toyota.

2.4.2 OpenNI

OpenNI[6] (*Open Natural Interaction*)[17] is a multi-language, cross-platform framework that defines APIs for writing applications utilizing Natural Interaction.
NI (*Natural Interaction*) refers to the concept that Human-Machine-Interaction is achieved by human senses and, most of all, vision and hearing. OpenNI aims to define a standard API that is able of dealing with both vision and sensors, and a vision and audio perception middleware, allowing communication between the two components.
OpenNI provide two types of APIs:

1. implemented APIs: allow to deal with the sensor device;
2. not implemented APIs: allow to deal with the middleware components.

The clear distinction between sensors and middleware components is based on the "write once, deploy everywhere" principle. In fact OpenNI allows the porting of applications and moreover enables to write algorithm that works with known raw data independently from the sensor that has generated them. From the producer point of view, instead, OpenNI offers the possibility of building sensors for applications by just providing raw data and not APIs on how to deal with them. An application of OpenNI is for example the tracking of real-life 3D scenes.
OpenNI is an open source API that is publicly available.

The OpenNI Framework is an abstract layer (Figure 2.5) that provides the interface for both physical devices and middleware components. Multiple components can register to this framework based on the specific API and they are called modules.
A module is responsible for producing and processing the data of the sensor and the currently supported ones are:

1. 3D sensor;
2. RGB camera;
3. IR camera;
4. audio device.

Based on this, OpenNI provides also the following middleware components:

1. full body analysis;

[5]`http://www.pointclouds.org`
[6]`http://www.openni.org`

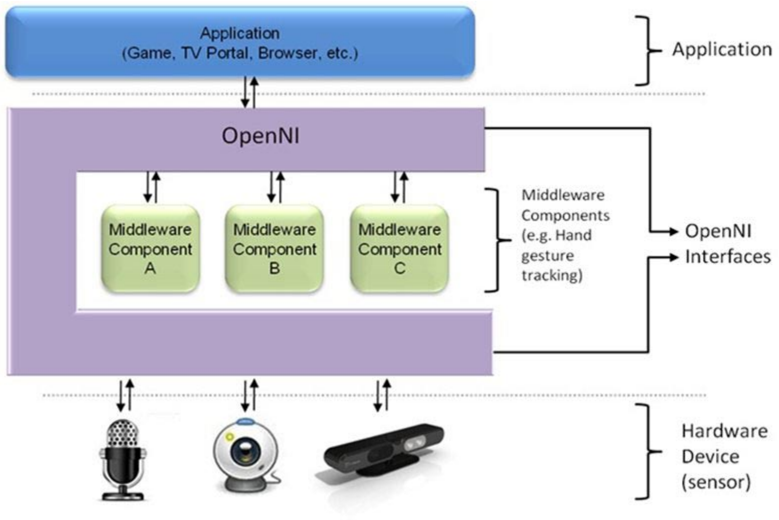


Figure 2.5: *The OpenNI abstract layered structure.*

2. hand point analysis;
3. gesture detection;
4. scene analyzer (segmentation, clustering and coordinates framing).

OpenNI relies on Production Nodes. They represents the productive part of the system, that is they create the data required for the interaction. These data can be either low level ones, RGB for example, either composited ones. In fact production nodes can also control lower level production nodes and they can in turn be used by higher level ones. In order to define communication and hierarchy these nodes are organized in production chains.

2.4.3 OpenCV

OpenCV[7] (*Free Open Source Computer Vision*) is a library of programming functions mainly aimed at real time computer vision [18][19]. Example applications of the OpenCV library are Human-Computer Interaction (HCI), object identification, segmentation and recognition, face recognition, gesture recognition, motion tracking, ego motion, motion understanding, structure from motion (SFM), stereo and multi-camera calibration and depth computation, mobile robotics. As for PCL, OpenCV is completely integrated into ROS which also provides image type conversions between OpenCV and ROS formats. It has a BSD license and it is free for commercial or research use.

[7]`http://opencv.willowgarage.com`

Chapter 3

System overview

The recognition approach we decided to follow in order to accomplish our objectives is an hybrid approach with uses 2D data to recognize several classes of objects and 3D data to compute localization and pose estimation. Such approach involves several steps for each frame, that will be resume in next sections. The main challenges when using an object detector are that the resulting output is unreliable and sparse, i.e., detectors only deliver a discrete set of responses and usually yield false positives and missing detections.

Our purpose of performing this operation from a mobile robot makes this task even more challenging. We need to face problems like changing views, variable environments or the conversion from the local coordinates system to the world one.

The chapter is organized as follows. Sections 3.1 and 3.2 explain briefly the approaches we followed to create a 2D and a 3D processing module by combining existing methodologies with the data generated by our sensors, in particular the Kinect. Then, in section 3.3, 3.3.1 and 3.3.2, we describe the implementation and two main nodes of the system.

3.1 2D Module

An object recognition system finds objects in the real world from an image of the world, using object models which are known a priori. This task is surprisingly difficult [20]. Humans perform object recognition effortlessly and instantaneously. Algorithmic description of this task for implementation on machines has been very difficult. Object recognition in RGB images is an active research area in computer vision and, in past decades, numerous approaches have been proposed. Among them, there are consolidated techniques that recognize object really well and consist in computing interesting features in the frame and check if its match with others references features of our objects. One of these techniques make is called Bag of Feature and use a dictionary of visual word to train a classifier as SVM (*Support Vector Machine*) to recognize different classes of objects. The drawback of this approach is that the performance can be easily affected by background clutter and occlusions and, not less important, it usually can't detect in which area the object is recognized.

Our idea is to use a second method to detect the area when the class of the object is recognized, with fast search structures that allowed to store several images with different angle of view for the same object. To reduce the number of false matching a double control is applied, with a NN ratio test and the homography of the remaining points. A simple overview of our approach is visible in Figure 3.1.

3.2 3D Module

Once we have a class and an area which contains the object, we need a method to extract location and pose and, its size for every dimensions. For this step we used the pointclouds cor-

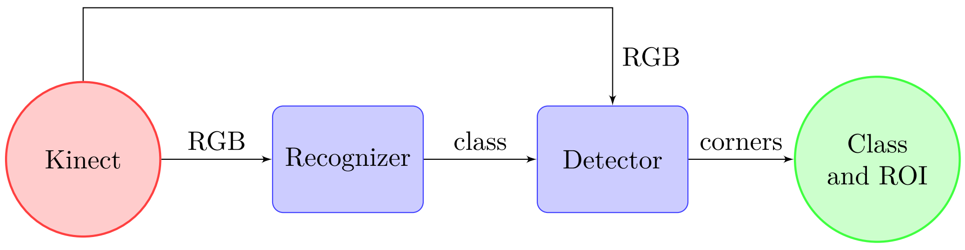


Figure 3.1: *Our 2D module.*

rispondent to the area previously detected, then filtering all the useless information and obtain only the object. These operations are high cost computationally, but with a reduced number of points the time dropped drastically, and consist on downsampling, filtering by distance, clustering, deleting of ground plane and removing of outliers, then the biggest cluster is selected.

Problems can arise when dealing with full occlusions (e.g. an object can be located behind another one). In this case, since our objective is a full detection, we need a method to recover the original shape of the object. Our idea consists in finding a corrispondence between different parts. An overview of the 3D module is visible in Figure 3.2, while a detailed description of the method is reported in chapter 4.

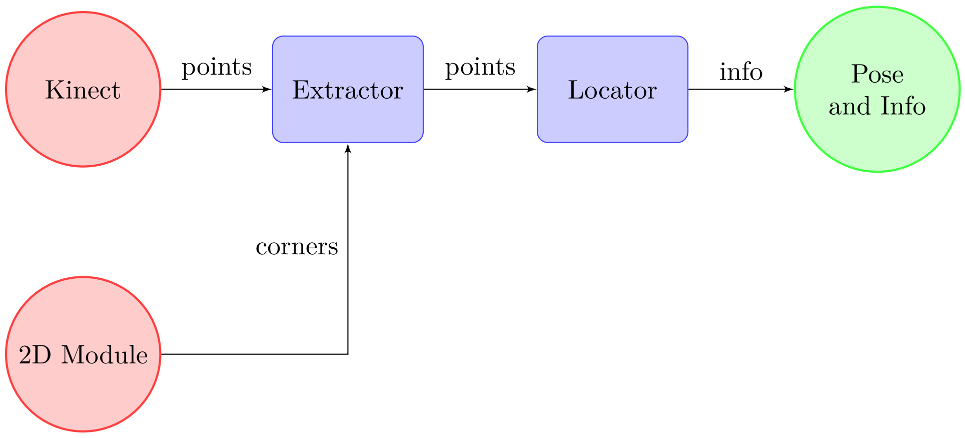


Figure 3.2: *Our 3D module.*

3.3 Implementation

The whole system has been developed to run exclusively with ROS and to use all of its potentialities as much as possible. Thus, in the remainder of the text, we use the words *program* and *node* indistinctly to refer to a standalone executable. When we talk about *publishing* or *subscribing* it means we use the message-passing architecture to exchange information among different nodes.

3.3.1 ROS Kinect node

The main source of data is the Kinect. We run it as a simple node that publishes both the RGB image and the point cloud. While the time needed for the publication of the RGB image is negligible, the generation and publication of a point cloud are quite a computationally expensive task: it takes about one second to process 640×480 points. This time is decreased significantly by reducing the number of points.

3.3.2 JiaoLong Perception node

This node compute the data from the Kinect node and, to be sure to has the same corrispondence between RGB image and the point cloud, it obtains the first one from the second one, with the construction of the image from the colour information of the single point. Then with both data can recognize, detect and extract informations about the object and published it on a dedicate topic, where the autonomous robot can collect the results.

Chapter 4

2D Module

Object recognition is a key feature for building robots capable of moving and performing tasks in human environments. This ability in real scenes is one of the most challenging problems in computer vision, as it is necessary to deal with difficulties such as viewpoint changes, occlusions, illumination variations, background clutter or sensor noise. Furthermore, in a mobile robotics scenario a new challenge is added to the list: computational complexity. In a dynamic world, information about the objects in the scene can become obsolete even before it is ready to be used if the recognition algorithm is not fast enough. All these complications make object recognition in real scenes a hard problem, that demand significant effort.
Moreover when an object is recognized becomes interesting detect in which area of the image is positioned. Then the field of object detection is a natural evolution of object recognition, which inherits all difficulties.

This chapter is organized as follows, first will be presented common techniques for the image processing, then we will move to the core of 2D module or rather how are structured and built the recognition and detections methods.

4.1 Image processing

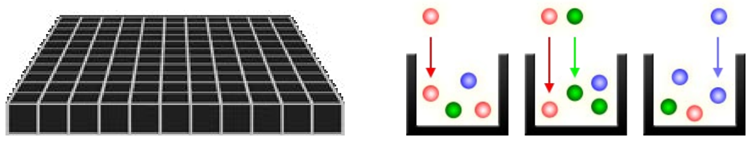
Images collected by a robot or a sensor in the real world, are presented in a computer as a matrix. Each pixel has a pixel value which describes how bright that pixel is, and/or what color it should be, which needs to be discretized before to be elaborated. Despite the sensors used on mobile robots have a low resolution, it return a huge amount of informations, that is hard to compute with real-time constraints for a computer. For this reason, first task is reduce the quantity of informations related to an image and at the same time, it must avoid to lose useful informations. These will be used in a second time to obtain corrispondences between two (or more) images of the same scene (or object).

4.1.1 Image acquisition

Over the years the computer vision has improved the basic techinques for image acquisition, then we will present briefly an overview of these techniques more used:

- RGB camera;
- Stereo camera;

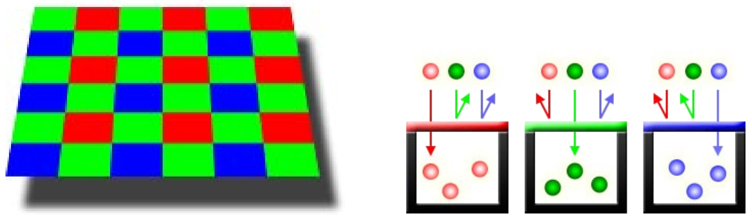
Traditional camera uses an array of millions of tiny light cavities or "photosites" to record an image. When the exposure begins, each of these is uncovered to collect and store photons. With a certain frequency there is a refresh step where, the camera closes each of these photosites, and then tries to assess how many photons fell into each. The relative quantity of photons in each cavity are then sorted into various intensity levels, whose precision is determined by bit depth (0 - 255 for an 8-bit image).



(a) *Cavity Array* **(b)** *Light Cavities*

Figure 4.1: *Creation of a grayscale image.*

However, Figure 4.1 illustrate only the creation of grayscale images, since these cavities are unable to distinguish how much they have of each color. To capture color images, a filter has to be placed over each cavity that permits only particular colors of light. Virtually all current digital cameras can only capture one of three primary colors in each cavity, and so they discard roughly 2/3 of the incoming light. As a result, the camera has to approximate the other two primary colors in order to have full color at every pixel. The most common type of color filter array is called a "Bayer array" shown in Figure 4.2.



(a) *Color Filter Array* **(b)** *Color Filters*

Figure 4.2: *Creation of a color image.*

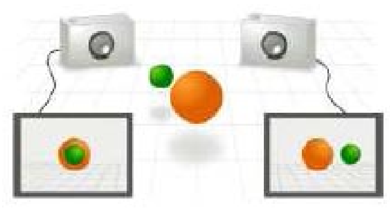
Now we only need to compute these informations in a 2D world, where we have knowledge of chromatic and luminance properties but not of depth, then we lose informations about position and proportion in the real world. Over the years has been developed several techniques to extract these information, however did not give high efficency results.

Stereo vision is the extraction of 3D information from digital images, such as obtained by a CCD camera. By comparing information about a scene from two vantage points, 3D information can be extracted by examination of the relative positions of objects in the two panels. This is similar to the biological process Stereopsis that was a notion well known since the Renaissance period.

In traditional stereo vision, two cameras, displaced horizontally from one another are used to obtain two differing views on a scene, in a manner similar to human binocular vision. By comparing these two images, the relative depth information can be obtained, in the form of disparities, which are inversely proportional to the differences in distance to the objects. To compare the images, the two views must be superimposed in a stereoscopic device, the image from the right camera being shown to the observer's right eye and from the left one to the left eye.

In real camera systems however, several pre-processing steps required.

1. The image must first be removed of distortions, such as barrel distortion to ensure that the observed image is purely projectional.
2. The image must be projected back to a common plane to allow comparison of the image pairs, known as image rectification.
3. The displacement of relative features is measured to calculate a disparity map.
4. Optionally, the disparity as observed by the common projection, is converted back to the height map by inversion. Utilising the correct proportionality constant, the height map can be calibrated to provide exact distances.


(a) *Principle of stereo vision*

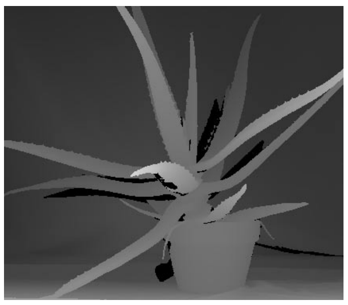

(b) *Disparity map*

Figure 4.3: *Stereo vision system.*

4.1.2 Features computation

Feature detection [21] is the process of deciding where and at what scale to sample an image. The output of feature detection is a set of keypoints that specify locations in the image with corresponding scales and orientations. These keypoints are distinct from feature descriptors, which encode information from the pixels in the neighborhood of the keypoints.

There is a substantial body of literature that focuses on detecting the location, we will focus more on the ones developed from the goal of finding keypoints useful for image registration that are stable under minor affine and photometric transformations. These feature detection methods are referred to as Interest Point Operators.

While there are many variations, an interest point operator typically detects keypoints using scale space representations of images. A scale space represents the image at multiple resolutions, and is generated by convolving the image with a set of guassian kernels spanning a range of s values. The result is a data structure which is, among other things, a convenient way to efficiently apply image processing operations at multiple scales. Interest point operators detect locally discriminating features, such as corners, blob-like regions, or curves. Responses to a filter designed to detect these features are located in a three dimensional coordinate space, *(x; y; s)*, where *(x; y)* is the pixel location and *s* is the scale. Extremal values for the responses over local *(x; y; s)* neighborhoods are identified as interest points. Interest point operators are designed to be robust to small affine and photometric image transformations, with the goal of being able to find the same keypoints in two similar but distinct images.
Main characteristics for a good detector are:

Repeatability. The skill to detect many features point independently by trasformations related to the images.

Distinctiveness/informativeness. The intensity patterns around the features point detected should contains a lot of "information", this must increase the efficency of matching step.

Locality. Features should be detected by analyzing as much local context as possible, to avoid the probabilty that a feature includes occlusions and/or other objects.

Quantity. Should be detected many features, also in small objects.

Accuracy. The detection must be accurate, not only spatially, but also respect to the scale.

Efficiency. Often is required to use it in real-time context.

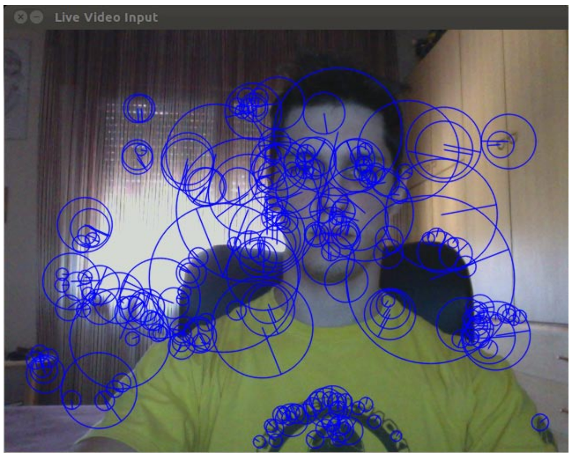


Figure 4.4: *Example of features extracted, as we can see there is a high distribution on areas with high density of "information".*

In addition to determining where and to what extent a feature exists in an image, there is a separate body of research to determine how to represent the neighborhood of pixels near a localized region, called the feature descriptor. The simplest approach is to simply use the pixel intensity values, scaled for the size of the region, or an eigenspace representation thereof.
Main characteristics for a good descriptor are:

Repeatability. Gives two features corrispondent, the descriptor should be as much as possible the same, independently by the transformations that regard the two images.

Distinctiveness/informativeness. The "information" on the patch should be accurate, in order to make easy distinguish each feature from the others.

