Robot bin picking: 3D pose retrieval based on Point Cloud Library

università degli studi di padova
Facoltà di ingegneria
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Laureando: Stefano Squizzato
Prof. relatore: Emanuele Menegatti
Tutore Aziendale: Stefano Tonello

December 9, 2012

Alla mia famiglia, Eleonora, Sandro e Monica per avermi sostenuto e permesso di giungere fino a questo traguardo.

A Jessica, che mi è sempre vicina, in ogni momento.

Ad Alberto, Alessandro, Luca ed Umberto, per questi cinque impegnativi anni passati a studiare assieme.

Stefano Squizzato

Abstract

This work covers the problem of object recognition and 6 DOF pose estimation in a point cloud data structure, using PCL (Point Cloud Library). The result of the computation will be used for bin picking purposes, but it can also be applied to any context that require to find and align a specific pattern. The goal is to align an object model to all the visible instances of it in an input cloud. The algorithm that will be presented is based on local geometry FPFH descriptors that are computed on a set of uniform keypoints of the point clouds. Correspondences (best match) between such features will be filtered with RANSAC procedure: from this data comes a rough alignment, that will be refined by ICP algorithm. Robust dedicated validation functions will guide the entire process with a greedy approach. Parallelism has also been implemented using OpenMP API. Time and effectiveness will be deeply discussed, since the target industrial application imposes strict constraints of performance and robustness.

The result of the proposed solution is really appreciable, since the algorithm is able to recognize almost all the present objects, with a minimal percentage of false negatives and an almost zero false positives rate. Experiments have been conducted on a large dataset, that was acquired with a triangulation system made up by one camera and two intersecting lasers as structured light sources. Such vision system has been mounted first on a fixed position over a conveyor belt, then on a moving robotic arm, in order to cover a larger area.

Contents

Chapter 1

Introduction

1.1 Overview

Nowadays robotic systems used in industrial production play an increasing important role that will shape the mainstream business technology in the near future. Most systems based on robot manipulators used in manufacturing companies do not include advanced sensors for environment perception. In fact, this kinds of robots are only programmed to play the same sequence of movements continuously, hence they are blind towards the world they are embedded in.

In these conditions, they absolutely need a structured input and a completely known environment, in order to work properly. This requirement is difficult to obtain, especially for limited production batches that are likely to exist in small and medium-sized businesses. The work cell may be reconfigured, the products may vary in shape and size or can be positioned randomly from the previous production processes.

Enabling a robot manipulator to have perception of the environment in which it operates can drastically improve the flexibility of the system, making it versatile to different jobs at will.

In this context, this work aims to design a 3D vision system that is able to solve the problem of the bin-picking, that turns into the visual identification of poses of the objects in the container.

A vision algorithm will be proposed, which is capable of finding the position and orientation of the target within the container : the accuracy has to be enough to permit the robot manipulator to grab the object without causing issues like damages or errors.

This step encloses most of the "intelligence" of the system, since after identification of the pose of an object, the physical pick, collision detection and following industrial processes are treated in the same way as before.

In this work we have been using PCL (Point Cloud Library), an Open-Source framework that deals with 3D data structures, which is getting widely spread among the community of robotic applications since its launch on May 2011.

Combining this powerful platform with prior knowledge of laser triangulation techniques (with correlated 2D image processing) to obtain a dataset, is an excellent choice to approach to this kinds of unexplored problems.

Being successful with these applications means increasing production, saving time and money, supplying Just-In-Time orders, re-qualifying the manpower, providing low-cost solutions for both small and big businesses and being the first mover towards a disruptive technology market that is able to change perspectives even in the near future.

1.2 The bin picking problem

The bin picking problem consists in:

1. Obtain a dataset that represents the bin (like images or 3D scans)
2. Identify all the objects in the bin, that are all their 6 DOF poses
3. Choose the best object in the set, the one with minimum occlusions to avoid collisions
4. Define a picking point for the chosen object, and then evaluate if the pick is feasible
5. Define a path to let the robot manipulator pick the object without damaging anything
6. Manage errors in the picking sequence, like wrong identifications or wrong pick, and try again.

In this thesis steps 2 and 3 are fully covered: the main goals are the robust identification of the objects using point clouds as datasets and evaluating their goodness for the picking process. The other points are not part of this thesis, and they should be handled separately.

1.3 The bin picking working cell

This section contains a brief description of the components of the bin picking system.

- Robot manipulator: the kinematic chain (usually with 6 or 7 joints) that moves on the bin and can be configured with different tools, depending on the desired actions.
- Gripper: the tool mounted on the TCP (Tool Center Point) of the robot manipulator that deals with the grasping of the object.
- Vision system: they are mainly sensors (such as cameras) and lights (structured lights, lasers, illuminators).
- Vision support structure: it can be a fixed structure (for 2D cameras) or a robot on which the system is mounted (for 3D scanning systems); in this last case the robot can be either the same manipulator that takes care of the pick, or an additional Cartesian coordinate robot, also called linear robot.
- Computer to run the heavy computations of the vision algorithms.
- Coordinator: action must be synchronized, and such coordination can be obtained by a PLC or even the same computer that links all the robots and the the vision tools; in a fully automated system there is also the need to define a communication protocol to trigger actions between each component.

Each component works on a Cartesian reference system on its own, but they absolutely need to agree on a common reference system through translations of inputs and/or outputs. Such translations are provided by sophisticated calibration procedures that are performed during the setup of the system.

Chapter 2

Hardware & Software tools

2.1 Overview on 3D Scanners

The actual methods to provide 3D scans are very different and have advantages and disadvantages basing on the application.

In the current state of the art of the strategies that avoid the contact with the object scanned, there are three main categories: the stereo vision, the time-of flight cameras and the structured light.

- Stereo vision: this passive method is low cost but the accuracy depends on features and textures of the scanned object.
- Time-of flight cameras: this active method is used for long range applications (10-100m) and have limited accuracy, low resolution, and a lot of computation complexity.
- Structured light: this active method is based on the projection of a known light pattern on the object to be scanned, such that the way that the pattern deforms when striking surfaces allows vision systems (one or more cameras) to calculate the depth and surface information of the objects in the scene. There are a lot of light patterns that can be used in this application such as:
  - infrared points, used for example in the Microsoft Kinect
  - stripe light patterns, that require a pattern analysis using Fourier transform and Gray code
  - laser planes, that require triangulation calculations.

In the following sections the laser triangulation system will be presented.

2.2 The 2.5D vision system

The scanning system is made up by 3 main hardware components:

- 1 sensor: that is a high resolution camera placed in the middle.
- 2 structured light projectors: that are lasers placed symmetrically respect to the camera.

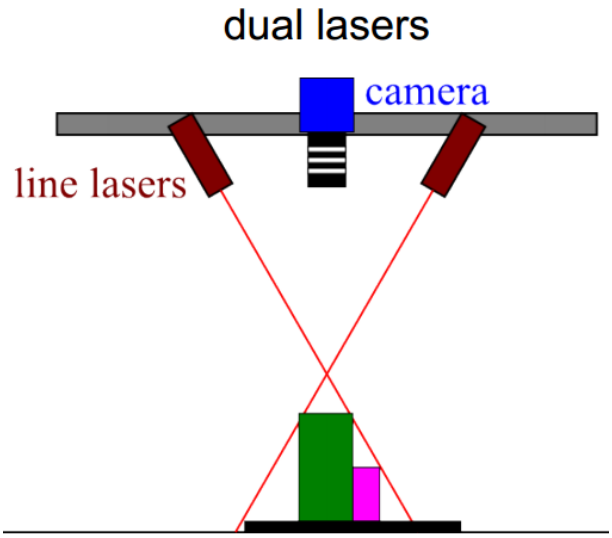

Figure 2.1: Dual laser/Single camera 3D scanning system

The lasers produce a 5mW power, 640 to 670 nm wavelength red colored planes that hit the scene projecting two segmented lines, depending on the actual objects.