Compactness. A compact descriptor does not contain redundant data neither high correlated data. Not only this has a deep impact on memory constraints, but also on the performance of matching process.

Efficiency. Should be many feature points, then it is necessary describe them quickly.

4.2 Object recognition

Numerous methods for object recognition have been developed over the last decades, but few of them actually scale to the demands posed by a mobile robotics scenario. We should like to identify processes that are sufficiently generic to cope with many object types simultaneously and which are readily extended to new object types. At the same time, these processes should handle the variations in view, imaging, lighting and occlusion, typical of the real world, as well as the intra-class variations typical of semantic classes of everyday objects. Two successful general object recognition approaches include: the constellation method proposed by Lowe together with its interest point detector and descriptor SIFT [3] and a bag of features approach, the one developed by Nister and Stewenius [5]. The authors of both approaches have specifically addressed the issue of computational complexity and claim that proper implementations of their algorithms can recognise a significant number of objects in real time.

For this part we have decided to use bag of features approach, then with some variations using the costellation method for the detection task.

A **Bag of Features** corresponds to a histogram of the number of occurrences of particular image patterns in a given image. The main advantages of the method are its simplicity, its computational efficiency and its invariance to affine transformations, as well as occlusion, lighting and intra-class variations. This approach can be motivated by an analogy to learning methods using the *bag-of-words* representation for text categorization [22][23][24]. The idea of adapting text categorization approaches to visual categorization is not new. Zhu et al [6] investigated the vector quantization of small square image windows, which they called *keyblocks*. They showed that these features produced more "semantics oriented" results than color and texture based approaches, when combined with analogues of the well-known vector-, histogram-, and n-gram-models of text retrieval.
Now we explain the categorization algorithms and the choice of their components.

4.2.1 The method

The main steps of our method are:

- Detection and description of image patches.
- Assigning patch descriptors to a set of predetermined clusters (a vocabulary) with a vector quantization algorithm.
- Constructing a bag of keypoints, which counts the number of patches assigned to each cluster.
- Applying a multi-class classifier, treating the bag of keypoints as the feature vector, and thus determine which category or categories to assign to the image.

Ideally these steps are designed to maximize classification accuracy while minimizing computational effort. Thus, the descriptors extracted in the first step should be invariant to variations that are irrelevant to the categorization task (image transformations, lighting variations and occlusions) but rich enough to carry enough information to be discriminative at the category level. The vocabulary used in the second step should be large enough to distinguish relevant changes in image parts, but not so large as to distinguish irrelevant variations such as noise.

We refer to the quantized feature vectors (cluster centres) as "features" by analogy with "keywords" in text categorization. However, in our case "words" do not necessarily have a repeatable meaning such as "eyes", or "car wheels", nor is there an obvious best choice of vocabulary. Rather, our goal is to use a vocabulary that allows good categorization performance on a given training dataset. Therefore the steps involved in training the system allow consideration of multiple possible vocabularies:

- Detection and description of image patches for a set of labeled training images.
- Constructing a set of vocabularies: each is a set of cluster centres, with respect to which descriptors are vector quantized.
- Extracting bags of features for these vocabularies.
- Training multi-class classifiers using the bags of features as feature vectors.
- Selecting the vocabulary and classifier giving the best overall classification accuracy.

We now discuss the choices made for each step in more detail.

4.2.2 Feature extraction

For this project we decided to use SURF (*SpeedUp Robust Features*) which has shown fast and accurate performance in several situations.

SURF [4] also known as approximate SIFT [3], employs integral images and efficient scale space construction to generate keypoints and descriptors very efficiently. SURF uses two stages namely keypoint detection and keypoint description[4]. In the first stage, rather than using DoGs as in SIFT, integral images allow the fast computation of approximate Laplacian of Gaussian images using a box filter. The computational cost of applying the box filter is independent of the size of the filter because of the integral image representation. Determinants of the Hessian matrix are then used to detect the keypoints. So SURF builds its scale space (Figure 4.11) by keeping the image size the same and varies the filter size only.

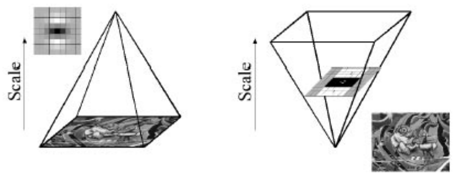


Figure 4.5: *Scale space on SURF.*

The first stage results in invariance to scale and location. In the final stage, each detected keypoint is first assigned a reproducible orientation. For orientation, Haar wavelet responses in x and y directions are calculated for a set of pixels within a radius of 6σ where σ refers to the detected keypoint scale. The SURF descriptor is then computed by constructing a square window centered around the keypoint and oriented along the orientation obtained before. This window is divided into 4 x 4 regular sub-regions and Haar wavelets of size 2σ are calculated within each sub-region (Figure 4.12). Each sub-region contributes 4 values thus resulting in 64D descriptor vectors which are then normalized to unit length. The resulting SURF descriptor is invariant to rotation, scale, contrast and partially invariant to other transformations. Shorter SURF descriptors can also be computed however best results are reported with 64D SURF descriptors.

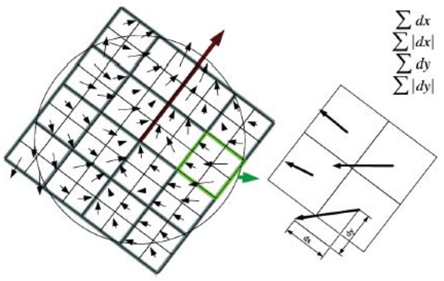


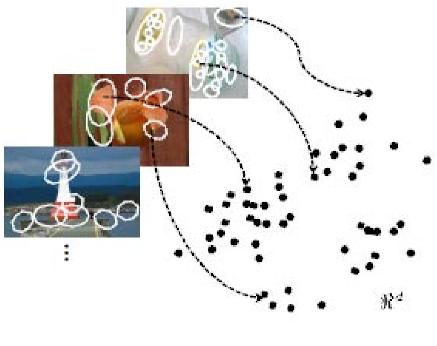
Figure 4.6: *SURF Descriptor.*

4.2.3 Visual vocabulary construction

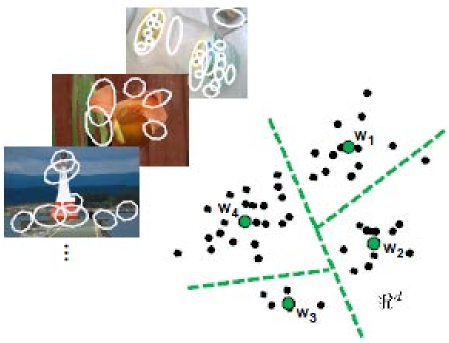
In our method, the vocabulary is a way of constructing a feature vector for classification that relates “new” descriptors in query images to descriptors previously seen in training (Figure 4.7a). One extreme of this approach would be to compare each query descriptor to all training descriptors: this seems impractical given the huge number of training descriptors involved. Another extreme would be to try to identify a small number of large “clusters” that are good at discriminating a given class: for instance [7] operates with 6 parts per category. In practice the best tradeoffs of accuracy and computational efficiency are obtained for intermediate sizes of clustering.

Most clustering or vector quantization algorithms are based on iterative square-error partitioning or on hierarchical techniques. Square-error partitioning algorithms attempt to obtain the partition which minimizes the within-cluster scatter or maximizes the between-cluster scatter. Hierarchical techniques organize data in a nested sequence of groups which can be displayed in the form of a dendrogram or a tree. They need some heuristics to form clusters and hence are less frequently used than square-error partitioning techniques in pattern recognition.

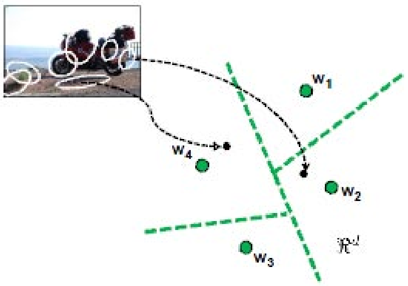
We use the simplest square-error partitioning method: k-means [25]. This algorithm proceeds by iterated assignments of points to their closest cluster centers and recomputation of the cluster centers (Figure 4.7b). Two difficulties are that the k-means algorithm converges only to local optima of the squared distortion, and that it does not determine the parameter k. There exist methods allowing to automatically estimating the number of clusters. However, in the present case we do not really know anything about the density or the compactness of our clusters. Moreover, we are not even interested in a "correct clustering" in the sense of feature distributions, but rather in accurate categorization. We therefore run k-means several times with different number of desired representative vectors (k) and different sets of initial cluster centers. We select the final clustering giving the best tradeoff between the risk in categorization and the performance.



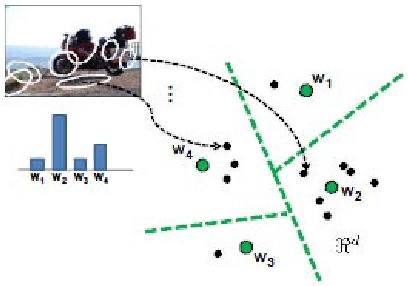
(a)



(b)



(c)



(d)

Figure 4.7: *(a) A large corpus of representative images are used to populate the feature space with descriptor instances. The white ellipses denote local feature regions, and the black dots denote points in some feature space. (b) Next the sampled features are clustered in order to quantize the space into a discrete number of visual words. The visual words are the cluster centers, denoted with the large green circles. The dotted green lines signify the implied Voronoi cells based on the selected word centers. (c) Now, given a new image, the nearest visual word is identified for each of its features. This maps the image from a set of high-dimensional descriptors to a list of word numbers. (d) A bag-of-features histogram can be used to summarize the entire image. It counts how many times each of the visual words occurs in the image.*

4.2.4 Categorization

Once descriptors have been assigned to clusters (Figure 4.7c) to form feature vectors (Figure 4.7d), we reduce the problem of generic visual categorization to that of multi-class supervised learning, with as many classes as defined visual categories. The categorizer performs two separate steps in order to predict the classes of unlabeled images: training and testing. During training, labeled data is sent to the classifier and used to adapt a statistical decision procedure for distinguishing categories. Among many available classifiers, we used the SVM since is it often known to produce state-of-the-art results in high dimensional problems.

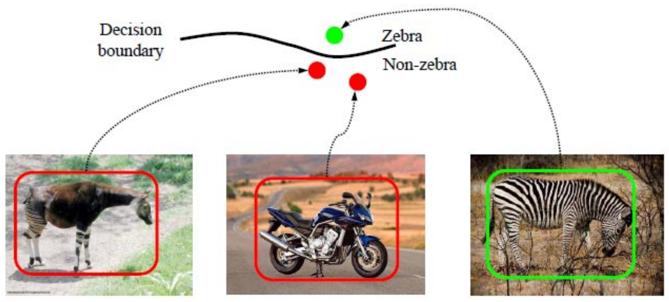


Figure 4.8: *Example of zembra classification, as we can see it is a yes/no classifier.*

The SVM classifier finds a hyperplane which separates two-class data with maximal margin [26]. The margin is defined as the distance of the closest training point to the separating hyperplane (Figure 4.9). For given observations **X**, and corresponding labels **Y** which takes values ±1, one finds a classification function:

$$f(x) = sign(w^t x + b) \tag{4.1}$$

where w, b represents the parameters of the hyperplane.

Data sets are not always linearly separable. The SVM takes two approaches to this problem. Firstly it introduces an error weighting constant **C** which penalizes misclassification of samples in proportion to their distance from the classification boundary. Secondly a mapping Φ is made from the original data space of **X** to another feature space. This second feature space may have a high or even infinite dimension. One of the advantages of the SVM is that it can be formulated entirely in terms of scalar products in the second feature space, by introducing the *kernel*

$$K(u, v) = \Phi(u) \cdot \Phi(v) \tag{4.2}$$

Both the kernel **K** and penalty **C** are problem dependent and need to be determined by the user. In the kernel formulation, the decision function can be expressed as

$$f(x) = sign\left(\sum_i y_i \alpha_i K(x, x_i) + b\right)$$

where x_i are the training features from data space X and y_i is the label of x_i.
Here the parameters α_i are typically zero for most i. Equivalently, the sum can be taken only over a select few of the x_i. These feature vectors are known as *support vectors*. It can be shown that the support vectors are those feature vectors lying nearest to the separating hyperplane. In our case, the input features x_i are the binned histograms formed by the number of occurrences of each keypoint v_i from the vocabulary **V** in the image I_i.

In order to apply the SVM to multi-class problems we take the one-against-all approach. Given an m-class problem, we train m SVM's, each distinguishes images from some category i from images from all the other *m-1* categories j not equal to i. Given a query image, we assign it to the class with the largest SVM output.

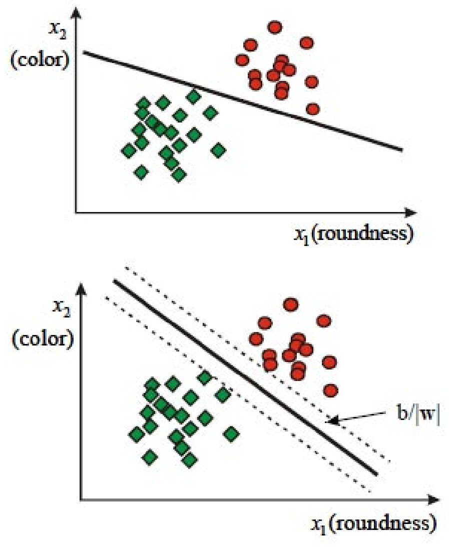


Figure 4.9: *The separation of hyperplane is ambiguous, for this reason it is used a margin that leat to a more accurate separation.*

4.3 Object detection

The branch of object detection is a natural evolution of object recognition, because if it is true which the task of recognition is classify one or more classes on a image, from a list known a priori, it is also true which does not compute spatial information of these classes of objects. In poor words if the object recognition methods answer to the question "which class of object is present on this image?", object detection answer to the question "which is the area that contains this object?". As an evolution, it inherits the problems and difficulties previously found, in fact object class detection in images is challenging because of two problems. First, objects exhibit large variations due to intra-class variability, illumination changes, etc. Second, objects may appear anywhere in an image with unknown scale, and need to be localised.
For this part we have decided to use a variation of the costellation method proposed by Lowe [3].

A **Costellation Method** approach is a single view object detection and recognition system with some interesting characteristics for mobile robots, most significant of which are the ability to detect and recognize objects at the same time in an unsegmented image. Another interesting features is the best-bin-first algorithm [27] used for approximate fast matching, which reduces the search time by two orders of magnitude for a database of 100,000 keypoints for a 5% loss in the number of correct matches. The first stage of the approach consists on matching individually the SIFT descriptors of the features detected in a test image to the ones stored in the object database using the Euclidean distance. False matches are rejected if the distance of the first nearest neighbor is not distinctive enough when compared with that of the second. In Figure 4.10, the matching features between a test and model images can be seen. The presence of some outliers can also be observed. Once a set of matches is found, the generalized Hough transform is used to cluster each match of every database image depending on its particular transformation (translation, rotation and scale change). Although imprecise, this step generates a number of initial coherent hypotheses and removes a notable portion of the outliers that could potentially confuse more precise but also more sensitive methods. All clusters with at least three matches for a particular image are accepted, and fed to the next stage: the Iterative Reweighed Least Squares is used to improve the estimation of the affine transformation between the model and the test images.

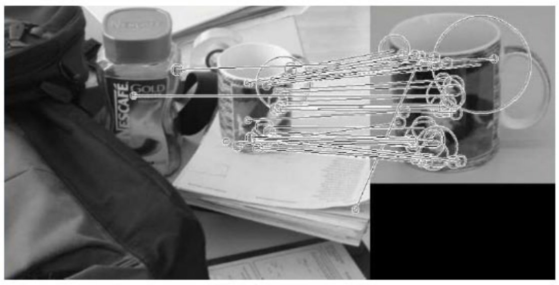


Figure 4.10: *Matching stage in the Lowe object recognition method.*

As we say, costellation method is good for a single view object detection and recognition, but we need of a robust system, which must be able to detect object from different angle of views. Then we modified the basic method with some tricks, first we use as many structures as classes are, the right structure will be selected using the result of previous section, second for each class we save images of the same object from different angle of views. With these two simple variations we still have a fast search structure and also we can use 2D information to detect a 3D object.

4.3.1 The method

The main steps of our method are:

- Detection and description of image patches for a each set of classes of images.
- Constructing a set of fast seach structures: each is a multiview container.
- Selecting the right structure, corrispondent to a class of objects.
- Search the best corrispondence between the features of a query frame and the ones stored.
- Filtering the results to discard false corrispondance and obtained the best match.
- Return the corners of the bounding area of the object detected on the frame.

We now discuss the choices made for each step in more detail.

4.3.2 Feature extraction

For this project we decided to use SURF (SpeedUp Robust Features) which has shown fast and accurate performance in several situations.

SURF also known as approximate SIFT, employs integral images and efficient scale space construction to generate keypoints and descriptors very efficiently. SURF uses two stages namely keypoint detection and keypoint description. In the first stage, rather than using DoGs as in SIFT, integral images allow the fast computation of approximate Laplacian of Gaussian images using a box filter. The computational cost of applying the box filter is independent of the size of the filter because of the integral image representation. Determinants of the Hessian matrix are then used to detect the keypoints. So SURF builds its scale space (Figure 4.11) by keeping the image size the same and varies the filter size only.

The first stage results in invariance to scale and location. In the final stage, each detected keypoint is first assigned a reproducible orientation. For orientation, Haar wavelet responses in x and y directions are calculated for a set of pixels within a radius of 6σ where σ refers to the detected keypoint scale. The SURF descriptor is then computed by constructing a square window centered around the keypoint and oriented along the orientation obtained before. This window is divided into 4×4 regular sub-regions and Haar wavelets of size 2σ are calculated within each sub-region (Figure 4.12). Each sub-region contributes 4 values thus resulting in 64D descriptor vectors

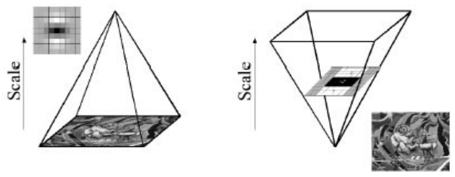


Figure 4.11: *Scale space on SURF.*

which are then normalized to unit length. The resulting SURF descriptor is invariant to rotation, scale, contrast and partially invariant to other transformations. Shorter SURF descriptors can also be computed however best results are reported with 64D SURF descriptors.

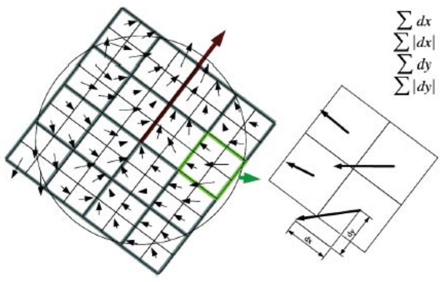


Figure 4.12: *SURF Descriptor.*

4.3.3 Search structure construction

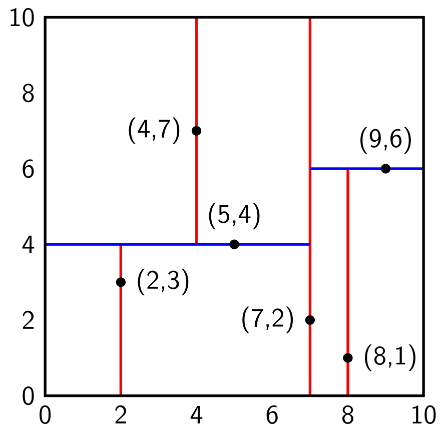
When we finished to extract the descriptors from the training set of images, becomes necessary build a structure to store them and when needs, search inside quickly. For this reason we decide to use a variation of *kd-tree* [28] called *randomized kd-tree* [29], which improves the general performance and solve the problem of classical kd-tree.

Given a set of n data points $\mathcal{X} = \{x_1, ..., x_n\}$ with $x_i \in \mathcal{R}^k$ being a k-dimensional point, the goal is to build a tree structure to index these data points so that the nearest neighbors of a query vector x_q can be fast found.

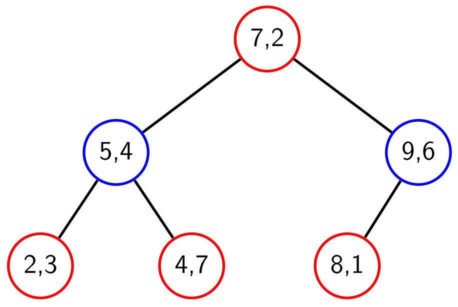
Construction. A kd-tree (k-dimensional tree) is a special case of binary space partitioning trees, which is constructed in a recursive manner. At the root, the data points are split into two halves by a *partition hyperplane*. Then each half is assigned to one child node, and is recursively split in the same manner to create a balanced binary tree. The leaf node may contain a single point or more than one points in different implementations. In this way, each node in the constructed kd-tree corresponds to a cell in $\mathcal{R}^k$, bounded by a set of partition hyperplanes (Figure 4.13).

Here, a partition hyperplane is perpendicular to a *partition axis* and decided by a *partition value*. The partition axis in the conventional kd-tree is the coordinate axis with the greatest variance, and the partition value is the median of the projections of the data points along the partition axis.

Search. With the classical approach, the search time may become almost linear then is inefficient for big size of n. The reason of the efficiency diminishing in high dimensional data is that searching a kd-tree usually takes a lot of time to backtrack through the tree to find the optimal solution. By limiting the amount of backtracking, the certainty of finding the true nearest neighbors is sacrificed and replaced with a probabilistic performance. Recent



(a) *The resulting space decomposition.*



(b) *The resulting kd-tree.*

Figure 4.13: *A simple example of the creation of a kd-tree from a set of points:(2, 3), (5, 4), (9, 6), (4, 7), (8, 1), (7, 2).*

research has therefore aimed at increasing the probability of finding the true nearest neighborwhile keeping backtracking within reasonable limits. For instance a priority search [30] was proposed for fast approximate nearest neighbor search, and for its performance we have decided to use it.

To find the nearest neighbor of a query point, a top-down searching procedure is performed from the root to the leaf nodes. At each internal node, it is required to inspect which side of the partition hyperplane the query point lies in, then the associated child node is accordingly accessed. The descent down process requires $\log_2 n$ (the height of the kd-tree) comparisons to reach a leaf node. The data point associated with the first leaf node is the first candidate for the nearest neighbor, which is not necessarily the true nearest neighbor. It must be followed by a process of backtracking, or iterative search, in which other leaf nodes are searched for better candidates. The widely used scheme with high chance to find the true nearest neighbor early is *priority search* based on a *priority queue*, in which the cells are searched in the order of their distances from the query point. The search terminates when there are no more cells within the distance defined by the best point found so far.

The nearest neighbor (NN) search [31] in the high-dimensional case [27] may require visiting a very large number of nodes, and even the process costs linear time. Therefore, alternatively, an approximate nearest neighbor (ANN) search [3][27] is usually performed, through an advanced search termination scheme, e.g., after searching a specified number of nodes, or if the distance from the closest cell to the query exceeds $\delta = d(x_p, x_q)/(1 + \epsilon)$, where x_q is the query point, x_p is the NN found so far, and ϵ is a positive termination parameter, which guarantees that no subsequent point to be found can be closer to q than δ. In this manner, the search is guaranteed to have some certain probability to obtain the true nearest neighbor.

Search on multiple trees. With multiple trees, we need to expand the concept of a priority search on a single tree. Conceptually, searching m trees with a limitation of n search nodes is simply searching each tree for n/m nodes. This can be easily implemented by searching trees sequentially. However, this is not optimal, and besides does not scale to a case where we impose no limitation on the number of searched nodes (searching for the true nearest neighbour) because we would already nd the best solution and would not be required to search extra trees.

After descending each of the trees to nd an initial nearest-neighbour candidate, it is selected the best one from all the trees. Then it is pooled the node ranking by using one queue to order the nodes from all m trees. In this way, nodes are not only ranked against other nodes within the same tree, but also ranked against other nodes in all trees. As a result, nodes from all the trees are searched in the order of their distance from the query point simultaneously.

Randomized KD-Tree. This method of doing independent multiple searches is to create multiple kd-trees with different orientations. Suppose we have a data set $\mathcal{X} = \{x_1\}$. Creating kd-trees with different orientations simply means creating kd-trees from rotated data Rx_i, where R is a rotation matrix. A principal (a regular) kd-tree is one created without any rotation, $R = I$. By rotating the data set, the resulting kd-tree has a different structure and covers a different set of dimensions compared with the principal tree. Instead of explicitly rotating the tree, using randomness on parameters can also alter the tree structure.

In accordance with the principle of selecting a partitioning value in the dimension with the greatest variance, it is considered creating extra search trees with the following idea. In the standard kd-tree, the dimension which the data is divided is the one in which the data has the greatest variance. In reality, data variance is quite similar in many of the dimensions, and it does not make a lot of difference in which of these dimensions the subdivision is made. It is adopted the strategy of selecting at random (at each level of the tree) the dimension in which to subdivide the data. The choice is made from among a few dimensions in which the data has high variance. Multiple trees are constructed in this way, different from each other in the choice of subdivision dimensions. In this randomisation strategy, in contrast to rotating the data explicitly, the data set stays in the original space, thus, saving some computation on data projection while building the tree.

4.3.4 Matching

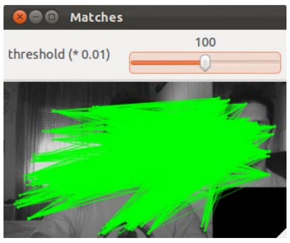
Once features have been extracted from all n images (linear time), and stored in multiple randomized kd-tree, they must be matched with query features. Then are required some techniques to reduce false corrispondences and outiers.

Ratio of distances. The best candidate match for each feature is found by identifying its nearest neighbor in the database of features from training images. The nearest neighbor is defined as the feature with minimum Euclidean distance for the invariant descriptor vector.

However, many features from an image will not have any correct match in the training database because they arise from background clutter or were not detected in the training images. Therefore, it would be useful to have a way to discard features that do not have any good match to the database. A global threshold on distance to the closest feature does not perform well, as some descriptors are much more discriminative than others. A more effective measure is obtained by comparing the distance of the closest neighbor to that of the second-closest neighbor. This measure performs well because correct matches need to have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching. For false matches, there will likely be a number of other false matches within similar distances due to the high dimensionality of the feature space. We can think of the second-closest match as providing an estimate of the density of false matches within this portion of the feature space and at the same time identifying specific instances of feature ambiguity.

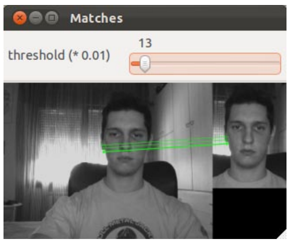
The probability density functions for correct and incorrect matches are shown in terms of the ratio of closest to second-closest neighbors of each keypoint. For our object detection implementation, we reject all matches in which the distance ratio is greater than 0.8, which eliminates 90% of the false matches while discarding less than 5% of the correct matches. When the index of the best matched view is found, a second control is applied to the descriptors with a more strict threshold, this guarantee accurate results.

Homography. Since homography simplifies the relationship between correspondences and its computation cost is relatively small, homograhpy is used extensively in computer vision systems and plays an important part in multiple view geometry.



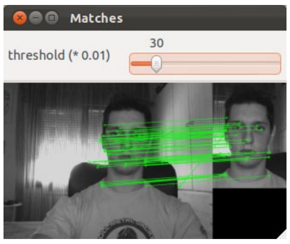
threshold (* 0.01) 100

(a)



threshold (* 0.01) 13

(b)



threshold (* 0.01) 30

(c)

Figure 4.14: *(a) Show a high number of false matches gives by a low value of distance. (b) Show a low number of positive matches give by a high value of distance. (c) Show a right number of matches give by the value suggested by Lowe.*

Single view geometry indicates how the world scene is imaged into the final image we see, which is also referred as camera model. A 3D/2D correspondence (x, X), where x and X are both in homogeneneous form and the world coordinate is defined on the plane 3D point X is laid on, should satisfy the following equation:

$$x = PX = K[\ R \mid t\ ] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = K[r_1, r_2, t] = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = HX$$

where P is known as projective camera matrix, K is the camera's intrinsic matrix, and R and t are the pose matrix, which is referred as the rotation matrix and translation vector respectively. r_i is the i-th column of R. H is the homography matrix relating 3D/2D correspondences on a world plane.

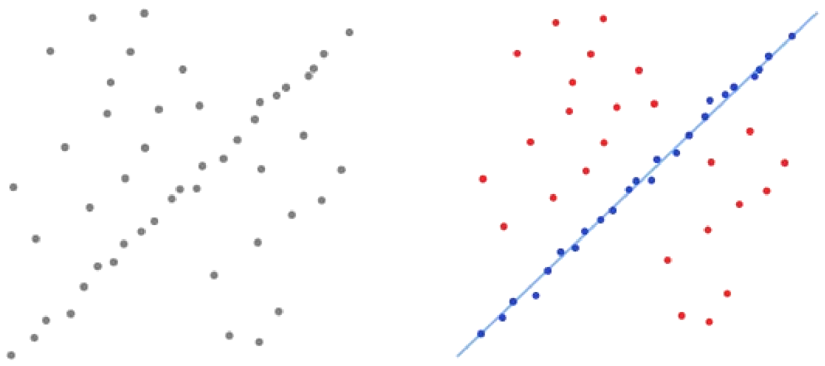
For a 2D/2D correspondence (x, x'), when it refers to 3D point on a plane, it will satisfy:

$$x' = Hx \quad H^{-1}x' = x$$

where H is the homography matrix relating two views of planar target. Unlike fundamental matrix F in epipolar geometry, which images a point to a line, H describes a "Point-Point" imaging. H is a full rank matrix. Four point correspondences can be used to get a unique solution of H up to a scale.

Current vision registration methods can be classified as model based and move-matching based. When a model of the target scene is defined, 3D/2D feature correspondences can be extracted for every frame. Using these correspondences, though the features would be of various kinds (points, lines, free curves, etc.), cost functions or equations based on the pose variables are setup, because these correspondences must satisfy certain camera model. Move-matching refers to the multiple view geometry among several adjacent frames.

The acquired correspondence set can be denoted as P. Surely, P has outliers. **RANSAC** (RANdom SAmple Consensus) algorithm can help the computing of homography beginning with P, known as putative correspondence set. RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, as depicted in Figure 4.15. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability. The final results of this method are a set of inliers and a reliable H.



(a) *A data set for which a line has to be fitted.*

(b) *Fitted line with RANSAC; outliers have no influence on the result.*

Figure 4.15: *A simple overview of how RANSAC algorithm works.*

4.4 Implementation details

To implement the recognition and detection phase, we have created two different classes into a main package called `2d_processing`. The first, namely `object_recognition`, which receive the RGB image obtained from the point cloud. The second class, the `object_detector`, performs the remaining operations, using the result of previous node to "activate" the right structure. Both parts are connected and implemented under ROS.

4.4.1 Object recognition

To recognize the objects we have to pass a folder containing a training set of images, the number of training images for each class and the number of classes, if this step is just done, it is possible pass only the files obtained from a previous step of training. In both case, the program will return an index with the corrispondent class recognized on the scene.

The implementation of Bag of Features [32] is based on the one available in the OpenCV library, the same for SURF features, while for the SVM has been used the library opensource LIBSVM [33]. To build this part we followed the implementation described in [34].

4.4.2 Object detection

As in the previous step we have to pass a file setting which contains the links of the objects' database, after that are built a vector of search structures in equal number of classes.

In each of these images are extracted and computed SURF descriptors, then these values are stored in the search structures, based on FLANN [35] library of Muja and Lowe, which used 4 randomized kd-tree that will be searched in parallel and recursively traversed 64 times; a higher value for this parameter would give better search precision, but also take more time. The training step can be skipped if it has already done and saved the results.
The right structure will be selected by the index computed on the recognition step, then a k-nn search is applied to find the best descriptors matching, which will be saved on a dedicated vector.

As described in the previous section, two refinement steps are applied to this vector, with a NN ratio test on the 2 NN estimated by the search, and the homography, to delete false corrispondences and outliers respectively. At the end, four coordinates' point are obtained (Figure 4.16) by method `getCorners` which used the homography matrix and the matched points to compute a perspective transformation to compute the distorced corners of the object.

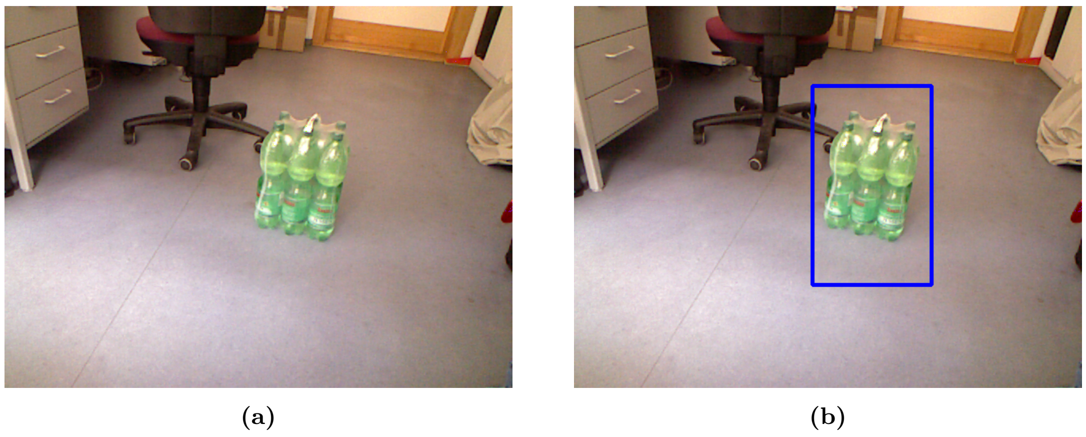


Figure 4.16: *(a) Microsoft Kinect RGB image. (b) ROI obtained.*

Chapter 5

3D Module

The estimation of the 3D orientation and location of an object is a fundamental task in computer vision. The combination of these two values is referred to as the pose of an object, even though this concept is sometimes used only to describe the orientation. This information can then be used, for example, to allow a robot to manipulate an object [36][37] or to avoid moving into the object [38][39] (i.e. robot navigation). However, pose estimation of objects in complex and cluttered scenes is challenging because the appearance of an object in a 2D image is sensitive to illumination, shadows, and lack of proportion and distance knowledge. In addition, the objects may partially occlude each other. Some of these problems can be avoided using depth information. Since recent depth acquisition systems have reached a high level of reliability, the use of RGB-D data, to overcome some of these problems is promising.
To achieve this goal we need first to extract the object from the point clouds, which increase the size of our problem. Previous step comes in our help, with the corners of the ROI which contains the object, we can extract only a box of points in corrispondence with that values. With this intermediate step, we drastically reduce the number of data to compute, then at this point we segment the point clouds obtained to isolate the object from the background, groundfloor, and so on. The last stage leads to the pose of the object in the space.

This chapter is organized as follows, first will be presented common techniques for the point clouds, then we will move to the core of 3D module or rather how are structured and built the point cloud segmentation and pose methods.

5.1 Point cloud processing

By denition, we will refer to a collection of 3D points as a point cloud structure $\mathcal{P}$. Point clouds represent the basic input data format for 3D perception systems, and provide discrete, but meaningful representations of the surrounding world. Without any loss of generality, the $\{x_i, y_i, z_i\}$ coordinates of any point $p_i \in \mathcal{P}$ are given with respect to a fixed coordinate system, usually having its origin at the sensing device used to acquire the data. This means that each point p_i represents the distance on the three dened coordinate axes from the acquisition viewpoint to the surface that the point has been sampled on.

5.1.1 Data acquisition

Though there are many ways of measuring distances and converting them to 3D points, in the context of mobile robotic applications [40], the most used approaches are:

- Stereo camera;
- LIDAR or rangefinder;
- Time-of-Flight camera;

- RGB-D camera;

LIDAR (Light Detection And Ranging) is an optical remote sensing technology that can measure the distance to, or other properties of a target by illuminating the target with light, often using pulses from a laser.

LIDAR technology is being used in Robotics for the perception of the environment as well as object classification. The ability of this technology to provide three-dimensional elevation maps of the terrain, high precision distance to the ground, and approach velocity can enable safe landing of robotic and manned vehicles with a high degree of precision. 3D imaging (Figure 5.1) can be achieved using laser ranging system that applies a pulsed laser and a fast gated camera, it can also be performed using arrays of high speed detectors and modulation sensitive detectors arrays typically built on single chips using CMOS and hybrid CMOS/CCD fabrication techniques. These cameras have the disadvantage to be extremely expensive.