The triangulation technique works by calculating the height basing on the deviation of the laser light from its natural course. The procedure takes the data of the camera (captures presenting all the lines of the planes) and calculates the height of each pixel in the line, hence providing at each frame a line of height values.

More precisely, the camera has the following characteristics: 4:3 ratio, 28 fps, 1600x1200 resolution.

2.2.1 Conveyor belt

In this layout the scanning system is fixed on a mechanical support over the moving conveyor belt.

The speed of the conveyor belt used for this work was set in the range 10 to 30 mm/s, hence the quality of the resulting point cloud is high and focused on a small area.

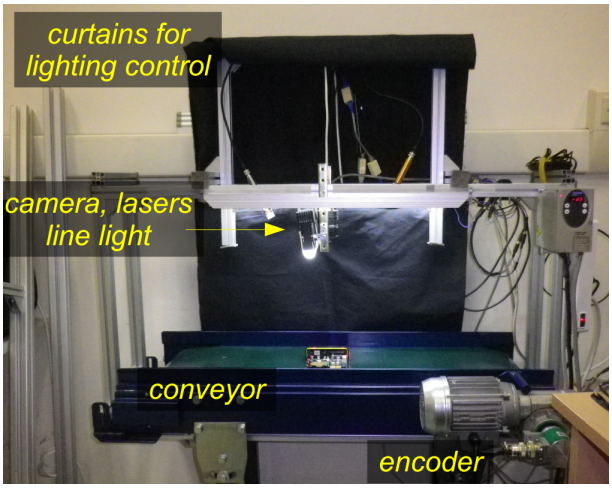

Figure 2.2: The conveyor belt system: lasers and camera are occluded by the aluminum structure

2.2.2 Robotic Arm

In this layout the scanning system is mounted on the moving robotic arm over the fixed area to be scanned.

The speed of the scanning that has been used for this work was set in the range 30 to 60 mm/s, hence the quality of the resulting point cloud is lower, while enlarging the scanned area.

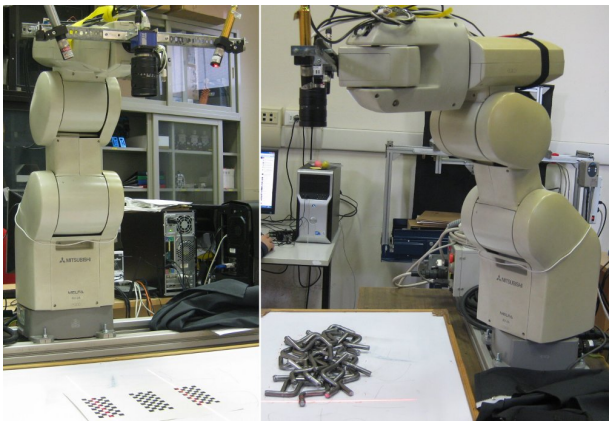

Figure 2.3: The robotic arm, while calibrating and scanning

2.3 The Point Cloud Library

The Point Cloud Library (or PCL) is a large scale, open project for 2D/3D image and point cloud processing. 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.

PCL is released under the terms of the BSD license and is open source software. It is free for commercial and research use.

PCL is cross-platform, and has been successfully compiled and deployed on Linux, MacOS, Windows, and Android/iOS. To simplify development, PCL is split into a series of smaller code libraries, that can be compiled separately. This modularity is important for distributing PCL on platforms with reduced computational or size constraints.

2.3.1 Details

In version 1.6 of the framework there are several modules:

- **common**: contains the common data structures and methods used by the majority of PCL libraries. The core data structures include the PointCloud class and a multitude of point types that are used to represent points, surface normals, RGB color values and feature descriptors. It also contains numerous functions for computing distances/norms, means and covariances, angular conversions and geometric transformations.
- **features**: contains data structures and mechanisms for 3D feature estimation from point cloud data. 3D features are representations at a certain 3D point or position in space, which describe geometrical patterns based on the information available around the point. The data space selected around the query point is usually referred as the k-neighborhood.
- **filters**: contains outlier and noise removal mechanisms for 3D point cloud data filtering applications. It also contains generic filters used to extract subsets of point cloud, or to exclude parts of it. It provides a voxel-grid class to down-sample a point cloud by intersecting it with a lattice of points.

- **geometry**: reserved for future work, it will contain computational geometry data structures and algorithms.
- **io**: contains classes and functions for reading and writing point cloud data (PCD and PLY) files, as well as capturing point clouds from a variety of (OpenNI compatible) sensing devices
- **kdtree**: provides the kd-tree data-structure, using FLANN implementation, that allows for fast nearest neighbor searches. A Kd-tree (k-dimensional tree, in most cases in this work it is a 3d-tree) is a space-partitioning data structure that stores a set of k-dimensional points in a tree structure that enables efficient range searches and nearest neighbor searches. Nearest neighbor searches are a core operation when working with point cloud data and can be used to find correspondences between groups of points or feature descriptors or to define the local neighborhood around a point or points.
- **keypoints**: contains implementations of several point cloud keypoint detection algorithms. Keypoints (also referred to as interest points) are points in an image or point cloud that are stable, distinctive, and can be identified using a well-defined detection criteria. Typically, the number of interest points in a point cloud will be much smaller than the total number of points in the cloud, and when used in combination with local feature descriptors at each keypoint, the keypoints and descriptors can be used to form a compact but distinctive representation of the original data. Harris, Narf, Sift and Uniform keypoints are implemented in the current version.
- **octree**: provides efficient methods for creating a hierarchical tree data structure from point cloud data. This enables spatial partitioning, downsampling and search operations on the point data set. Each octree node the has either eight children or no children. The root node describes a cubic bounding box which encapsulates all points. At every tree level, this space becomes subdivided by a factor of 2 which results in an increased voxel resolution.
- **registration**: contains many point cloud registration algorithms for both organized an unorganized (general purpose) datasets, including ICP, correspondence finding and rejection, transformation estimators.
- **sample_consensus**: holds SAmple Consensus (SAC) methods like RANSAC and models like planes and cylinders. These can combined freely in order to detect specific models and their paramters in point clouds. Some of the models implemented in this library include: lines, planes, cylinders, and spheres. Plane fitting is often applied to the task of detecting common

indoor surfaces, such as walls, floors, and table tops. Other models can be used to detect and segment objects with common geometric structures.

- **search**: provides methods for searching for nearest neighbors using different data structures, including kd-trees, octrees, brute force and specialized search for organized datasets.
- **segmentation**: contains algorithms for segmenting a point cloud into distinct clusters. These algorithms are best suited to processing a point cloud that is composed of a number of spatially isolated regions. In such cases, clustering is often used to break the cloud down into its constituent parts, which can then be processed independently. It also contains algorithms to find differences between two point cloud, that can be used for example for quality inspection purposes.
- **surface**: deals with reconstructing the original surfaces from 3D scans. Depending on the task at hand, this can be for example the convex/concave hull, a mesh representation or a smoothed/resampled surface with normals. Creating a convex or concave hull is useful for example when there is a need for a simplified surface representation or when boundaries need to be extracted. Meshing is a general way to create a surface out of points which algorithms are based on marching cubes. Smoothing and resampling can be important if the cloud is noisy, or if it is composed of multiple scans that are not aligned perfectly. The complexity of the surface estimation can be adjusted, and normals can be estimated in the same step if needed.
- **visualization**: this library was built for the purpose of being able to quickly prototype and visualize the results of algorithms operating on 3D point cloud data. Similar to OpenCV's highgui routines for displaying 2D images and for drawing basic 2D shapes on screen. The package makes use of the VTK library for 3D rendering for range image and 2D operations. Point clouds, normals, range images, correspondences can be added to the viewer window.

2.3.2 Development

The project is being developed by a large number of engineers and scientists from many different organizations, geographically distributed all around the world.

The project is financially supported by Open Perception, Willow Garage, NVidia, Google, Toyota, Trimble, Urban Robotics, Honda Research Institute, Sandia Intelligent Systems and Robotics, Dinast, Optronic, Velodyne, CogniMem Technologies, Fotonic, and Ocular Robotics.