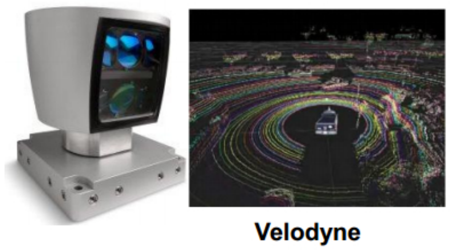


Figure 5.1: *LIDAR and 3D map.*

A Time-of-Flight Camera (ToF camera) is a range imaging camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point of the image (Figure 5.2). The time-of-flight camera is a class of scannerless LIDAR, in which the entire scene is captured with each laser or light pulse, as opposed to point-by-point with a laser beam such as in scanning LIDAR systems.
In contrast to stereo vision or triangulation systems, the whole system is very compact: the illumination is placed just next to the lens, whereas the other systems need a certain minimum base line. In contrast to laser scanning systems, no mechanical moving parts are needed. Furthermore it is very easy to extract the distance information out of the output signals of the ToF sensor, therefore this task uses only a small amount of processing power. Lastly ToF cameras are able to measure the distances within a complete scene with one shot. As the cameras reach up to 100 frames per second, they are ideally suited to be used in real-time applications. The cons of this system are given by noise coming from artificial lighting or the sun, interference with other ToF cameras and multiple reflections that leads at wrong distance.

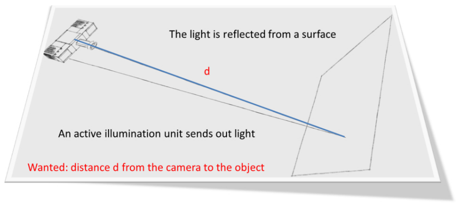

The light is reflected from a surface
d
An active illumination unit sends out light
Wanted: distance d from the camera to the object

Figure 5.2: *Principle of ToF camera.*

Furthermore in this scenario is entered a new technology called RGB-D, which merge the depth to the colours information. The large scale diffusion of this new way to collect data is due to Microsoft, first to present a cheap control for videogames named Kinect.

The majority of gestural control systems were based on the time-of-flight (ToF) method that consists in sending an infrared light, or similar, into the environment, then at same time, VGA camera recording data, where for each pixel containing RGB informations will be added the depth computed for its. Then there is a 1:1 corrispondence between RGB matrix and Depth map. The pros of this solution are: cheap coast, high framerate with hig resolution and collecting of all the interesting informations of the environment, where cons are almost the same of ToF camera.

5.1.2 Data Representation

Once a 3D point cloud dataset has been acquired using one of the methods presented in the previous section, it needs to go through a series of geometric processing steps in order to extract meaningful information that can help a robot in performing its tasks. It is the role of a mapping system therefore to process and convert the raw input point cloud data into different representations and formats based on the requirements imposed by each individual processing step.

Point cloud needs a representation which hold multiple properties per point, the definition of a point $p_i = \{x_i, y_i, z_i\}$ changes to that of $p_i = \{f_1, f_2, f_3 \cdots fn\}$, where f_i defines a feature value in a given space (color, class label, geometry, etc), thus changing the concept of a 3D point to a nD one. From these requirements, we can deduce that an appropriate I/O data storage format for a point cloud $\mathcal{P}$, would be to save each point with all its attribute values on a new line in a file, and thus have a file with n lines for the n total number of points in $\mathcal{P}$.
A fictitious example is shown bellow:

$$\begin{bmatrix} x_1 & y_1 & z_1 & r_1 & g_1 & b_1 & d_1 & \cdots \\ x_2 & y_2 & z_2 & r_2 & g_2 & b_2 & d_2 & \cdots \\ & & & & & & & \cdots \\ x_n & y_n & z_n & r_n & g_n & b_n & d_n & \cdots \end{bmatrix}$$

where x_i, y_i, z_i represent the 3D point coordinates, r_i, g_i, b_i a color associated with each point, and d_i the distance from the sensor to the surface.

To use spatial decomposition techniques such as kd-trees or octrees, and partition the point cloud data $\mathcal{P}$ into chunks, such that queries with respect to the location of the points in $\mathcal{P}$ can be answered fast. Though different from an implementation point of view, most spatial decomposition techniques can construct and give hints of a volumetric representation for a cloud $\mathcal{P}$, by enclosing all its points in boxes (also called “voxels”) with different widths.

Octree representations are also popular in the context of collision detection applications, where we can perform raycasting to the voxels encompassing $\mathcal{P}$ to discover the portions of space which are free or occluded. A big advantage of octree data structures is that they are easy to update and support point insertion and deletion almost natively.

5.2 Object extraction

Storing and processing large point cloud datasets represents one of the main bottlenecks of a 3D perception system. This section presents solutions that can be used for the processing of larger 3D point cloud datasets, in the context of point *clustering* and *segmentation* [41][42]. The concepts are somewhat similar and in some situations they can be used interchangeably. In the context of the work presented here, their main purpose is to isolate the main structure from the others, in order to obtain an accurate 3D model of our object and a lower the computational resources needed by other subsequent algorithmic steps.
Now we explain the extraction algorithms.

5.2.1 The method

The main steps of our method are:

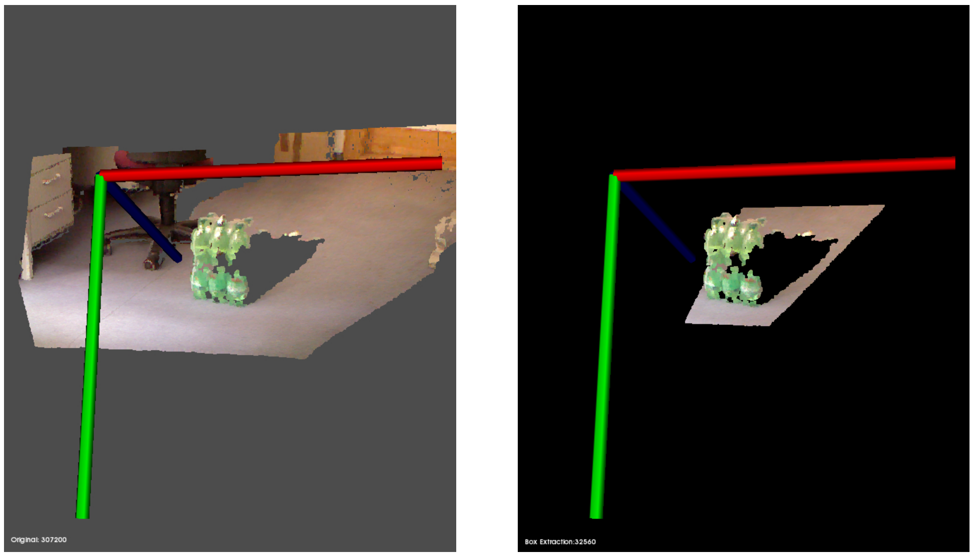
- Extracting a box of point cloud using corners previously detected.

- Downsampling of point cloud to reduce the amount of data.
- Deleting of points farther than 4m from the sensor to avoide noisy data.
- Removing of ground plane.
- Cleaning objects from residual outliers.
- Selecting the biggest cluster remaining.

We now discuss the choices made for each step in more detail.

5.2.2 Filtering

Box extraction. Exploiting the 1:1 corrispondence between pixels and points, we can simply extract the point cloud included on the corners computed previously and pass these points contained in the region to a new structure.



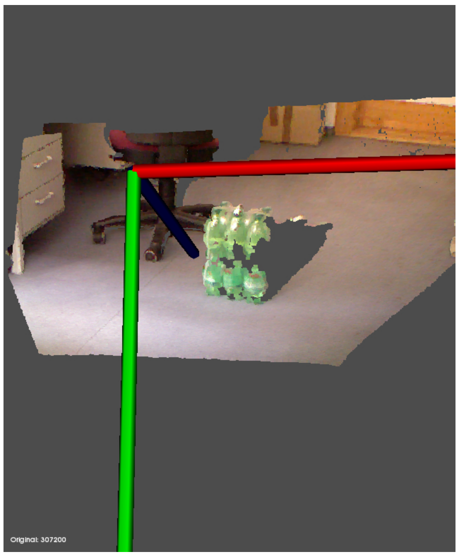
(a) **(b)**

Figure 5.3: *(a) Original point cloud. (b) Extracted point cloud.*

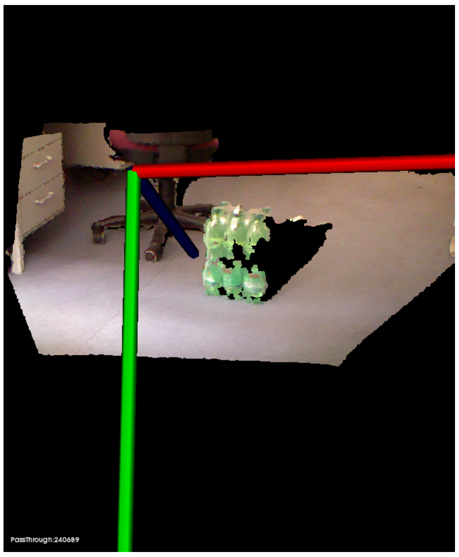
This easy step allow to use to reduce the data with an average factor of 5×, depending on the object's size (10× on Figure 5.3).

PassThrough filter. With this step, we can simply "cut-off" all the points over a threshold predefined, in our case 4m, because over this distance there is a high probability that the points are affected by noise and inaccurate values, then makes the final result not satisfactory. An example is shown in Figure 5.4.

Downsampling. For this method, are considered the spatial frequency content of the point cloud. Then using a voxel grid filter is created a 3D voxel grid (think about a voxel grid as a set of tiny 3D boxes in space) over the input point cloud data. In each voxel (i.e., 3D box), all the points present will be approximated (i.e., downsampled) with their centroid. This approach is a bit slower than approximating them with the center of the voxel, but it represents the underlying surface more accurately. Figure 5.5 show an example of adjustable grid size, $2cm$.

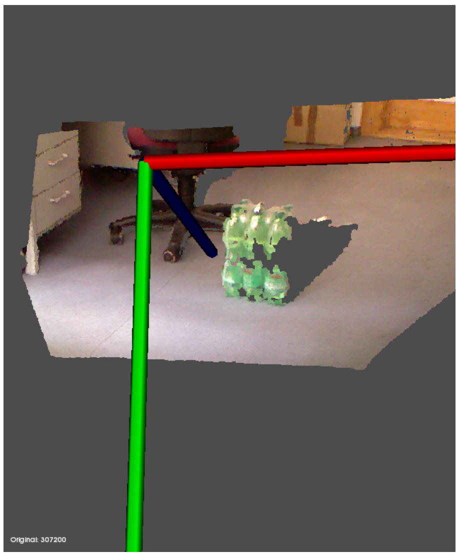


(a)

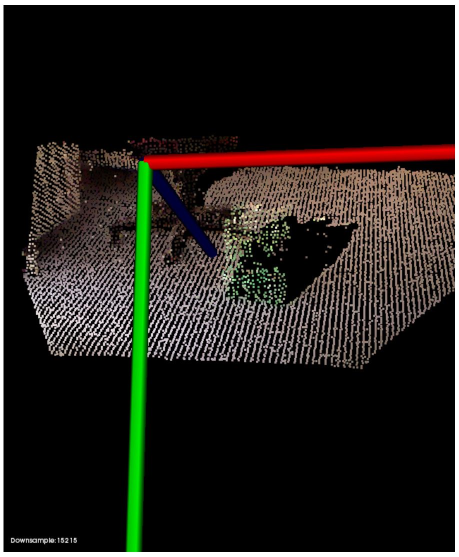


(b)

Figure 5.4: *(a) Original point cloud. (b) Filtered point cloud.*



(a)



(b)

Figure 5.5: *(a) Original point cloud. (b) Downsampled point cloud.*

Removing outliers. With helps of Figure 5.6 we can better understand how this method works. The user specifies a number of neighbors which every index must have within a specified radius to remain in the point cloud. For example if one neighbor is specified, only the yellow point will be removed from the point cloud. If two neighbors are specified then both the yellow and green points will be removed from the point cloud. With this method we can obtain with a good approximation, the removal of almost all the outliers present on the scene.

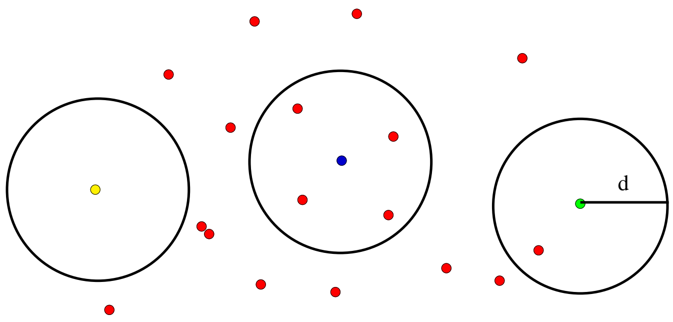


Figure 5.6: *(a) Original point cloud. (b) Outlier removed point cloud.*

5.2.3 Segmentation

Now that we have decrease the size of our point cloud, we can apply a segmentation step, to delete the ground plane, then isolate the objects present on the scene. To generalize the problem, first we estimate the surface normals, then with the result we will able to remove only the points that are perpendicular to the z axis.

Estimating normals. Though many different normal estimation methods exist, the simplest method is based on the first order 3D plane fitting as proposed by [43]. The problem of determining the normal to a point on the surface is approximated by the problem of estimating the normal of a plane tangent to the surface, which in turn becomes a least-square plane fitting estimation problem in $\mathcal{P}^k$. The plane is represented as a point x and a normal vector , and the distance from a point $p_i \in \mathcal{P}^k$ to the plane is defined as $d_i = (p_i - x) \cdot \vec{n}$. The values of x and are computed in a least-square sense so that $d_i = 0$. By taking:

$$x = \bar{p} = \frac{1}{k} \cdot \sum_{i=1}^{k} p_i$$

as the centroid of $\mathcal{P}^k$, the solution for is given by analyzing the eigenvalues and eigenvectors of the covariance matrix $\mathcal{C} \in \mathbb{R}^{3\times3}$ of $\mathcal{P}^k$.

In general, because there is no mathematical way to solve for the sign of the normal, its orientation computed via Principal Component Analysis (PCA) as shown above is ambiguous, and not consistently oriented over an entire point cloud dataset. The figure below presents these effects on two sections of a larger dataset representing a part of a kitchen environment. The right part of the figure presents the Extended Gaussian Image (EGI), also known as the normal sphere, which describes the orientation of all normals from the point cloud. Since the datasets are 2.5D and have thus been acquired from a single viewpoint, normals should be present only on half of the sphere in the EGI. However, due to the orientation inconsistency, they are spread across the entire sphere.

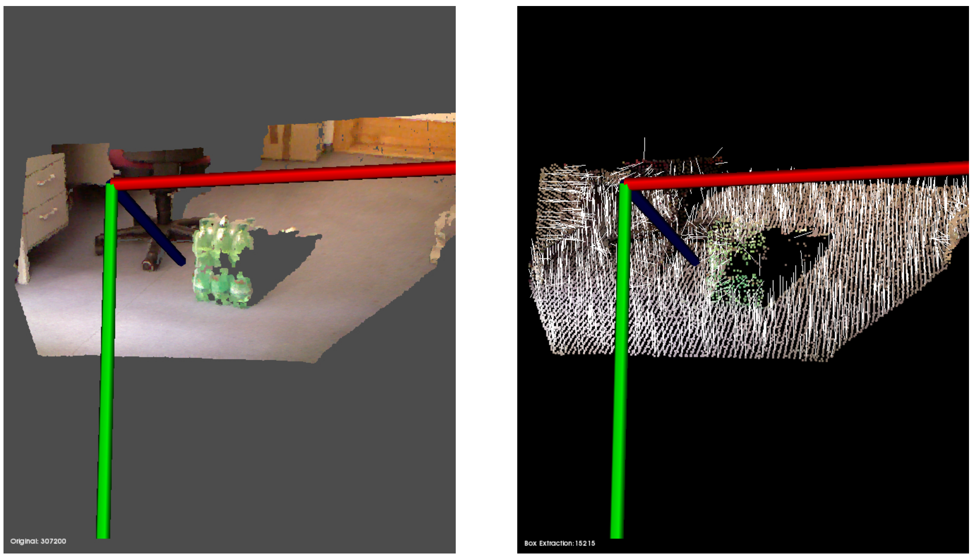
The solution to this problem is trivial if the viewpoint v_p is in fact known. To orient all normals consistently towards the viewpoint, they need to satisfy the equation:

$$\vec{n_i} \cdot (v_p - p_i) > 0$$

Plane model segmentation. In the context of the applications presented in this method, we are interested in segmenting out with a big emphasis on planar structures which represent a big part of the points in $\mathcal{P}$.

To speed up the search, the algorithm will make use of a Random Sample Consensus (RANSAC) method to generate model hypotheses. Since the model to be found represents a plane, and since three unique non-collinear points define a plane, the algorithm uses the following steps:

1. randomly select three non-collinear unique points p_i, p_j, pk from $\mathcal{P}$;



(a) **(b)**

Figure 5.7: *(a) Original point cloud. (b) Normals point cloud.*

2. compute the model coefficients from the three points ($ax + by + cz + d = 0$);
3. compute the distances from all $p \in \mathcal{P}$ to the plane model (a, b, c, d);
4. count the number of points $p^*\mathcal{P}$ whose distance d to the plane model falls between $0 \leq |d| \leq |d_t|$, where d_t represents a user specified threshold with the angle of the surface normals;.

The last step represents one of the many ways of "scoring" a specific model. Every set of points p^* is stored, and the above steps are repeated for k iterations. After the algorithm is terminated, the set with the largest number of points (inliers) is selected as the support for the best planar model found. From all $p^* \in \mathcal{P}$, the planar model coefficients are estimated in a least-squares formulation.

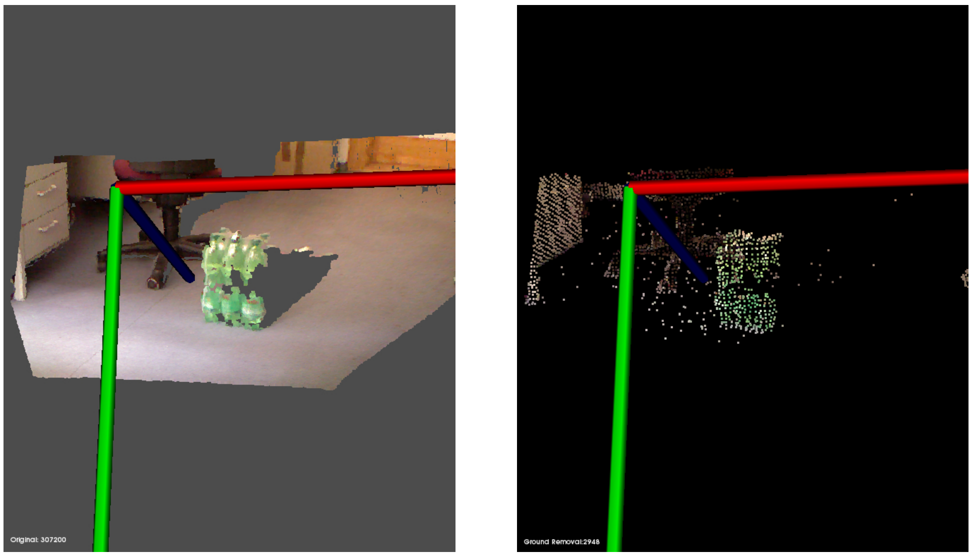
The above presented computational steps have been simplified to illustrate the basic theoretical process required to fit the geometric plane model. In reality, if we were to find the horizontal support from $\mathcal{P}$, we would need to impose additional constraints, such as the normal of the plane having to be parallel to some global world $\vec{z}$ axis which gives the the "horizontal" orientation.

After the ground removal, we apply the radial outlier removal just presented previously, which remove the residual noise on the point cloud, as we can observe in Figure 5.9.

5.2.4 Clusterization

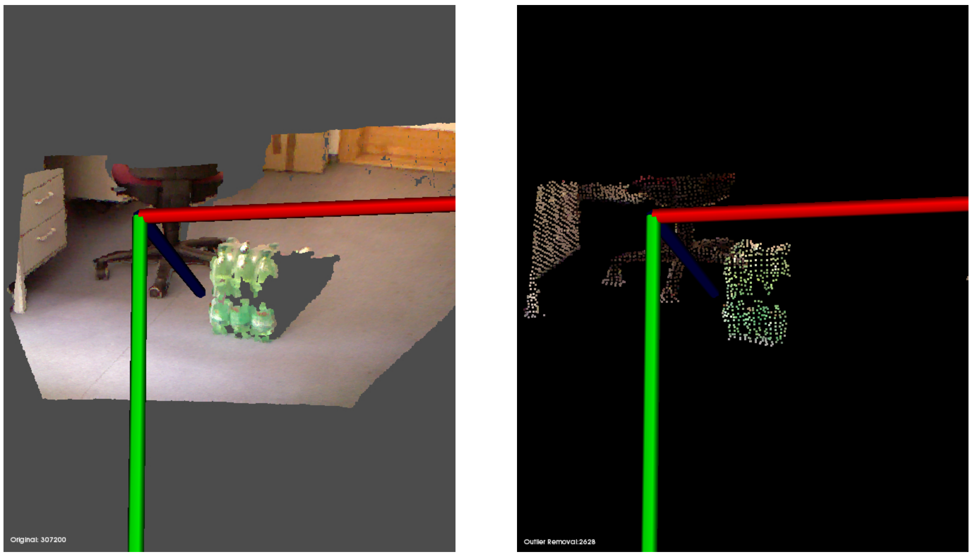
A clustering method needs to divide an unorganized point cloud model $\mathcal{P}$ into smaller parts so that the overall processing time for $\mathcal{P}$ is significantly reduced. Most of the simpler methods in this category rely on spatial decomposition techniques that find subdivisions and boundaries to allow the data to be grouped together based on a given measure of "proximity". To better explain its theoretical steps, let us assume the following application example. Given some input dataset $\mathcal{P}$ representing an environment, and a geometric model representing the supporting plane that can hold objects for pick and place manipulation tasks, find and segment the individual object point clusters lying on the plane.

To achieve this goal, the system needs to understand what is an object point cluster first and what differentiates it from another point cluster. In a more mathematical sense, a cluster is defined as follows. Let $O_i = \{p_i \in \mathcal{P}\}$ be a distinct point cluster from $O_j = \{p_j \in \mathcal{P}\}$ if:



(a) (b)

Figure 5.8: *(a) Original point cloud. (b) Point cloud after ground plane removal.*



(a) (b)

Figure 5.9: *(a) Original point cloud. (b) Point cloud after outlier removal.*

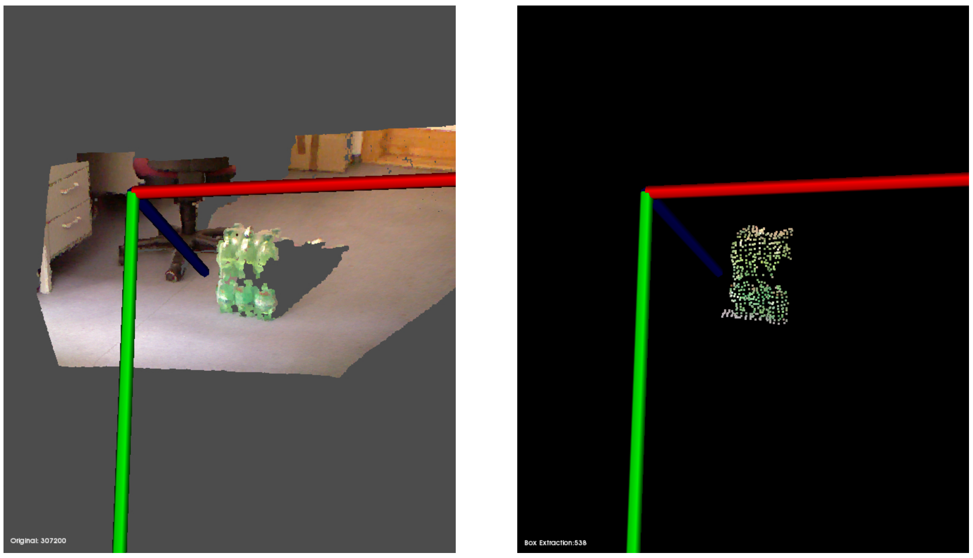
$$min\|p_i - p_j\|_2 \geq d_{th}$$

where d_{th} is a maximum imposed distance threshold. The above equation states that if the minimum distance between a set of points $p_i \in \mathcal{P}$ and another set $p_j \in \mathcal{P}$ is larger than a given distance value, then the points in p_i are set to belong to a point cluster O_i and the ones in p_j to another distinct point cluster O_j. From an implementation point of view, it is therefore important to have a notion on how this minimal distance between the two sets can be estimated. A solution is to make use of approximate nearest neighbors queries via kd-tree representations. Therefore the algorithmic steps that need to be taken could look as follows:

1. create a kd-tree representation for the input point cloud dataset $\mathcal{P}$;
2. set up an empty list of clusters $\mathcal{C}$, and a queue of the points that need to be checked $\mathcal{Q}$;
3. then for every point $p_i \in \mathcal{P}$, perform the following steps:
   - add p_i to the current queue $\mathcal{Q}$;
   - for every point $p_i \in \mathcal{Q}$ do:
     - search for the set $\mathcal{P}_i^k$ of point neighbors of p_i in a sphere with radius $r < d_{th}$;
     - for every neighbor ${p_i}^k \in \mathcal{P}_i^k$, check if the point has already been processed, and if not add it to $\mathcal{Q}$;
   - when the list of all points in $\mathcal{Q}$ has been processed, add $\mathcal{Q}$ to the list of clusters $\mathcal{C}$, and reset $\mathcal{Q}$ to an empty list;
4. the algorithm terminates when all points $p_i \in \mathcal{P}$ have been processed and are now part of the list of point clusters $\mathcal{C}$.

Applied on the remaining point cloud dataset supported by the planar model, the proposed algorithm constructs a set of separated Euclidean object clusters.

In our case we will obtain few clusters, because we supposed to extract a small area including only the our interested object, or at least few other objects. For this reason we compare all the clusters found to save only the biggest one and delete the others. This final step return as with a good approssimation the point cloud of the object detected in previous step (Figure 5.10).



(a) (b)

Figure 5.10: *(a) Original point cloud. (b) Point cloud clusterized.*

5.3 Object information

Now that we have the our object, we need to know its **size** and its *location* on the world and *orientation* with respect to the sensor (combination of position and orientation is referred to as the **pose** of an object). To simplify this step we compute an approssimation of the object, finding a bounding box that contains the point cloud previously found.

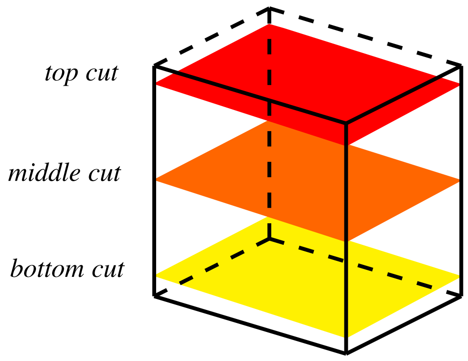


Figure 5.11: *Cutted object.*

Our approach for this method consists on "cut" our point cloud with three planes, near top, middle and bottom of our object as in Figure 5.11. At this point we project all the remaining points on the ground to obtain an imprint of the borders, then we search the minimum and maximum points for x and z coordinates, y minimum and maximum are computed on the original point cloud.

5.3.1 Size

A problem give by RGB-D data is the partial view of the environment, which do not allow to know the exact shape of the object, well to find the approximate bounding box, we use a simple geometric computation to estimate it.

The three points of the surface are used to compute the hidden corner with a two variables system, where the angles used are the complementary of the parallel faces.
In our case we choose the Cramer's rule:

$$\begin{cases} ax + bz = e \\ cx + dz = f \end{cases} \qquad \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$$

Assume ad-bc nonzero. Then, x and z can be found with Cramer's rule as

$$x = \begin{vmatrix} e & b \\ f & d \end{vmatrix} \bigg/ \begin{vmatrix} a & b \\ c & d \end{vmatrix} = \frac{ed - bf}{ad - bc}$$

and

$$z = \begin{vmatrix} a & e \\ c & f \end{vmatrix} \bigg/ \begin{vmatrix} a & b \\ c & d \end{vmatrix} = \frac{af - ec}{ad - bc}$$

At the end of this computation we are in the situation of Figure 5.12, where the red corners are given by point cloud and green ones are the computed corners.
Now to extract the size's information we need only to estimate the length of the borders for each dimension (Figure 5.13).

$$size_1 = \sqrt{(x' - x)^2 + (z' - z)^2}$$

$$size_2 = \sqrt{(x'' - x')^2 + (z'' - z')^2}$$

$$size_3 = y''' - y'$$

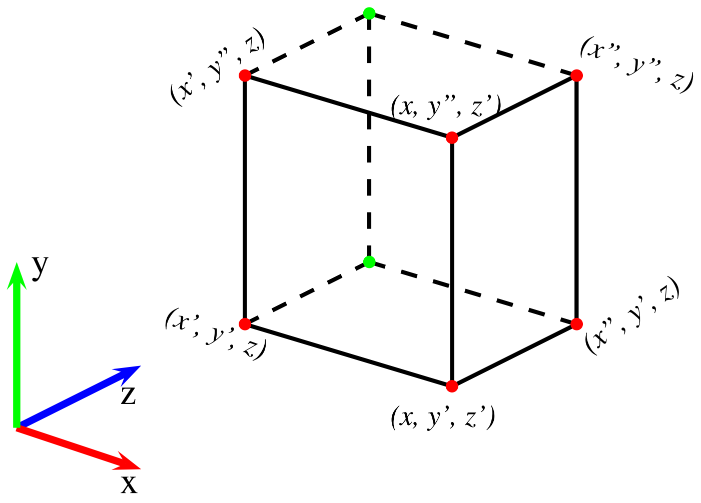


Figure 5.12: *Hidden corners computated.*

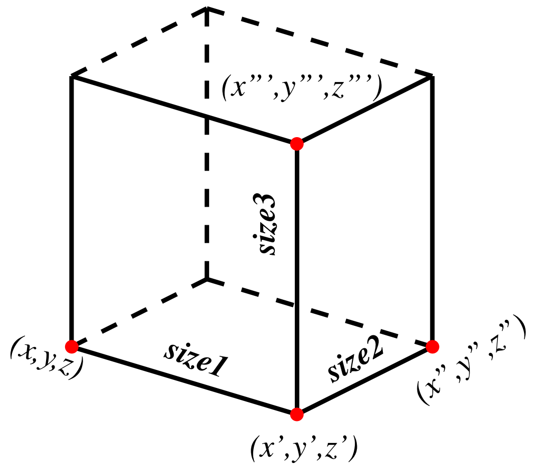


Figure 5.13: *Computed size.*

5.3.2 Pose

Now that we have an approximation of the size, we need to find the orientation relative to the sensor camera (then our wheelchair too). With the previous assumptions and point cloud, becomes easy estimate the rotation of the object, and the position in the environment.
As represent in Figure 5.14, we assume that our current positione relative to the object, defines a baseline, then with the segment $\overline{AB}$ and $\overline{BC}$, we can compute the arctangent of these values, to obtain the corrispondent angles which will represent the rotation of the object (we do not need of y angle of rotation, because the object is positioned on the ground floor or in a plane).

Last part consist on find the location of the object, but this information is just contained on point cloud data, then we only need of previous three corners and use it as reference coordinate.

5.4 Implementation details

To implement the extraction and information phase, we have created two different classes into a main package called `3d_processing`. The first, namely `object_extraction`, which receive the

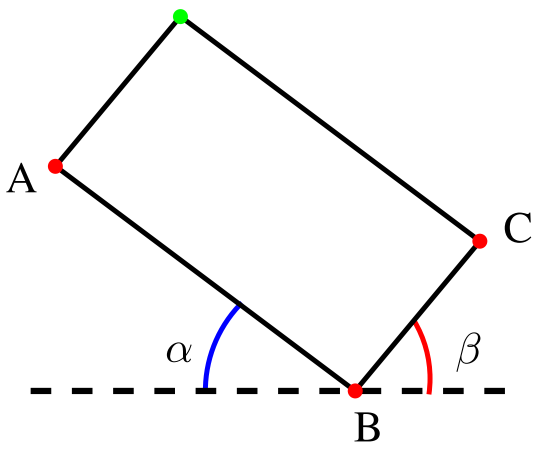


Figure 5.14: *Rotation of the object.*

corners computed from the RGB scene. The second class, the `object_information`, performs the remaining operations, using the point cloud filtered on the previous node to estimate different informations. Both parts are connected and implemented under ROS.

5.4.1 Object extraction

To extract the objects we used the corners computed one the previous step, then a method called `boxExtraction` has the task of return a point cloud of only that region while it receives as input the original point cloud (associate of the RGB image) and the vector of coordinates. Then two steps of filtering are applied with methods `downsample` which downsample the point cloud with `VoxelGrid` method with *leaf* of $1cm$ size, and `passthrough` which cut the farther points of 4m using `PassThrough`.

The segmentation is the most sensitive part, where several steps are implemented on the method `segmentation`. First are computed the normals with `NormaEstimation`, then a RANSAC estimation method fitting the plane using the normals, is used to detect planes perpendicular to y axis.

The point cloud obtained is again refined with the `removal` method which deletes points with less than 50 neighbours around a minimum radius of $50cm$ using `RadiusOutlierRemoval`.

Final step is method `cluster` which store in a kd-tree point cloud filtered in past steps and using `EuclideanClusterExtraction` start to clusterize all the points with a minimum of 100 and a maximum of 25000 points respectively. Iteratively for each indices which contained a cluster is measured the number of points, then the biggest one is saved. This one must be the object detect on 2D process with a good approximation.

5.4.2 Object information

Using the point cloud obtained in the previous step, we first estimate *min* and *max* points for each dimension with `getMinMax3D`. Point cloud is sliced with three planes of $3cm$ height near bottom, middle and top, and points obtained project in the same plane. *Min* and *max* are computed again to find the minimum and maximum values of x and z axes, while `cramerRule` [44] method estimate the hidden corners to reconstruct an appressimation of bounding box's object.

Then height is easly computed using the *min* and *max* values of y axis, where lenght and width are computed with the *Euclidean distance*. For the rotation angle is used the arctangent of the slope of the biggest face shows to the sensor.

Last part of distance from the camera, we just need to read the coordinate information (x_i, y_i, z_i) contained on the point cloud's corner found on the previous steps.

Chapter 6

System evaluation

In this chapter we show the performance for each block, and the global system in static and dynamic environment. We performed tests both while keeping our platform as static and while moving it on a predefined path.

The chapter is subdivided as following. Section 6.1 describes the testing environment used. Section 6.2 demonstrates the results of the classifier to distinguish among different classes' object. Section 6.3 analyze the performance of the detector for a single view and show how choose the best value for multiview detection, then results under these new conditions. Section 6.4 show the accuracy of the data estimated in 3D module. Section 6.5 analyzes the time necessary to perform the different operations and the frame rate reached. In the end, Section 6.6 demonstrates a real world application of the system.

6.1 Testing environment

6.1.1 Framework

The entire system is implemented in C++ within ROS, making use of highly optimized libraries for 2D computer vision, 3D point cloud processing (OpenCV and PCL) and tools (OpenNI) to create robot applications, with state of arts algorithms. Moreover is useful to developing open source and reusable software for other purposes and robots, though it becomes sometime inflexible and not parameterizable.

6.1.2 Data acquisition

The system is developed to use a RGB-D camera, then for our purpose we use the Kinect camera as mentioned before, with OpenNI to interface with it and compute/manipulate the RGB and depth informations through OpenCV and PCL.

6.1.3 Dataset

The dataset used in the test contains 256 classes of objects, with a large amount of differences, variations, occlusions and clutter backgrounds. This dataset is called Caltech-256 and it promised a high accuracy for the truth level performance of the system [45], thanks to the previous attributes mentioned and a different resolution's size of the images.

6.1.4 Hardware

All of the tests for which we report the results have been made using a desktop with an Intel Pentium Dual Core E5400 @ 2.7GHz con 4GB RAM DDR2 Bus 1066MHz, Ubuntu 11.10 32bit OS and ROS Fuerte release.