Version 1.0.0 has been released on May 2011 and until today the project has reached a good level of maturity, since the library is progressively more and more used in academic and industrial application.

Chapter 3

Registration

The registration is a very common technique in literature used to combine several datasets into a global consistent model.

The key idea is to identify corresponding points between the data sets and find a transformation that minimizes the distance (alignment error) between corresponding points. This process is repeated until the alignment errors fall below a given threshold: at this point the registration is said to be complete.

A motivation example is given by figure below, where a set of six individual datasets has been acquired using a tilting 2D laser unit. Each individual scan represents only a small part of the surrounding world, so the registration is required to obtain a merged point cloud model.

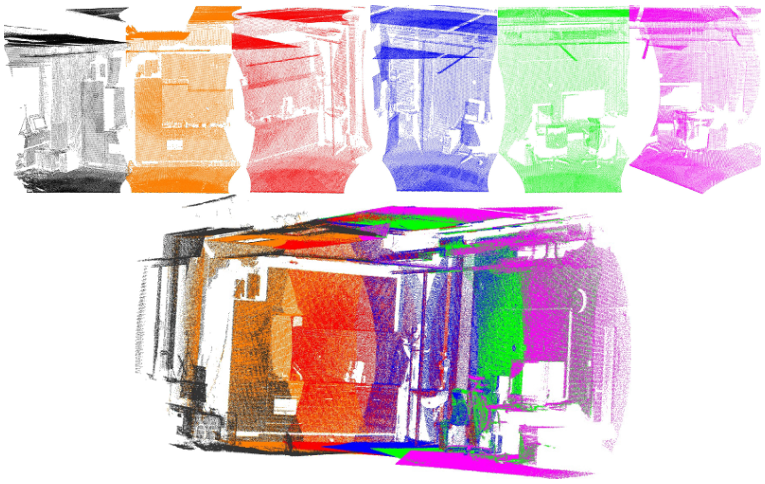

Figure 3.1: Example of the classical registration problem

Using techniques and algorithms provided by literature, it is also possible to

use the registration to align similar objects themselves, or a single object to a big scene. In this case we speak about *object recognition problem*, but it is very similar to the registration problem.

The Point Cloud Library, at current release 1.6, does not support object recognition algorithms, so an original solution had to be found. The OpenSource community however is developing new modules that deal with recognition of objects in the scene basing on correspondence grouping.

The figure below shows the results of the PCL developers trying to match the milk cartoon in the scene.

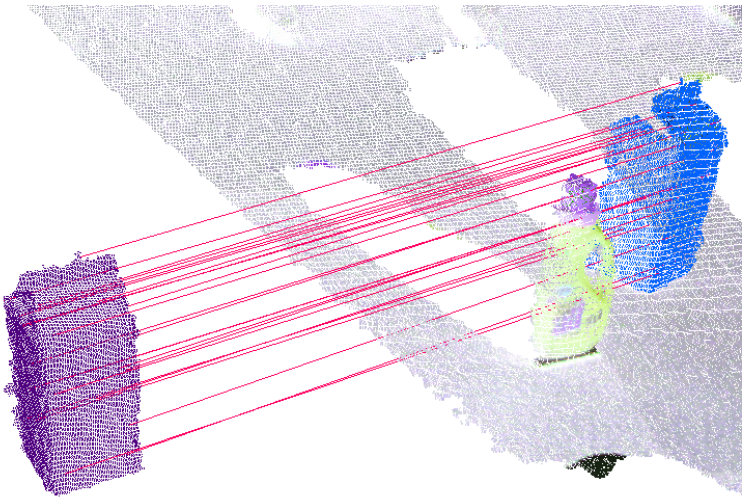

Figure 3.2: Results of the experimental "Object recognition module", available in trunk PCL library not-yet-released.

3.0.3 Definition of the problem

Our goal is to find the coordinates of the picking point of the best objects to peek that are placed in random positions inside the scene scanned by the laser triangulation system. Since the picking point may vary depending on the object, we will simply find the roto-translation matrices that move the object from a known position to in instance of it the world cloud: the alignment with the object found should be nearly perfect

3.0.3.1 Input

To achieve the goal we first have to define the input:

- The *object*: the point cloud of the part of the object to find that must be visible in order to recognize its pose. That is, the model of our object, the

pattern that we have to match. This object is pre-scanned with the same system and edited manually to remove noise and useless parts.

- The *world*: the noisy point cloud of the entire scene acquired with the scanning system. This scene contains the surface of the bin where all the same objects are placed in a random pose.

3.0.3.2 Output

The output of the algorithm is a set M of roto-translation matrices M_i used to align the object point cloud to the most part of the visible objects in the world point cloud (n objects).

$$M = \{M_1, M_2, \cdots M_n\}$$

This kind of rigid transformation is a 4x4 matrix that contains a 3x3 rotation matrix R and a translation 3D vector t

$$M_i = \begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

With each matrix there is a validation score $v_i \in \mathbb{R}$ that measures the goodness of the alignment, as described in the following sections.

$$V = \{v_1, v_2, \cdots v_n\}$$

Hence, each alignment provided by this algorithm is described by the couple $\{M_i, v_i\}$: the matrix and its validation score.

The algorithm also outputs a visualizer provided by the PCL library, where the results of the application of the transformations are applied. The user can then visually estimate the goodness of the results.

3.0.4 Workflow of the general registration process

The computational steps for two datasets are straightforward:

1. from a set of points, identify interest points (i.e., keypoints) that best represent the scene in both datasets;

2. at each keypoint, compute a feature descriptor;
3. from the set of feature descriptors together with their XYZ positions in the two datasets, estimate a set of correspondences, based on the similarities between features and positions;
4. data is assumed to be noisy and not all correspondences are valid, so reject those bad correspondences that contribute negatively to the registration process;
5. from the remaining set of good correspondences, estimate a rough transformation.
6. the steps 3-4-5 are iterated faster in the ICP algorithm to achieve a perfect alignment

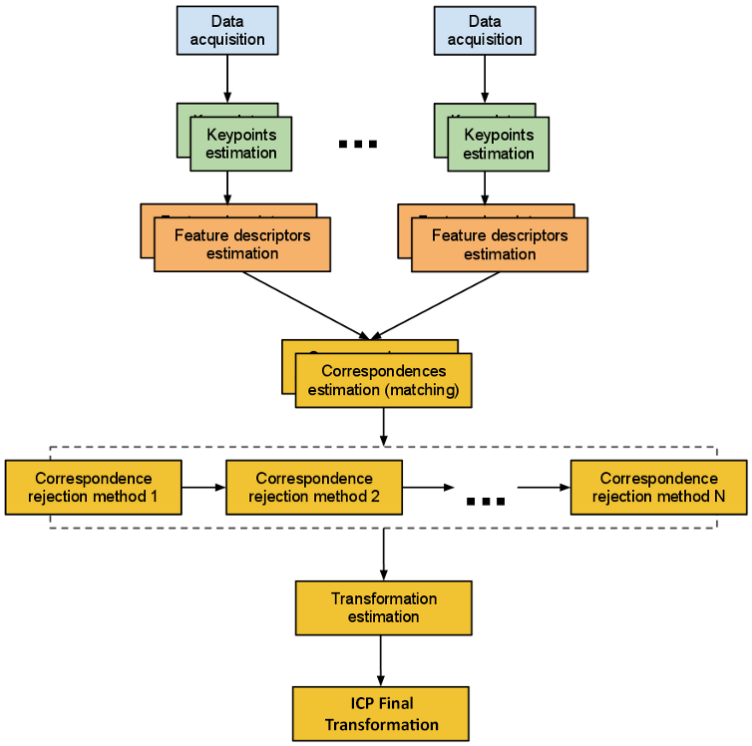

Figure 3.3: Workflow of a general registration problem

3.1 Validation Functions

In order to verify the correctness of any registration we can perform, we need a robust validation function, that is capable of telling a good alignment to a bad

one.

In the PCL library there is a class called TransformationValidationEuclidean, whose purpose is to calculate a double score parameter using two generic datasets and a maxRange parameter: the pair of points whose distance is less than maxRange are taken into account. This method is too generic, in our case the application outputs a lot of false positives.

For our needs three different validation functions have been developed:

1. Euclidean distance validation function: it is used in the initial registration to find out the best match using the square distances between points.
2. Percent of outliers validation function: it is used in the final registration to recognize occlusions using the number of outliers points.
3. Normal angles validation function: it has been tried together with the Euclidean distance validation function, but results are much worse, spending much more computation time.

The detailed description of the functions are given below.

3.1.1 Euclidean Distance validation function

It is used while performing the initial rough registration.

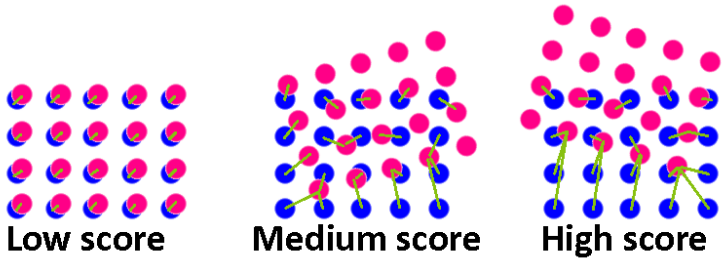

Figure 3.4: Blue = Object cloud. Red World cloud.
Three examples of the calculation of the Euclidean Score: the mean value of the green segments

3.1.1.1 Description of the function

A new specific-purpose fitness score that uses squared distances has been made as follows.

- [input] the world cloud consisting of n points

- [input] the object model cloud consisting of m points.
- [output] a single float *score* that represents the grade of alignment of the object into the world:
  - 0 means perfect alignment.
  - FLT_MAX ($3.40282 \cdot 10^{38}$) means worst alignment

$$score = \frac{1}{m} \sum_{i=0}^{m-1} min_{j=0}^{n-1} \parallel object.i - world.j \parallel^2$$

where object.i and world.j represent respectively the 3D vectors containing the coordinates of the points.

Algorithm 3.1 Pseudo code for the Euclidean distance validation function

1. for each point object.i find the nearest point world.j
2. calculate the squared distance
3. calculate the mean value of all squared distances

3.1.1.2 Results

This formula appears to be a very good method to roughly validate the achieved results, because it is based on squared distances (that penalize even the smaller distance object-world).

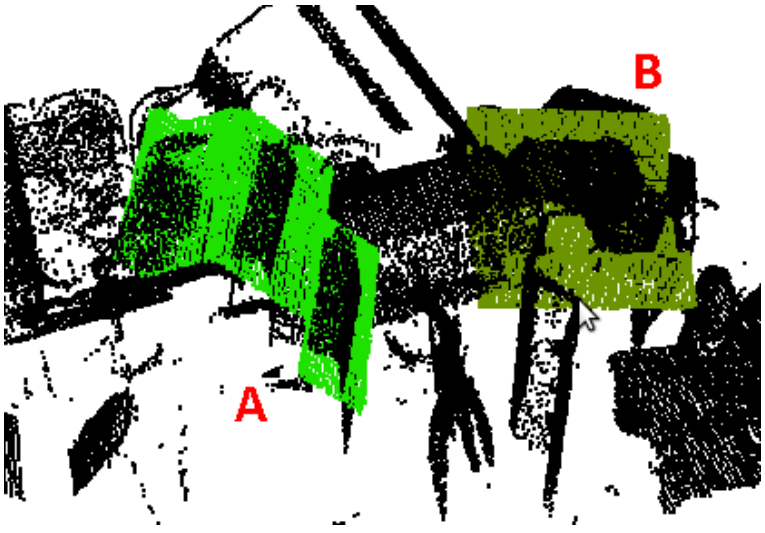

Figure 3.5: Examples of application of Euclidean distance Validation Function
A: wrong registration: score=11.8
B: completely wrong registration: score=42.5

The drawbacks of this approach are:

- the possible presence of occlusions: in this case the score grows very fast, compromising the entire effectiveness of the registration process. The key to avoid this is based on choosing the right object cloud.
- a wrong alignment can maintain small distances between object and world: in this case the score remains low and we cannot tell if the alignment is correct.

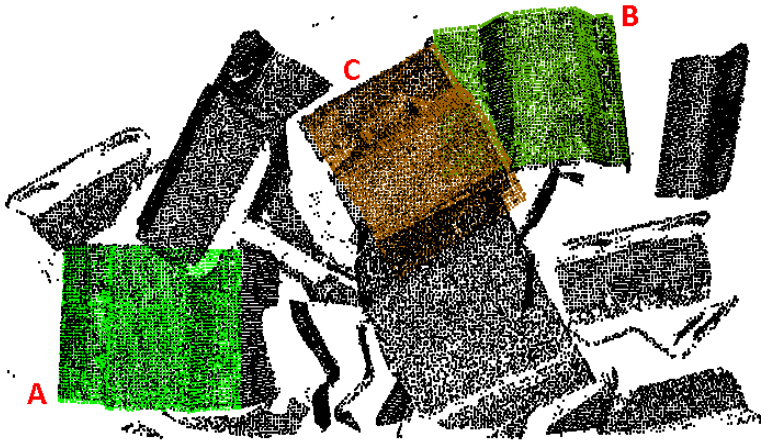

Figure 3.6: The case of occlusions: the score grows fast
A: nearly absence of occlusion: score=1.04
B: some occlusion: score=6.38
C: hard occlusion: score=12.63

3.1.1.3 Performance: $O(n \cdot logn)$

We take n as the size of the input (number of keypoints of the world cloud) because m is $O(n)$.

The search of the nearest point is done by the fast KD-Tree-FLANN implementation available in PCL library.

The function has to be fast, so it is not performed on the original clouds, but on a down-sampled version of them. Note that because of the $\frac{1}{m}$ in the formula, the score is independent from the size of the input, but the more points means the more accuracy.

- Construction of the KD-Tree: $O(n \cdot logn)$
- Find nearest for each object point: $O(m \cdot logn)$

- Total: $O(n \cdot logn)$ because m is $O(n)$

Note that n is not the size of the world cloud, but the size of the down-sampled world cloud (with uniform keypoints), that is much smaller.

3.1.2 Percent of outliers validation function

It is used while performing the final fine registration.

3.1.2.1 Description of the function

A simpler but more effective fitness score to recognize occlusions and bad matching has been developed.

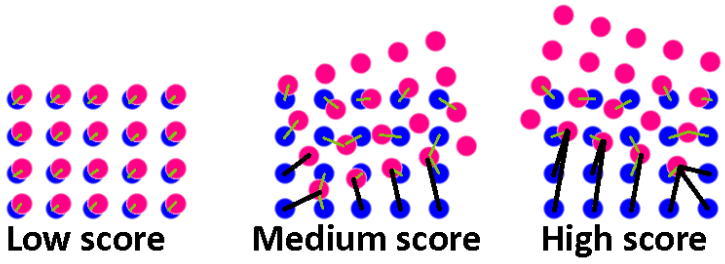

Figure 3.7: Blue = Object cloud. Red = World cloud.
Three examples of the calculation of the Euclidean Score: the percent of black (furthest) segments on all blue points.

- [input] the world cloud consisting of n points (so n is the size of the input)
- [input] the object model cloud consisting of m points.
- [output] a single float *score* that represents the percent of outliers basing on a small distance threshold.
  - 0 means perfect alignment: 0% of outliers
  - 100 means worst alignment: 100% of outliers

$$score = \frac{100}{m} \sum_{i=0}^{m-1} \begin{cases} 1 & if\ min_{j=0}^{n-1} \parallel object.i - world.j \parallel > distanceThreshold \\ 0 & otherwise \end{cases}$$

where object.i and world.j represent respectively the 3D vectors containing the coordinates of the points. In other words: no matter if the distance is high, each outliers counts as 1.