6.2 Classification performance

We present results from two experiments. In the first we explore the impact of the number of clusters on classifier accuracy and evaluate the performance of the SVM classifier. We used ten classes of very different subjects: bread maker, t-shirt, wine bottle, sneakers, watermelon, car, biliard, laptop, motorcycle, leopard. As explained it is a challenging dataset, not only because of the large number of classes, but also because it contains images with highly variable poses and significant amounts of background clutter, sometimes presence of objects from multiple classes although a large proportion of each image's area is occupied by the target category. The images have resolutions between 0.3 and 2.1 megapixels and were acquired with a diverse set of cameras. The images are color but only the luminance component is used in our method.

We used two performance measures to evaluate our multi-class classifiers.

- The confusion matrix:

$$M_{ij} = \frac{|\{I_k \in C_j : h(I_k) = i\}|}{C_j}$$

  where $i, j \in \{1, ..., N_c\}, C_j$ is the set of test images from category j and $h(I_k)$ is the category which obtained the highest classifier output for image I_k.

- The overall error rate:

$$R = 1 - \frac{\sum_{j=1}^{N_c} |C_j| M_{jj}}{\sum_{j=1}^{N_c} |C_j|}$$

In Figure 6.1 we present the overall error rates using SVM as a function of the number of clusters k. The standard-error on the maximum is in the range [1, 3]%. The error rate only improves slightly as we move from $k = 1500$ to $k = 3000$. We therefore assert that $k = 2000$ present a good trade-off between accuracy and speed.

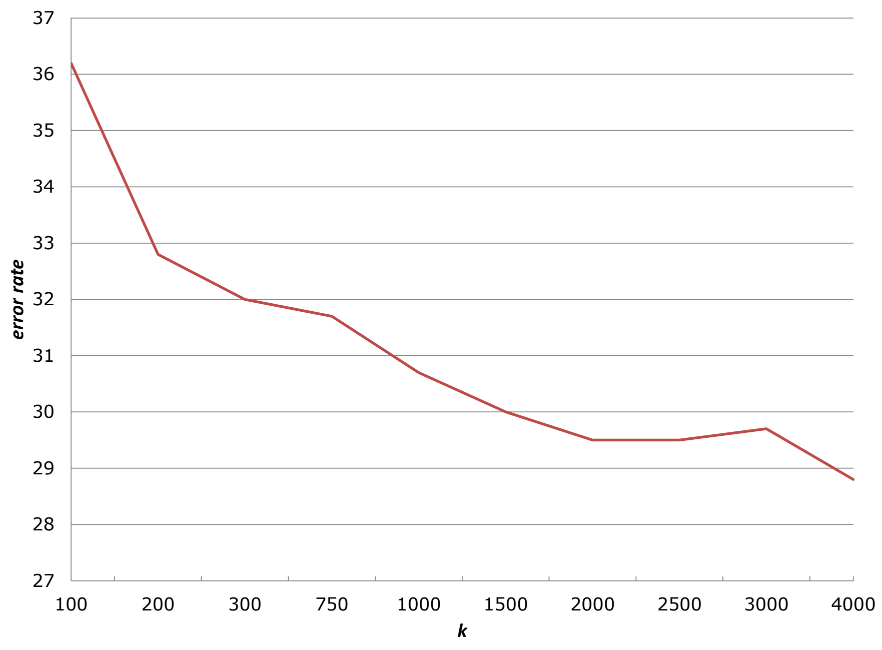


Figure 6.1: *The lowest overall error rate (percentage) found for different choices of k.*

Table 6.2 shows the performance as a function of category obtained with this k.

True classes ↓	bread maker	t-shirt	wine bottle	sneakers	watermelon	car	biliard	laptop	motorcycle	leopard
bread maker	72	3	3	10	5	0	5	2	0	0
t-shirt	7	70	8	7	3	0	2	2	2	0
wine bottle	2	10	67	2	8	2	5	2	3	0
sneakers	0	2	2	82	8	0	3	2	2	0
watermelon	2	0	0	3	78	3	3	3	2	5
car	0	0	0	0	3	92	5	0	0	0
biliard	0	8	18	3	18	2	45	5	0	0
laptop	5	7	17	7	7	3	17	37	2	0
motorcycle	2	0	5	2	7	0	7	0	78	0
leopard	0	0	2	3	5	2	0	0	2	87

Table 6.1: *Confusion matrix for the best vocabulary (k = 2000).*

Observing the single data, we can notice which bread maker gives problem on the classification of images with clutter backgrounds, this is probably due to a poor training step of this object contained in the dataset. In the t-shirt case, problems occur when the object absumed big distorsion or partial occlusion (sleeved folded or complete hidden), however it shows a good robustness both for rotation and clutter backgrounds. The wine bottle has particularly suffered the lack of "parting" between object and background, we can see that errors of classification were mainly occurred when the background was really dark, similar to the colour of the bottle. In this case apparently is not possible extract enough features to return the right classification of the image. There are no relevant notes regaring sneakers which have obtained good results, as the same as watermelon, where there are few wrong classification caused by excessive brightness and occlusions. Like for two previous classes there are not particular problems for the categorization of cars, which have shown the best overall rate. On the contrary biliards and laptops have obtained the worst results, in the first class the images which were classified with greater difficulty have been the ones with a strict frontal view of the biliards or when was present only a small part of it. Both cases have given poor features which cannot describes enough well the objects. Anyway these classes show as the classification problem is caused by the lack of complex shapes or well defined patterns, which lead to an insufficient number of features to describe the object. This consideration is confirmed by next two classes, where we obtained again results over average with well define structures like the motorcycle or texture for the leopard, which has shown very good performance despite really cluttered background.

6.3 Detection performance

Now we show the results from two experiments regarding the detector performance. In the first we explore the impact of the variation of several environmental conditions and angles. This is the most sensitive part of our project, because will help us to understand how the system will detect the object in different condition of luminosity, blurring and moreover angles of view, which is the main task for a mobile robot. We used one image where we applied all these transormation, then we detect the best value for the angle rotation to use in function of the samples of the object.

We used two performance measures to evaluate our multi-view detector.

- Precision:

$$precision = \frac{correctmatchesfound}{totalmatchesfound}$$

- Recall:

$$recall = \frac{correctmatchesfound}{correctmatchesfound}$$

then we evaluate the robustness of SURF descriptor under changes of the following conditions:

- *viewpoint.* This is measured by the roll, pitch, and yaw angles made by the normal vector of the card with the optical axis of the camera. As the SURF descriptor is known to be invariant under in-plane rotation, evaluation against a changes of the roll angle is not implemented. The 3D scene in our experiments merely consists of planar texture patterns. This simplifies the transformation required for generating the test images. In our implementation, the transformation governed by a specified pitch (or yaw) angle is equivalent to a 3×3 plane-to-plane homography.

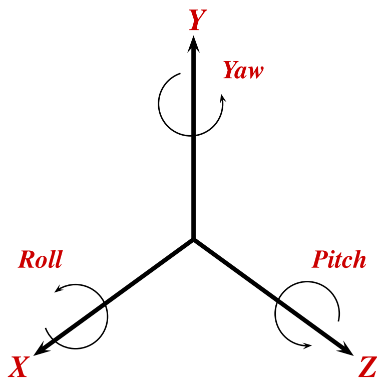


Figure 6.2: *Representation of the possible rotations, in this experiment we have used pitch and yaw, because SURF has been created to be invariant under roll rotations.*

- *illumination.* OpenCV provides some functions for brightening and darkening an image. We can see as a function with a parameter β, which governs the nonlinear transformation applied to the colour map of the image. Positive β values brighten and negative β values darken the image. Both of these operations result in a reduction of contrast within the image. The former operation gives an under-exposure effect while the latter gives an overexposure effect.

- *blurring.* The blurring operation involves convolution of the image by a Gaussian filter. The amount of blurring is determined by the standard deviation of the Gaussian filter.

To simplify the experiments, the above operations are applied independently and separately to our database of images. In real scenarios, blurring, under-exposure, and viewpoint changes can all happen together, making it much more challenging for the detector to successfully extract and match the keypoints. We have to consider the setting of our experiments to provide the best performance possible for the descriptor.

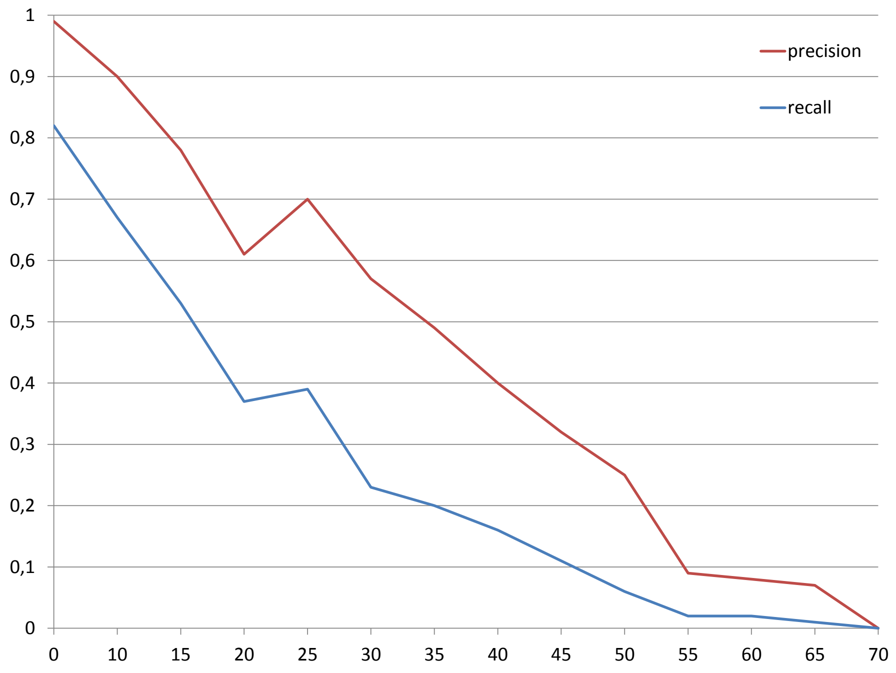


Figure 6.3: *Average precision and recall of the descriptor versus change of pitch angle for 10 test images. The vertical bars denote one standard deviation about the mean.*

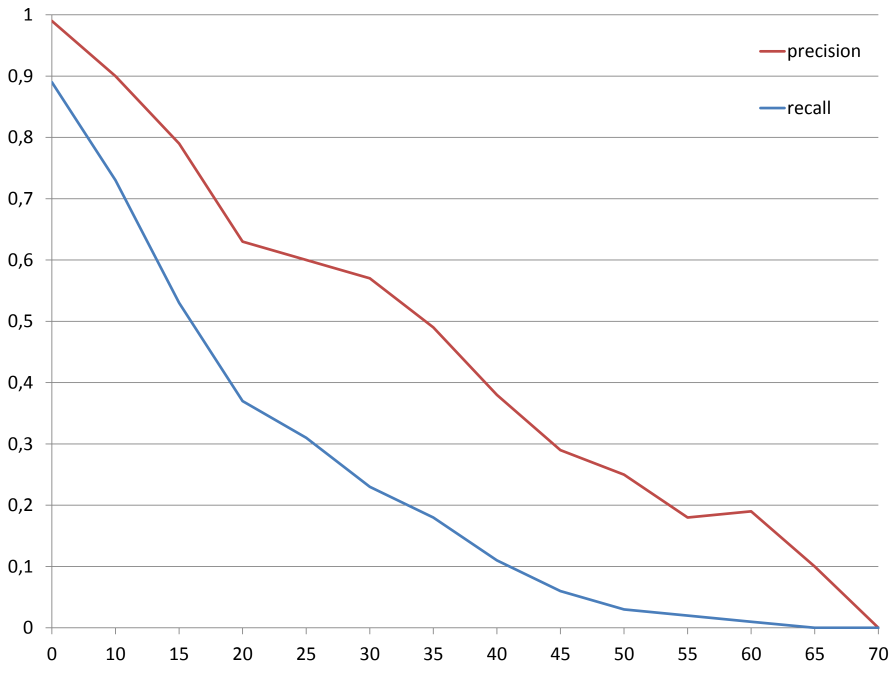


Figure 6.4: *Average precision and recall of the descriptor versus change of yaw angle for 10 test images.*

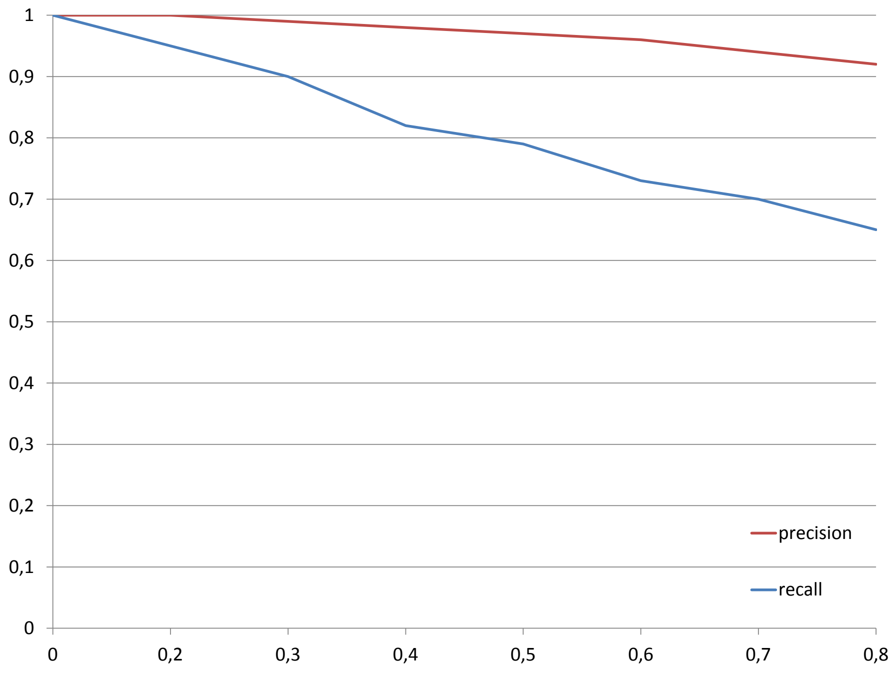


Figure 6.5: *Average precision and recall of the descriptor versus change of illumination for 10 test images. The positive β values on the abscissa denote brightening.*

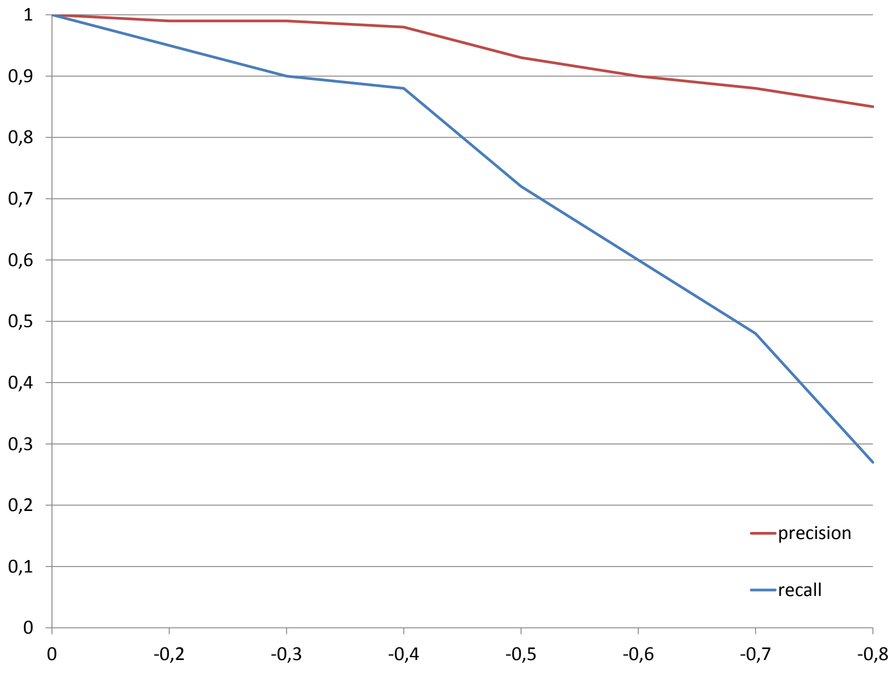


Figure 6.6: *Average precision and recall of the descriptor versus change of illumination for 10 test images. The negative β values on the abscissa denote darkening.*

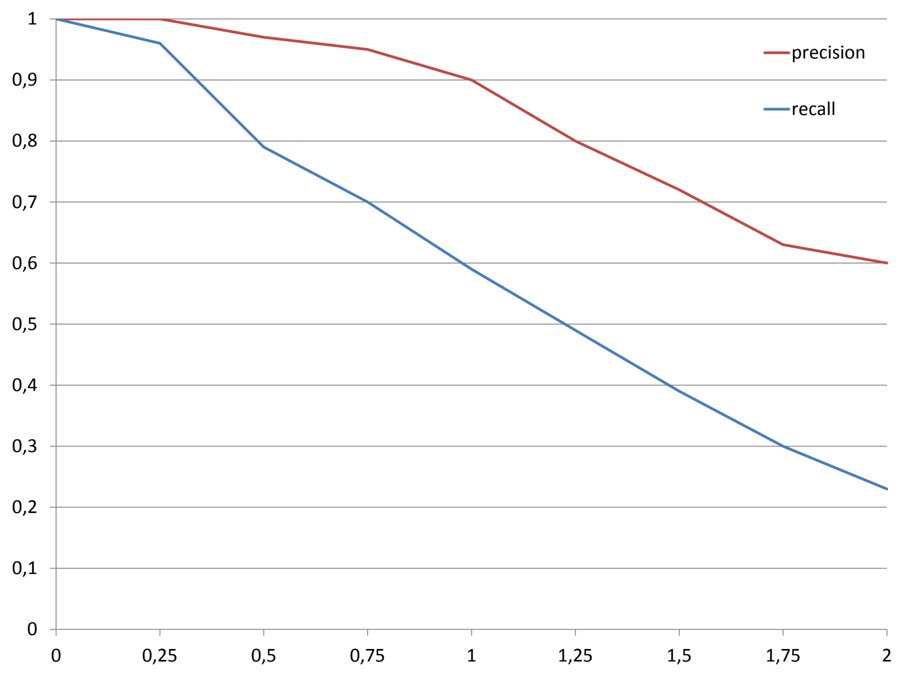


Figure 6.7: *Average precision and recall of the descriptor versus different levels of blurring for 10 test images. The parameter σ is the standard deviation of the Gaussian filter.*

These results help us to extract some interesting and useful informations. First we can notice that SURF is really robuts to changing variations of environment's luminosity, which is really important, as it on autonomous robot and mobile robot specifically, moving in different environments cause changing of lumonosity frequently. Moreover there are also good performance with blurring conditions, which is another important parameter for mobile robots, due to cheap sensors or quick changing of views.

Analyzing the results more interesting for us and this project, or rather the rotations of objects around z and y axes (Figure 6.2), we can observe how feature responds to extreme variations of angles of view. Results show how the performance decrease linearly and in some cases the trend is exponential.
This allow us to define a threshold within which take the model view of the object for a single image, smallest will be the variation, bigger will be the number of samples necessary for a good model, which will make heavy the building of the strucute, and search inside it. A good tradeoff seems to be near 15°, where both rotations return a precision and a recall of 0.87 and 0.67 respectively. In Figures 6.8 and 6.9 are reported the results of the test with this parameter.

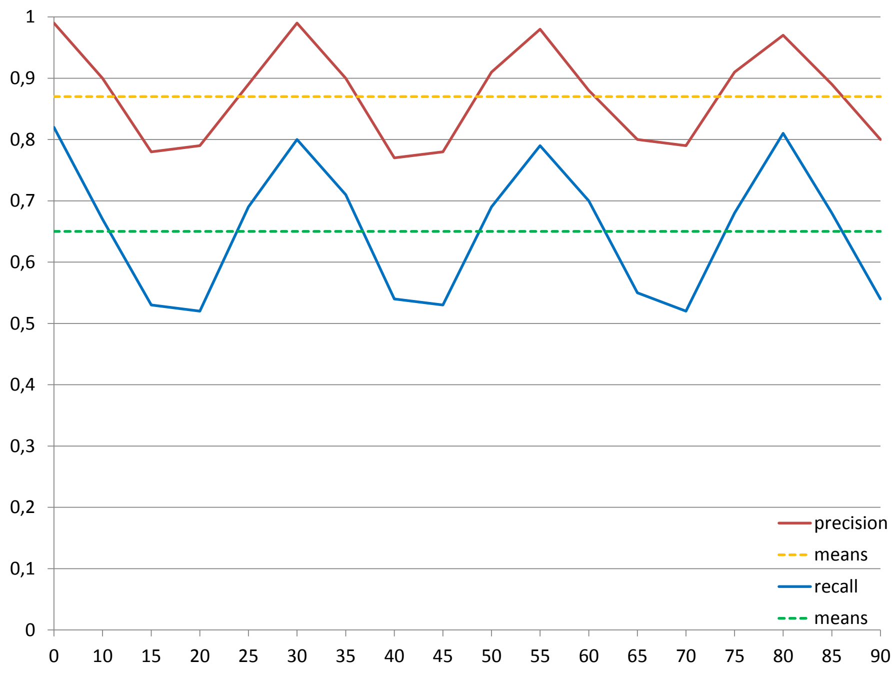


Figure 6.8: *Average precision and recall of the descriptor versus change of yaw angle for a model with variations of 15°.*

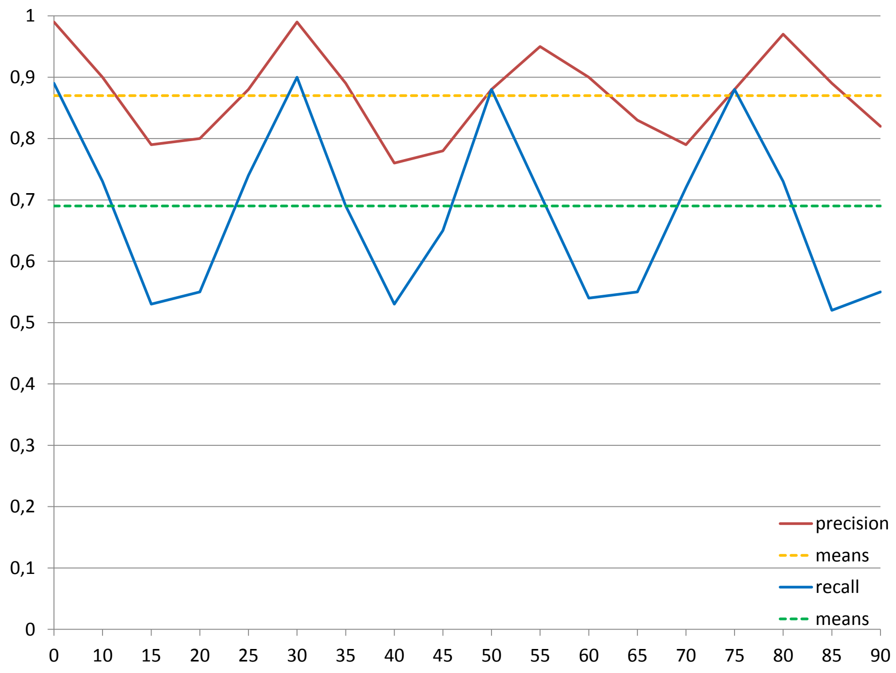


Figure 6.9: *Average precision and recall of the descriptor versus different levels of blurring for 10 test images. The parameter σ is the standard deviation of the Gaussian filter.*

6.4 Extractor and Information performance

Now we show some results about the precision of the data evaluate from the last two modules of the system. Then we will present values returned from the system into the real world.

All the steps have to face with the problem of accuracy of Kinect depth data, which was deeply analyzed on [1] and the results are shortly reported on Figure 6.10. This is confirmed in our tests, where the errors increase quadratically from a few millimeters at $0.5m$ distance to about $4cm$ at the maximum range of $5m$. Same problems are reported for the distance estimation, however this error is still acceptable, as it since the wheelchair works with quite big sized objects, and the final estimation takes account of this error introduced by the Kinect camera with some tradeoff parameters based on the class of the object.

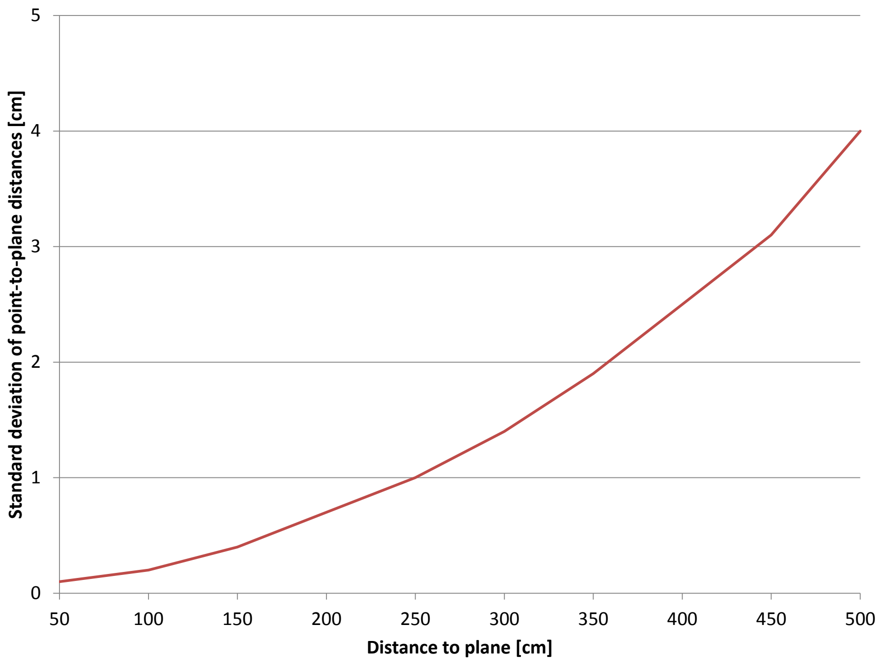


Figure 6.10: *Standard deviation of plane fitting residuals at different distances of the plane to the sensor.*

Geometrical error $e[mm]$	
$\mu(e)$	2.39
$\sigma(e)$	1.67
$max(e)$	8.64

Table 6.2: *Error e corresponds to the Euclidean distance between the sensors and the real one.*

In Figure 6.11 is reported an examples with the results returned of the object recognized by the system.

6.5 Time performances

Since one of our objectives was to operate with the highest frame-rate reachable, we performed some tests to evince the time performance of each step and the average frame rate.

```
CLASS
cartoon box
-----------
SIZE
height:  32.3cm
width:  27.3cm
depth:  18.4cm
-----------
rotation:  41.4°
-----------
distance:  133.7cm
```

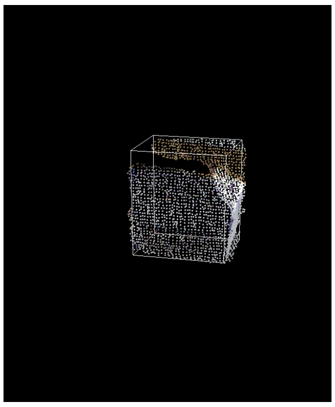


Figure 6.11: *The results of the information evaluated on the selected clusters.*

6.5.1 2D Module

Figure 6.12 shows the performance for the first step of our project, when the system have to categorize the object on the scene (the training step is not included), and as we can see it performs quite fast with an average VGA size of the images. In Figure 6.13 are reported the performances of the detector, which is the most expensive time cost of our system, especially the step of keypoints extraction and descriptors computation that together take almost half of time needed. The matcher shows good results for querying and matching, where the homography also results to be expensive on the total time load. An improvement could be given by employing the GPU version of the keypoints and descriptors which theorically provide performance at least 12$\times$ faster than the normal version [46].

6.5.2 3D Module

For the 3D processing, we can analyzed in Figure 6.14, where the first phase of point cloud extraction from a region of interest, obtained thanks to the 2D step, reduced drastically the computational time for all nexts phases, which can be compute in a less than 35ms. The phase that takes more time, is the segmentation, this is partially due to a sub phase, where the program has to compute the normals of the point cloud. However all this phases are highly dependent by the size of the object to extract. Then last step reported in Figure 6.15 which is composed of only simple estimations, takes only a small part of the total computational time.

An overview of the total computation time of the four main steps are reported in Figure 6.16. They indicate the average rates reached by the Kinect data publication and conversion from point cloud to RGB image, the recognition and detection phase, the extraction and estimation phase, which composed the our complete system (2D and 3D processing). The average time needed for the whole system is 350ms, then reached a frame rate of 3fps as maximum (depends also on the complexity of the scene). This results allow to perform well for a mobile robot in a real-time indoor environments, which does not required to work with high speed, as could be for instance, a car on the street.

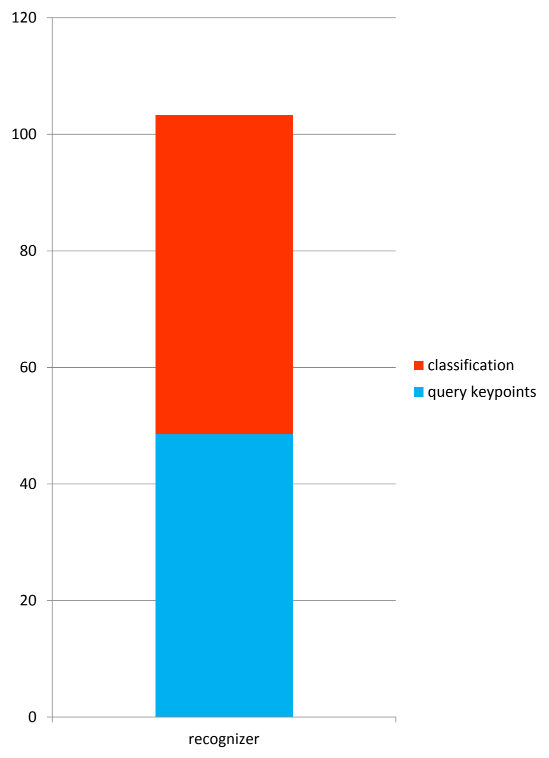


Figure 6.12: *Average time in ms for the main phases of the recognition process.*

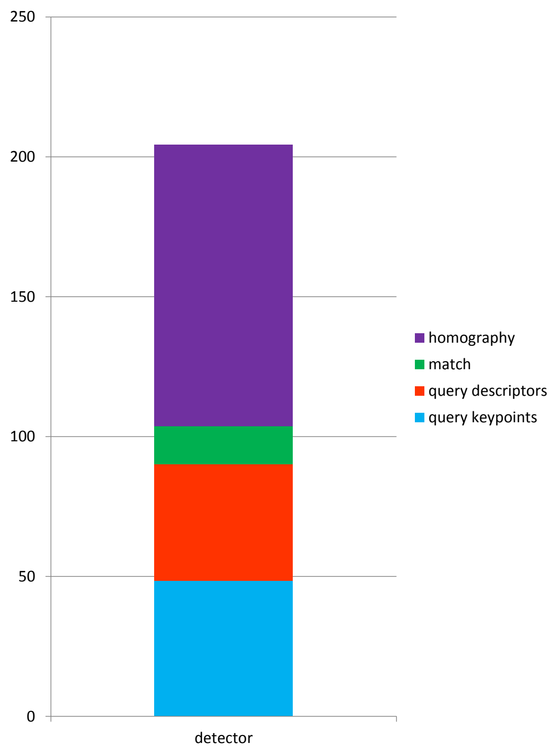


Figure 6.13: *Average time in ms for the main phases of the detection process.*

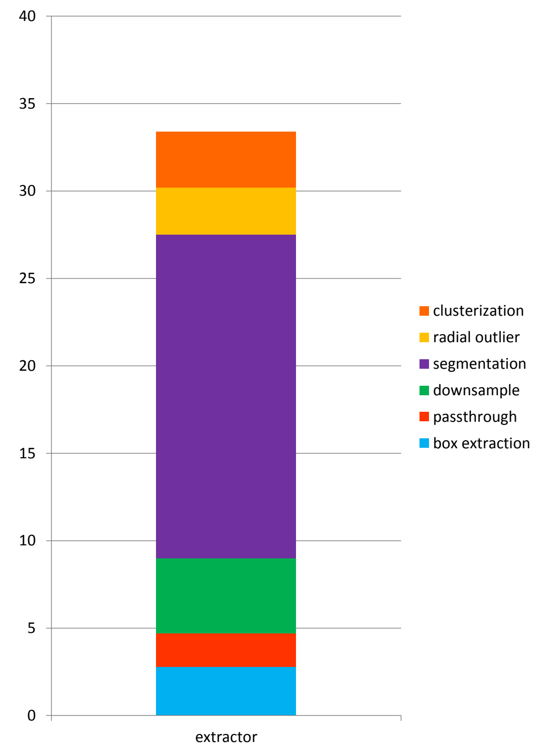


Figure 6.14: *Average time in ms for the main phases of the extraction process.*

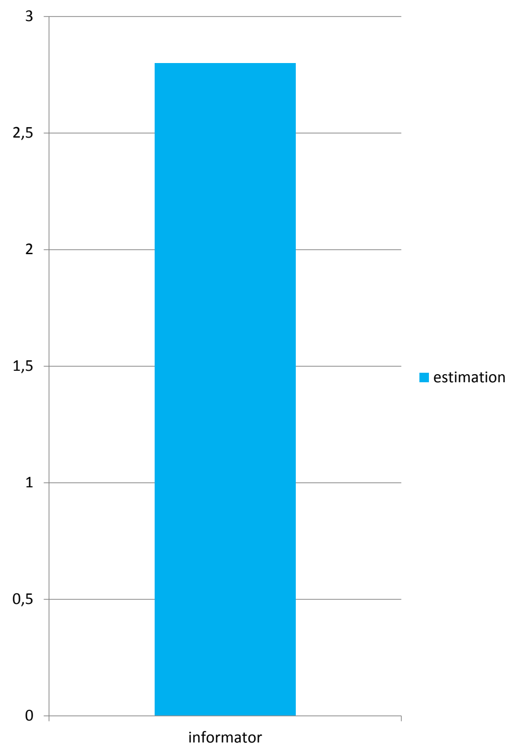


Figure 6.15: *Average time in ms for the main phases of the information estimation process.*

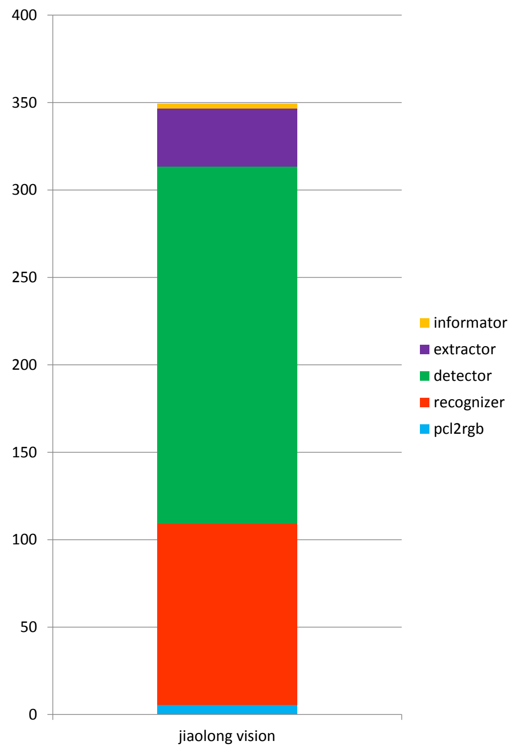


Figure 6.16: *Average in ms time for the main phases of the system.*

6.6 Real-world application

This tests have been performed by AR Labs PhD student Wei Zhi Xuan and presented on the ICIRA 2012 [47], and aimed to show how this software (on the test was used a previous version of the recognition system) improve the trajectory of the wheelchair and choose the right behaviour based on the object recognized.

In Figures 6.17 we can observe three different tasks we want the smart wheelchair performs.