Algorithm 3.2 Pseudo code for the Percent of outliers validation function

1. for each point object.i find the nearest point world.j
2. calculate the distance: if it is greater than the distanceThreshold it is an outlier
3. calculate the percent of outliers by dividing the sum by m and multiplying for 100

3.1.2.2 Results

This function is required because the previous one is not sensible to particular cases like the one shown in figure, where euclidean distances are small but the alignment is wrong. As shown, this function is able to tell a wrong alignment from a good one, meanwhile the euclidean distance function cannot discriminate a false positive.

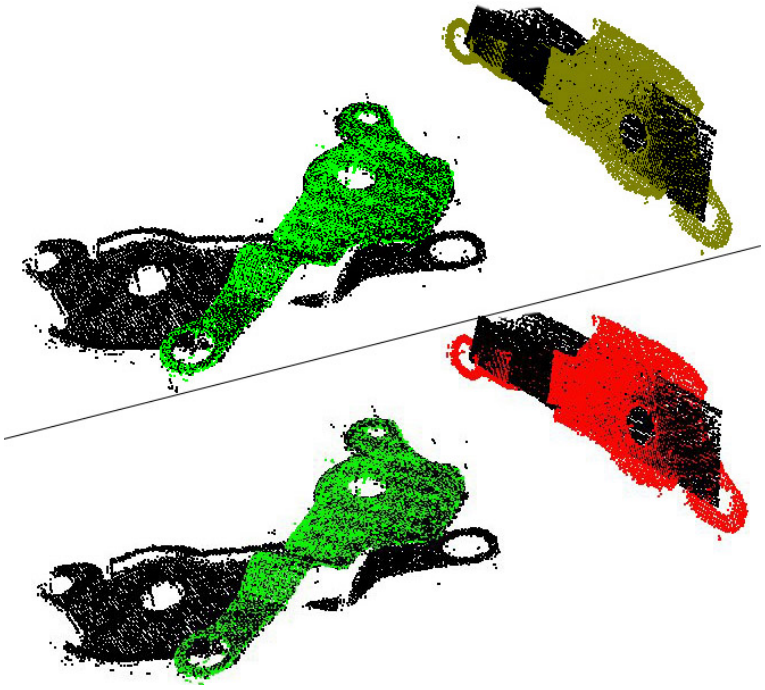

Figure 3.8: Comparison between Validation Functions.
(Top) Euclidean distance validation function.
Really good score on the left: 1.8 - Quite good score on the right: 7.4
(Bottom) Percent of outliers validation function.
Really good score on the left: 2.4 - Really bad score on the right : 37.2

In other words, the euclidean distance validation function is not capable of

telling an occlusion to a wrong alignment: the score is nearly the same.

For our kinds of objects the threshold for this functions has been set at 4.0 mm.

3.1.2.3 Performance $O(n \cdot logn)$

The cloud is down-sampled for the computation in this case too. The performance is the same as the Euclidean distance validation function.

3.1.3 Normal Angles Validation Function

This function has been tried to take the place of the Percent of outliers validation function in order to score the final registration, but the low performance of this procedure made it a bad choice if a good performance is required.

3.1.3.1 Description of the function

This function is very similar to the euclidean distance validation function, but it is based on the squared differences of angles between the normals of the object cloud and the normals of the world cloud. More precisely, the normals taken into account are calculated on each point in the object cloud and its closest neighbor.

- [input] the world cloud consisting of n points (so n is the size of the input)
- [input] the object model cloud consisting of m points.
- [output] a single float *score* that represents the grade of alignment of the object into the world:
  - 0 means perfect alignment: 0° of difference between normals for all the points
  - 180^2 means worst alignment: 180° means that all the normals are opposite

Algorithm 3.3 Pseudo code for the Normal Angles validation function

1. compute normals for each point of object and world cloud with a fixed given radius (5mm)
2. for each point object.i
   (a) find the nearest point world.j
   (b) calculate the squared distance of the angles between normal.object.i and normal.world.j
3. calculate the mean value of all squared distances

3.1.3.2 Results

Tests are shown in the following section, in comparison with the other two. This function is not good for our case, both for quality of the results and time consumption.

3.1.3.3 Performance $O(n \cdot k + n \cdot logn)$

This algorithm is slower than the Euclidean Distance algorithm described above, because the first step is the most complex.

For each point of the input clouds it has to find the normals using the neighboring points within a fixed given radius of 5mm. Hence, said k the average number of neighbors within this radius, the complexity of this first step is $O(2n \cdot k)$.

The other steps involve only the building of the KD-Tree ($n \cdot log(n)$) and the n-times search on it ($log(n)$), so the complexity remains $O(n \cdot logn)$.

3.1.4 Comparison between validation functions

Results are presented with an image, scores and times of the three functions compared.

The time in milliseconds presented in the following tables have been obtained using a AMD Quadcore processor @ 3.0 Ghz with 2 Gb RAM

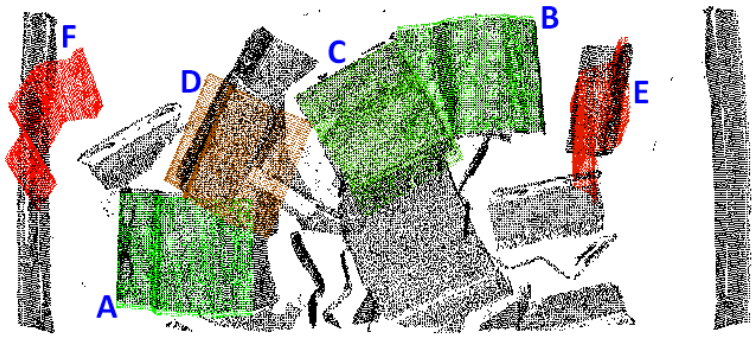

Figure 3.9: Comparison between different validation scores. See the following table for the data.

Capital letters refer to the above image: A, B, C are the correct attempts; D, E, F are wrong.

Scores (applied on downsampled clouds)	A	B	C	D	E	F
Euclidean Distance	1.0	5.7	13	22	83	70
Percent of Outliers	4.4	12	*18*	*50*	66	71
Normal Angles	112	1100	1200	1100	2500	11000

Note that the function Angles does not recognize any difference between B - C that are correct, and D that is completely wrong.

Time [ms] (applied on downsampled clouds)	A	B	C	D	E	F
Euclidean Distance	6	7	6	6	7	7
Percent of Outliers	6	9	6	7	8	8
Normal Angles	53	46	50	50	50	51

The above results are computed on down-sampled clouds during the final registration procedure.

Note that there is one order of magnitude of time difference between Normal Angles validation function and the other two.

For completeness now the tables with the same functions applied to the original (not down-sampled) clouds are shown.

Scores (applied on original clouds)	A	B	C	D	E	F
Euclidean Distance	0.5	5.4	13	21	82	69
Percent of Outliers	2.5	10	*18*	*46*	62	67
Normal Angles	64	820	1100	1300	2400	11000

The differences of angle scores between correct and incorrect examples are still not noticeable.

Time [ms] (applied on original clouds)	A	B	C	D	E	F
Euclidean Distance	22	25	27	27	36	38
Percent of Outliers	23	28	27	28	38	39
Normal Angles	339	329	301	321	321	324

Even the results performed on the original clouds present an order of magnitude of difference between Validation Function Angles and the other two functions.

Note that in every case the Percent Of Outliers functions shows a lot of difference between C (that is the worst of the correct attempts) and D (that is the best of wrong attempts).

3.2 Keypoints

Keypoints, generally speaking, are special points found in a point cloud, that are likely to be found in another similar point cloud. They are also known as "points of interest": in fact they should be able to describe effectively the entire cloud using much less data.

They are usually placed at corners of shapes and where the color/brightness gradient is changing fast. There are various methods to find keypoints, and each technique has its specific output.

3.2.1 NARF Keypoints

Narf keypoints are computed using a range image provided by the input stream. In this case the input stream is the point cloud itself, so there is no availability of a range image. However it is possible to obtain a range image from it, providing the 3 coordinates of the view and the 3 angles of the rotation around the axis.