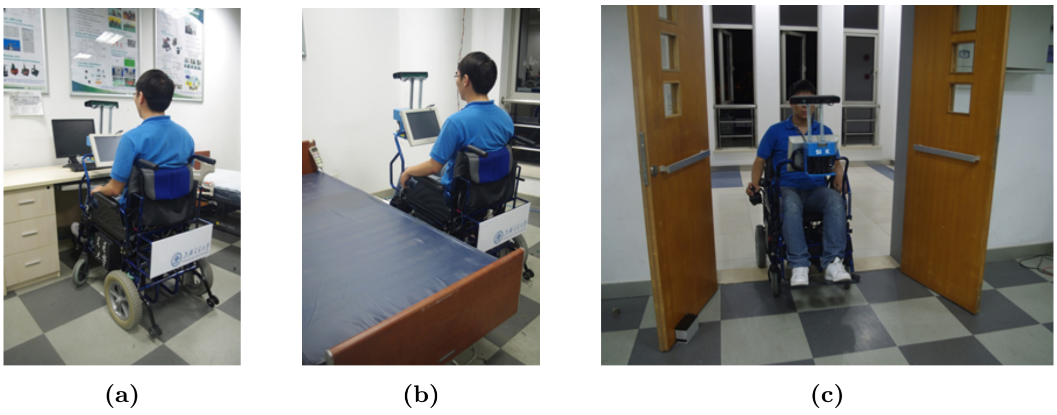


Figure 6.17: *Different tasks for the wheelchair. (a) Docking into the desk. (b) Accosting the bed. (c) Pass through the door.*

In Figure 6.18 is the whole architecture based on the previous shared control (below the dash). Object recognition contain the target information of the environment. At the same time, user intention is estimated to determine whether the user would like to reach the target. If not, the shared control will work as usual; if yes, the system will plan the motion to drive the wheelchair to the target, and the output of motion control will replace the output of joystick, meanwhile the system will adjust the internal parameters of the shared control to adapt to the different situations.

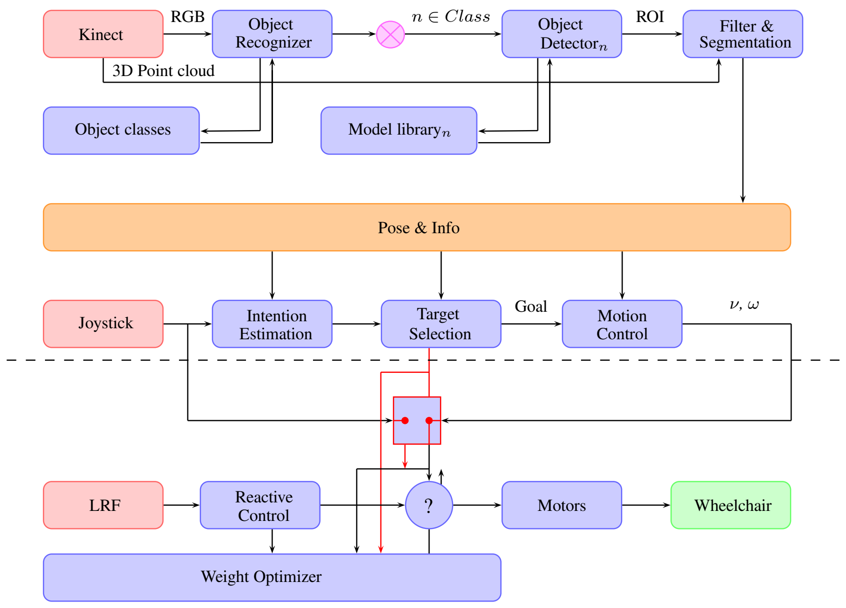


Figure 6.18: *System architecture.*

The experiment is implemented in the laboratory environment as shown in Figure 6.19,where the blue line is the trajectory recorded by the odometry of wheelchair. The tasks of the experiments are driving the wheelchair starting from the passageway, then passing through a doorway to get into the laboratory, and finally docking into the desk.

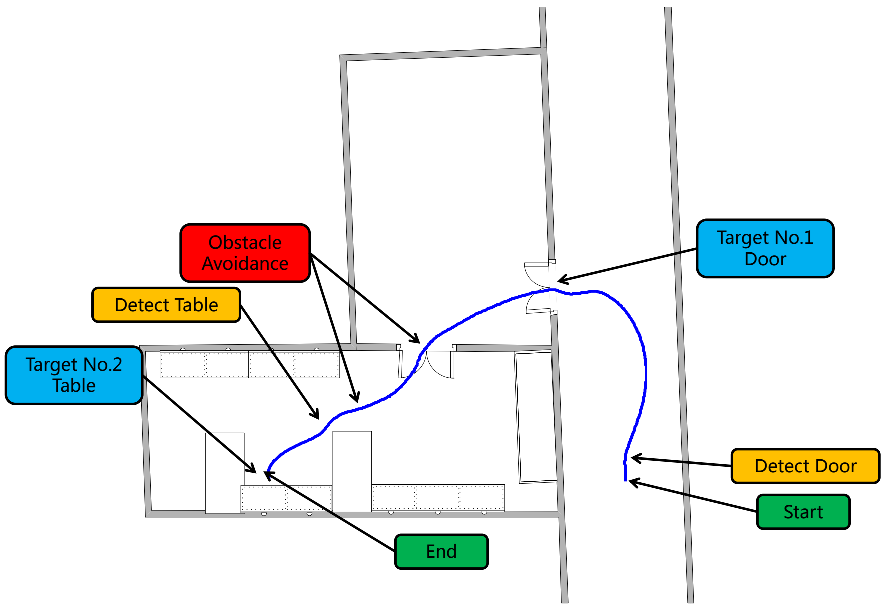


Figure 6.19: *Experimental environment and task.*

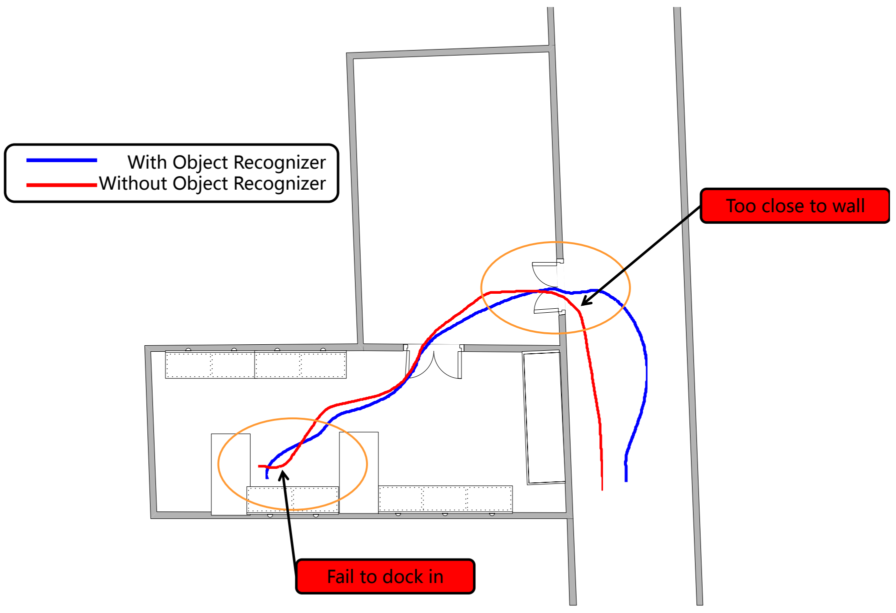


Figure 6.20: *Comparative experiment.*

Figure 6.20 shows the comparison of trajectories with and without the Recognition system activated. As shown, when passing through the doorway, the wheelchair took an arc to align the center of the door and passed through the doorway vertically. The wheelchair controlled without the system, instead, passed very close to one side, which is very dangerous. When docking into the desk, the wheelchair with the Recognition activated docked autonomously and precisely (Figure 6.21), on the opposite, the wheelchair could not approach to the table and failed to dock (Figure 6.22).

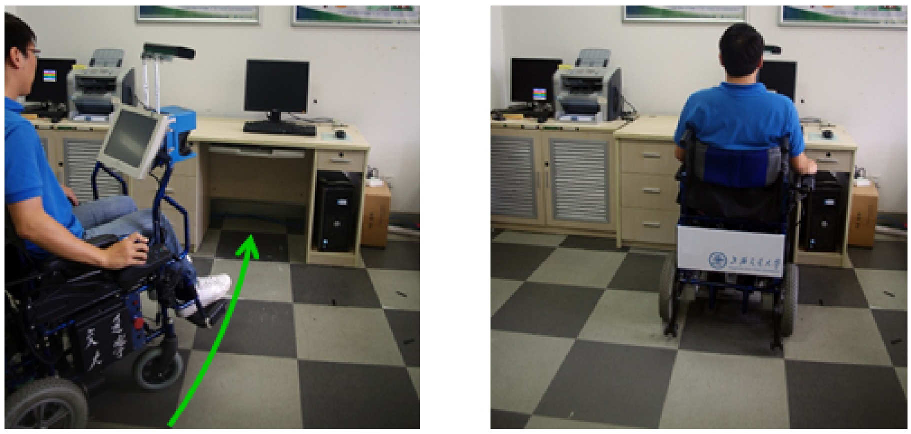


Figure 6.21: *Trajectory with the system of recognition.*

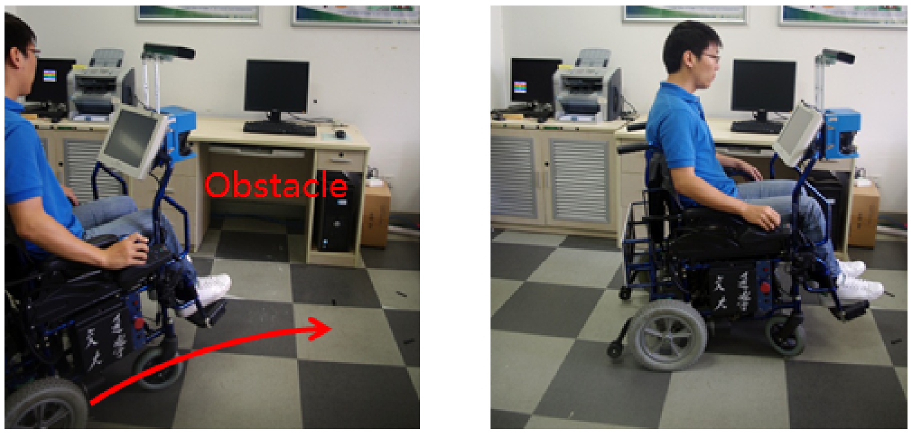


Figure 6.22: *Trajectory without the system of recognition.*

Chapter 7

Conclusions

In this work, we present a hybrid approach for recognition and detection of objects for a mobile robot, in particular for an autonomous wheelchair equipped with a RGB-D camera in an indoor environment. We have proposed a 2D recognition and detection to identify the class of object and its position on the scene, to estimate the pose and size with the 3D data. The results reported in Chapter 6 report that this architecture is able to keep the system computational load low and gives good results in terms of classification and accuracy, though lack of extras checking in extreme situations of occlusion and on the geometric structure of the object stored.

With these considerations, ways of future researches could leads to:

2D Features. Use of new robust and faster features and descriptors, as GPU versions of SURF or the new but not yet tested FREAK, which could increase the matching performance and at the same time reduce the computational load of the system.

Geometric Informations. Study of the geometry correlation of the object, to obtain a more robust and light structure of features for the detection phase.

3D Features. Use of features which work with point cloud, to have a second check on the object after the 2D processing, and compensate weak points of these methods, for instance darkening environment. As first approach we tried almost all the state of arts methods, with poor results both performance and computational time, that lead us to use an hybrid method.

Segmentation. Use of new segmentation methods to build a semantic map of the environment, then correlate these informations with the data obtained from the other phases.

Simulation. Create a 3D model of the autonomous wheelchair by using URDF and Xacro, then test new projects first in a simulation environment like Gazebo, just integrate in ROS.

Bibliography

[1] K. Khoshelham and S.O. Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors (Basel)*, 12(2):1437–54, 2012. 1, 49

[2] David Lowe. Object recognition from local scale-invariant features. pages 1150–1157, 1999. 1

[3] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004. 1, 17, 18, 21, 24

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008. 1, 18

[5] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society. 1, 17

[6] Lei Zhu, Aibing Rao, and Aidong Zhang. Theory of keyblock-based image retrieval. *ACM Trans. Inf. Syst*, 20:224–257, 2002. 1, 17

[7] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, Madison, Wisconsin, June 2003. 1, 18

[8] Federico Tombari and Luigi Di Stefano. Object recognition in 3d scenes with occlusions and clutter by hough voting. In *Proceedings of the 2010 Fourth Pacific-Rim Symposium on Image and Video Technology*, PSIVT '10, pages 349–355, Washington, DC, USA, 2010. IEEE Computer Society. 1

[9] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010. 1

[10] E. Herbst, X. Ren, and D. Fox. Rgb-d object discovery via multi-scene analysis. In *International Conference on Intelligent Robots and Systems (IROS)*, 2011. 1

[11] D. Filliat, E. Battesti, S. Bazeille, G. Duceux, A. Gepperth, L. Harrath, I. Jebari, R. Pereira, A. Tapus, C. Meyer, S. Ieng, R. Benosman, E. Cizeron, J.-C. Mamanna, and B. Pothier. Rgbd object recognition and visual texture classification for indoor semantic mapping. In *Proceedings of the 4th International Conference on Technologies for Practical Robot Applications (TePRA)*, 2012. 1

[12] Qinan Li, Weidong Chen, and Jingchuan Wang. Dynamic shared control for human-wheelchair cooperation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4278 –4283, may 2011. 4

[13] Yong Wang and Weidong Chen. Hybrid map-based navigation for intelligent wheelchair. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 637 –642, may 2011. 4

[14] Sarangi P. Parikh, Valdir Grassi Jr., Vijay Kumar, and Jun Okamoto Jr. Integrating human inputs with autonomous behaviors on an intelligent wheelchair platform. *IEEE Intelligent Systems*, 22(2):33–41, March 2007. 4

[15] Tom Carlson and Yiannis Demiris. Human-wheelchair collaboration through prediction of intention and adaptive assistance. In *2008 IEEE International Conference on Robotics and Automation*, pages 3926–3931. IEEE, 2008. 4

[16] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. 7

[17] Norman Villaroman, Dale Rowe, and Bret Swan. Teaching natural user interaction using openni and the microsoft kinect sensor. In *Proceedings of the 2011 conference on Information technology education*, SIGITE '11, pages 227–232, New York, NY, USA, 2011. ACM. 7

[18] Adrian Bradski. *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. O'Reilly Media, 1. ed. edition, 2008. Gary Bradski and Adrian Kaehler. 8

[19] Robert Laganière. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, May 2011. 8

[20] Nicolas Pinto, David D. Cox, and James J. Dicarlo. Why is real-world visual object recognition hard. *PLoS Computational Biology*. 9

[21] Tinne Tuytelaars and Krystian Mikolajczyk. *Local Invariant Feature Detectors: A Survey*. Now Publishers Inc., Hanover, MA, USA, 2008. 15

[22] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Heidelberg et al., 1998. Springer. 17

[23] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, March 2002. 17

[24] Nello Cristianini, John Shawe-Taylor, and Huma Lodhi. Latent semantic kernels. *J. Intell. Inf. Syst.*, 18(2-3):127–152, March 2002. 17

[25] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, New York, 2000. 19

[26] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998. 20

[27] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998. 21, 24

[28] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. 23

[29] Amalia Duch, Vladimir Estivill-Castro, and Conrado Martinez. Randomized k-dimensional binary search trees. In *Proceedings of the 9th International Symposium on Algorithms and Computation*, ISAAC '98, pages 199–208, London, UK, UK, 1998. Springer-Verlag. 23

[30] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 1000–, Washington, DC, USA, 1997. IEEE Computer Society. 24

[31] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000. 24

[32] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004. 27

[33] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. 27

[34] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004. 27

[35] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009. 27

[36] Zoltan Csaba Marton, Radu Bogdan Rusu, Dominik Jain, Ulrich Klank, and Michael Beetz. Probabilistic Categorization of Kitchen Objects in Table Settings with a Composite Sensor. In *Proceedings of the 22nd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 11-15 2009. 29

[37] Radu Bogdan Rusu. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *Articial Intelligence (KI - Kuenstliche Intelligenz)*, 2010. Invited paper. 29

[38] Radu Bogdan Rusu, Ioan Alexandru Sucan, Brian Gerkey, Sachin Chitta, Michael Beetz, and Lydia E. Kavraki. Real-time Perception-Guided Motion Planning for a Personal Robot. In *Proceedings of the 22nd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 11-15 2009. 29

[39] Radu Bogdan Rusu, Aravind Sundaresan, Benoit Morisset, Kris Hauser, Motilal Agrawal, Jean-Claude Latombe, and Michael Beetz. Leaving Flatland: Efficient Real-Time 3D Perception and Motion Planning. *Journal of Field Robotics (Special Issue on 3D Mapping)*, 2009. 29

[40] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitat Muchen, Germany, October 2009. Advisor: Univ.-Prof. Michael Beetz (TUM) Ph.D.; Committee: Univ.-Prof. Dr. Nassir Navab (TUM), Univ.-Prof. Michael Beetz (TUM) Ph.D., Prof. Kurt Konolige (Stanford) Ph.D., Prof. Gary Bradski (Stanford) Ph.D.; summa cum laude. 29

[41] Radu Bogdan Rusu, Andreas Holzbach, Gary Bradski, and Michael Beetz. Detecting and segmenting objects for mobile manipulation. In *Proceedings of IEEE Workshop on Search in 3D and Video (S3DV), held in conjunction with the 12th IEEE International Conference on Computer Vision (ICCV)*, Kyoto, Japan, September 27 2009. 31

[42] Dirk Holz, Stefan Holzer, and Radu Bogdan Rusu. Real-Time Plane Segmentation using RGB-D Cameras. In *Proceedings of the RoboCup Symposium*, 2011. 31

[43] J. Berkmann and T. Caelli. Computation of surface geometry and segmentation using covariance techniques. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(11):1114–1116, November 1994. 34

[44] Cleve Moler. Cramer's rule on 2-by-2 systems. *SIGNUM Newsl.*, 9(4):13–14, October 1974. 40

[45] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. 41

[46] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Commun. ACM*, 55(6):61–69, June 2012. 50

[47] Zhixuan Wei, Weidong Chen, and Jingchuan Wang. 3d semantic map-based shared control for smart wheelchair. In Chun-Yi Su, Subhash Rakheja, and Honghai Liu, editors, *Intelligent Robotics and Applications*, volume 7507 of *Lecture Notes in Computer Science*, pages 41–51. Springer Berlin / Heidelberg, 2012. 10.1007/978-3-642-33515-0_5. 53

Acknowledgments

Ringraziamenti vari alla famiglia, agli amici, ai parenti, ai compagni di corso ecc ecc Professori, dottorandi, amici di shanghai.