Even if the range image was reconstructed, the results were extremely slow and not globally relevant because NARF keypoints are very sensible to the viewpoint (i.e. position from which the scene is viewed).

3.2.2 Harris Keypoints

The Harris method is mainly used to find corners in a 2D image. PCL community has ported the same algorithm to work with 3D to 6D point clouds. In this work I have tested the effectiveness of the 3D HarrisKeypoints with all of

the 5 methods implemented in the PCL library: HARRIS, LOWE, TOMASI, CURVATURE, NOBLE.

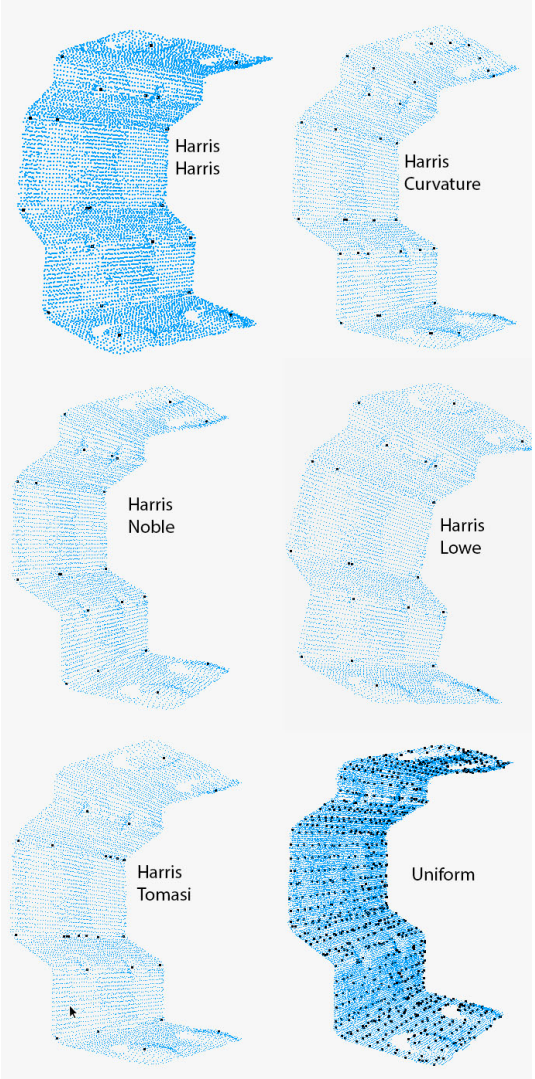

Figure 3.10: Comparison between the 5 variants of Harris Keypoints and the Uniform Keypoints (bottom right)

Harris keypoint detection is extremely slow, deeply depending on the input parameters. It however provides points at the corners that are important in other applications, but are less useful in this one.In fact, since the alignment is based on correspondence between descriptors and not between points, it is better to have more keypoints to compute better features.

3.2.3 Uniform Sampling

The uniform sampling for keypoints works somewhat like the voxel filter, that is:

- a grid of cubes (with edge = keypoint leafSize) is built.
- for each cube the centroid of the present points is computed and it is set as the voxel for that cube.

For the uniform keypoints another step is added to the algorithm:

- the point of the source cloud nearest to the computed voxel is set as the keypoint for that cube.

Hence, the difference between voxels and uniform keypoints is: the uniform keypoint belongs to the source cloud, the voxel does not; that is, keypoints are a subset of the input cloud, voxels are not.

The uniform keypoint has been chosen because voxels introduce new points and then more noise affecting results.

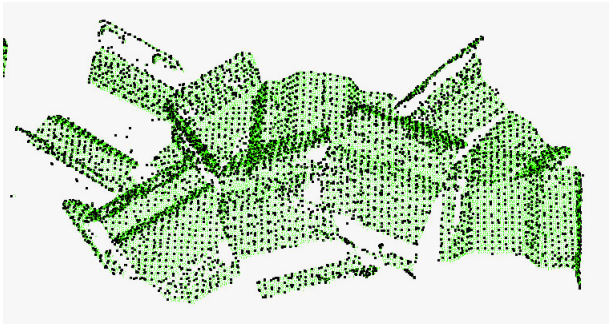

Figure 3.11: Uniform keypoints found on the world cloud

3.3 Features

Features (or descriptors), generally speaking, are structures containing data that describes the dataset.

They are very important to computer vision because most algorithms are performed using mainly them: features are placed somewhat between the real world and the computational world.

There are two main kinds of features:

- Local features: there are as many features as the size of input. These kind of features are often based on neighbors, so they can describe a particular set of adjacent points. If we take a 2D image for example, a local feature for a point could be the gradient of brightness with neighbors.
- Global features: there is just one descriptor for the entire dataset. If we take a 2D image for example, a global feature could be the average brightness or the average color, or a data structure containing many of this global characteristics.

3.3.1 PFH - (Persistent) Point Feature Histogram

PFH descriptors are a local features based on surface normals, XYZ 3D data and curvature, and they are capable to store the representation of the geometry around a specific point.

They contain an approximation of the geometry of a point using information about its k-nearest neighbors that are interconnected in a mesh (k^2connections).

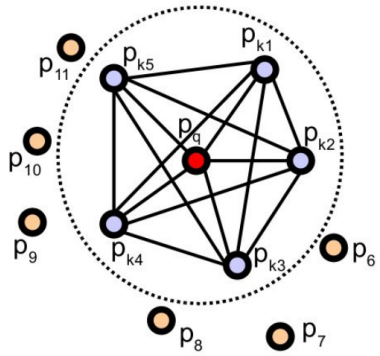

Figure 3.12: For the point $p_q$each edge of the mesh of the k-neighbors is computed

The pcl::PFHEstimation class is the PCL implementation that computes this kind of feature.

3.3.1.1 Input

- 3D x, y, z point coordinates (n points)
- Normals of the input points, computed with a normal radius r_n
- Feature radius $r_f \geq r_n$ of the sphere around the point in which find neighbors or number of neighbors to consider (k neighbors).

3.3.1.2 Output

The result of the function is an array of 125 float values (125 bytes) that represents the histogram of the so called "bins" which encodes the neighborhood's geometrical properties and provides an overall point density and pose invariant multi-value feature. This feature is stored in the pcl::PFHSignature125 type.

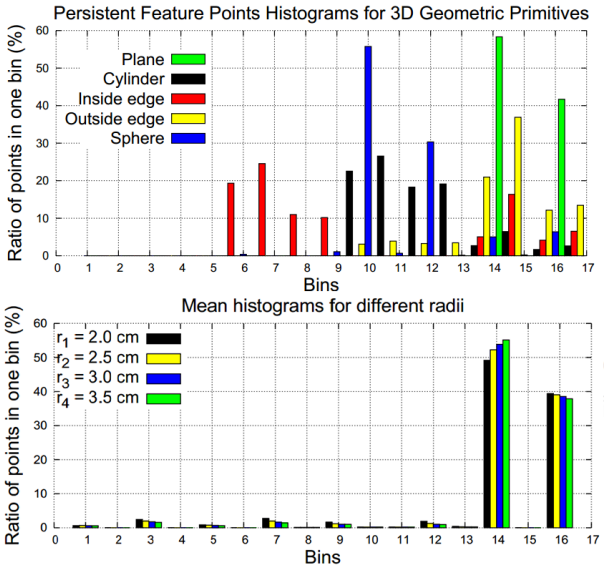

Figure 3.13: PFH histograms: different geometrical patterns (top), and computation with different radii (bottom).

3.3.1.3 Complexity $O(n \cdot k^2)$

Since all pairs of neighbors are taken into account, for each of the n points, there are k^2 pairs to consider.

The overall complexity of the algorithm is $O(n \cdot k^2)$

3.3.2 FPFH - Fast Point Feature Histogram

This algorithm computes the Simplified-PFH (SPFH) and the Fast-PFH (FPFH) reducing drastically the computation complexity and time comparing to the computation of the PFH, but maintaining most of its information.

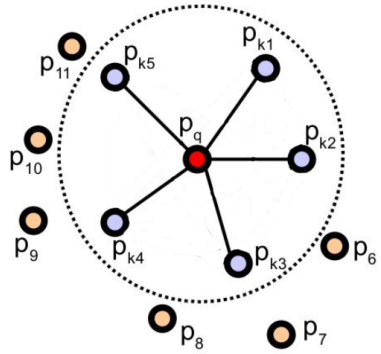

Figure 3.14: For the point $p_q$only the edge between itself and the k-neighbors are computed

The key differences between PFH and FPFH computation procedure are:

- The PFH of the point is computed with all the mesh of its neighbors ($O(k^2)$ connections), the FPFH takes into account only the $O(k)$ connections between itself and its neighbors.
- There are two steps of computation: in the first step the SPFH of all points are computed, in the second step for each point the values of neighbor's SPFH are used to weight the final FPFH histogram. At high level we can say that:
  - The PFH of the point contains all and only the information provided by its k neighbors (at distance r).
  - The FPFH of the point does not contain all the relationships between its neighbors, and contains some relationship between its neighbor's neighbors (at distance 2r).

Most of the discriminative power of the FPFH is retained, but with no doubts some fine details are lost.

3.3.2.1 Input

The input of this function is the same as the input of PFH features. Note that if we take the radius used to compute normals as 80% of the radius used to compute features, there is only one radius parameter in this calculation.

$$\begin{cases} r_n = 0.8 \cdot r_f \\ r_f = r_f \end{cases}$$

The effectiveness of the features really depends on the chosen radius, hence the entire procedure from the feature calculation is repeated iteratively for different radii: $r_f \in R$ [expressed in millimeters].

1. for each feature radius $r_f \in R$
   (a) find world and object FPFH features using radius r_f
   (b) proceed to next steps...

3.3.2.2 Output

The result of the function is an array of 33 float values (33 bytes) that represents the FPFH histogram, that is stored in the pcl::PFHSignature33 type.

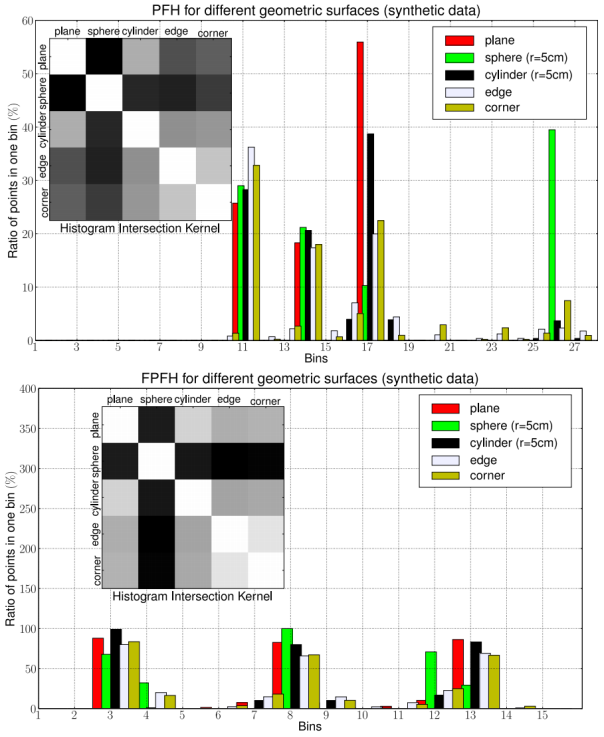

Figure 3.15: Comparison between PFH (top) and FPFH (bottom). They are different, but the pattern in the B/W square are very similar.

3.3.2.3 Complexity $O(n \cdot k)$

Since only point-to-neighbors connections are taken into account, for each of the n points, there are k pairs to consider.

The overall complexity of the algorithm is $O(n \cdot k)$

3.3.2.4 The parallel implementation

In the PCL library there is an optimized implementation of the algorithm that is multi threaded using Open MP library: pcl::FPFHEstimationOMP.

This class speeds up the overall performance by a factor of the number of processors in the machine, that is a big deal.

3.3.3 VFH - Viewpoint Feature Histogram

The Viewpoint Feature Histogram (VFH) descriptor is a global feature estimated on a point cloud using points, normals and FPFH features. The pcl::VFHEstimation class is used to find this kind of feature

The default VFH implementation contains:

- 45*3 binning subdivisions for the extended FPFH values (that are computed on the entire cloud).
- 45 binning subdivisions for the distances between each point and the centroid
- 128 binning subdivisions for the viewpoint component, that uses difference between viewpoint normal and surface normal.

which results in a 308-byte array of float values that are stored in a pcl::VFHSignature308 point type.

Having just one feature for the object could be useful, but it is extremely difficult to segment a single object in the world cloud, so the VFH feature for the world cloud contains too much noise and it is not exploitable in this case. After some testing, I have chosen not to use this type of feature.

3.4 Correspondences

After features have been computed, then the correspondences finding algorithm is performed.

A correspondence is simply a couple of features ($object.feature$; $world.feature$) that have similar corresponding values. Since features are computed at each keypoint, this step discover similarities between object keypoints and world keypoints, using features computed on them.

3.4.1 Correspondences finding

The PCL class pcl::registration::CorrespondenceEstimation can find this kind of information, given only the dataset of the object features and the world features. In particular:

- Object features (small dataset) are the ones we want to find in the world cloud, so we use them all.
- World features (big dataset) contain a lot of data, which makes it difficult to perform a robust search-and-match: it is better to use a Divide & Conquer strategy.

The problem of introducing a lot of false information and wrong data that definitely compromise the overall registration, can be avoided with few general heuristics to group the world features.

These methods have been tried.

- Global finding: find correspondences on the entire world cloud (no grouping)
- Sphere finding: find correspondences iteratively on parts of the world cloud enclosed into spheres
- Cluster finding: find correspondences iteratively on every cluster of the world cloud

3.4.1.1 Global

Finding correspondences on the entire world cloud is the first test done. It is not an heuristic, but the first brute-force procedure that tries to match all the world features with the object features.

The results are very bad, since the object features can be coupled with any world feature, at any distance: for example 2 features that are really close each other in the object, can be coupled respectively with 2 features that are very far each other in the world, that is impossible.

From this weakness comes the opportunity of grouping the dataset of world feature into sets that maintain a limited distance.

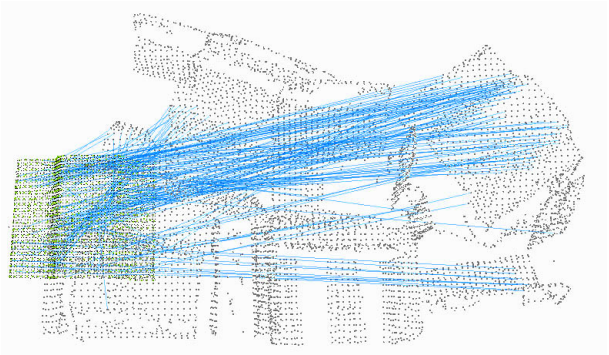

Figure 3.16: Correspondences found using all the world cloud. The distance between them is much larger than the size of the object.

3.4.1.2 Spheres

The first heuristic approach is to try with spheres: before we have to find the radius of the object, then construct the spheres in the world cloud.

We define the radius *rad* of the object as the maximum distance between the centroid of the object and its points.

1. build a 3D grid in the world cloud, with resolution *res*.
2. in each cross of the grid place the center of a sphere of radius *rad*.
3. for each sphere built:
   (a) find the correspondences between all the object features and the subset of world features that are placed inside the sphere
   (b) proceed to next steps...

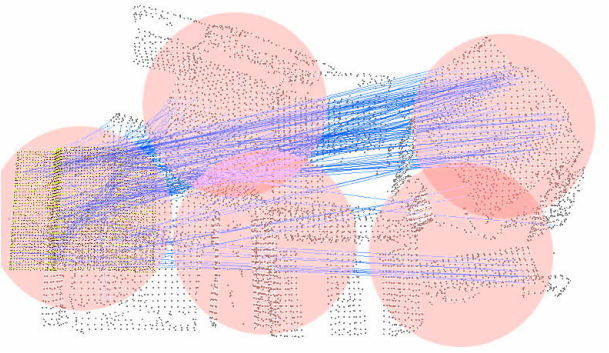

Figure 3.17: Correspondences found using spheres of world cloud. The key idea is that a good set of correspondences can not be larger than the object size.

This method seems valid, but it has a huge drawback: the performance.

In fact the number of spheres that can be constructed is:

$$n_{spheres} \approx \frac{width \cdot height \cdot depth}{res^3}$$

This huge number of spheres consumes a lot of computation time, because for each sphere a new estimation and all the following steps are performed.

Two strategies to avoid this brute-force approach are to filter spheres

3.4.1.3 Clusters

The best heuristic comes from a simple reason: the target object, as a single piece appears in the point cloud as single cluster. Splitting a point cloud into clusters means decomposing it into regions of space based on the euclidean distance between points.

The PCL function pcl::extractEuclideanClusters can perform this calculation, given:

- the input cloud
- the search method (KD-Tree)
- a spatial tolerance or threshold
- a $min_{cluster}$ cardinality of each cluster found
- a $max_{cluster}$ cardinality of each cluster found

The $min_{cluster}$ parameter can be tuned between 0 and the cardinality of the object cloud: $min_{cluster} = k \cdot \parallel object.cloud \parallel$ where $k \in [0, 1]$

The $max_{cluster}$ parameter can be tuned greater than the world point cloud cardinality : $max_{cluster} = h \cdot \parallel world.cloud \parallel$where $h \geq 1$

If the threshold is too small, the same object can be split into different clusters, if it is too big many objects can be grouped together. For this reason we have to choose the smallest threshold that does not divide a single object.

At the beginning, a good threshold has been proven to be slightly grater than the input cloud resolution (10% more): in fact with this parameter the single objects are not assigned to different clusters. During the test phase the parameter has been fine-tuned.

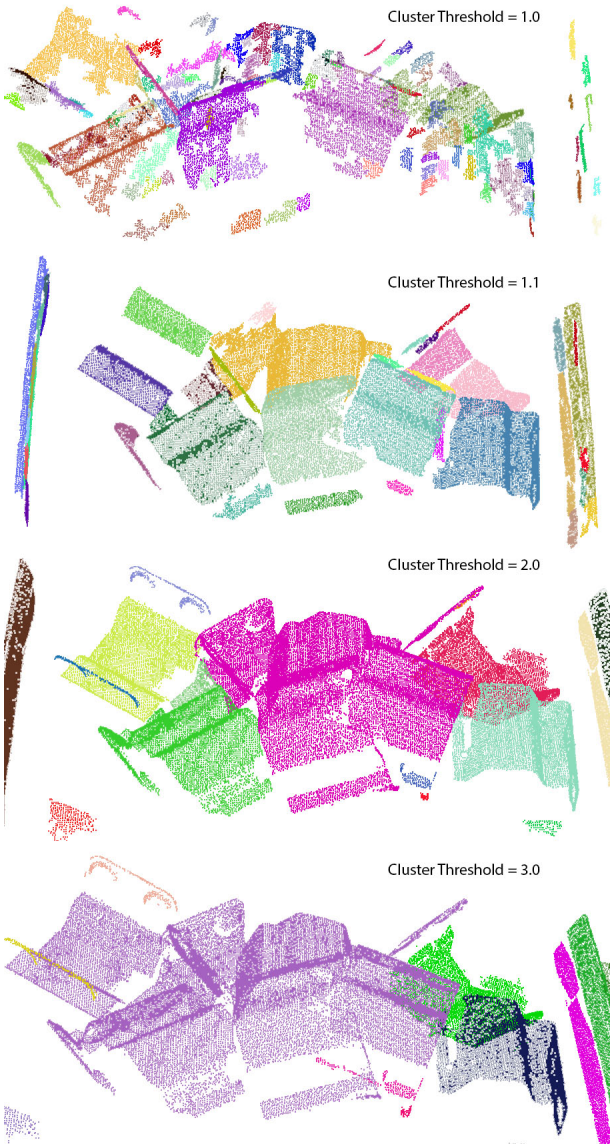

Figure 3.18: Clustering of the same cloud (resolution=1.0mm) with different tolerances:
(first) small tolerance leads to wrong clustering
(second) right tolerance can almost separate objects
(third, fourth) big tolerance make objects to be joined together

After cluster extraction has been performed, the correspondences can be found.

Here there are the summary of this method.

1. Clusterize the world cloud with tolerance $tol > res$ slightly grater than the cloud resolution res.

2. for each cluster found:

   (a) find the correspondences between all the object features and the subset of world features that belong to the cluster

   (b) proceed to next steps...

The major benefits using this approach instead the spheres is that there number of clusters is much lower, so the computation maintains the required performance.

$$n_{clusters} = O(\frac{\parallel world.cloud \parallel}{min_{cluster}}) \ll n_{spheres}$$

3.4.2 Correspondences filtering

When correspondences have been found it is possible to reject some of them based on specific condition that will be now presented. The base PCL class that does this job is pcl::registration::CorrespondenceRejector that is inherited by all following classes.

3.4.2.1 Rejector by Distance

All the correspondences that exceed a distance threshold are rejected. The correspondence distance is the measure between corresponding keypoints of object and world clouds.

This method is not good for our task, because an object in the world cloud can be both near to or far from the object cloud.

3.4.2.2 Rejector by Median Distance

The median distance between all the correspondences is computed, then all the correspondences that exceed a deviation threshold from that mean value are rejected.

This method too is not good for our task for the same reason of the distance rejector

3.4.2.3 Rejector by Feature

The pcl class implements a correspondence rejection method based on a set of feature descriptors. Given an input feature space, the method checks if each

feature in the object cloud has a correspondence in the world cloud, either by checking the first K (given) point correspondences, or by defining a tolerance threshold via a radius in feature space.

This method has been put aside because the results were not good for the task.

3.4.2.4 Rejector by Surface Normal

The pcl class implements a simple correspondence rejection method based on the angle between the normals at correspondent points. This method could not be tried because of some bugs in the pcl library v.1.6 that created collisions between header files, preventing the compile step to end successfully.

3.4.2.5 Rejector by Random Sample Consensus Model

The pcl class implements a correspondence rejection using Random Sample Consensus to identify inliers (and reject outliers). RANSAC is a non-deterministic iterative method to estimate parameters of a mathematical model, from a set of observed data which contains outliers. In this case the model is the object, and the observed data is the world cluster.

The rejector has some input parameters:

- the object cloud and the world cloud
- the non filtered correspondences given by the previous step
- the number of iterations, set as 50
- the inlier threshold th, a parameter used internally to discriminate inliers to outliers.

This threshold is really sensible, and the result of the function depends hardly on this parameter, that ranging in the interval $th \in T$, affects unpredictably the output.

Since the values in that range are good to try RANSAC method, all of them are tried (integer numbers) iteratively, taking only the threshold that gave the best result.

Hence the pseudo-code of the correspondence filtering is:

1. for each inlier threshold $th \in T$
   (a) filter correspondences with RANSAC algorithm performing 50 iterations with threshold th
   (b) proceed to next steps...

3.5 Initial Alignment

3.5.1 Using features and correspondences

The initial alignment, that is the 4x4 roto-translation matrix of the rough alignment, is computed using the filtered correspondences.

There are two classes in the PCL library that can estimate this transformation, without any parameter:

- pcl::registration::TransformationEstimationLM - (Levenberg-Marquardt): this is an iterative least-squares based algorithm. The result of this transformation does not bring always the best score
- pcl::registration::TransformationEstimationSVD - (Singular Value Decomposition): this is a closed-form solution based on the singular value decomposition of a covariance matrix of the data, providing the best possible solution in a single step. The results are very good, so this method has been adopted for this work.

This procedure proved to be the most effective because it is general purpose and based on local features that can discriminate the important parts of any object.

3.5.2 Using Principal Component Analysis

Principal Component Analysis, or simply PCA, is a statistical procedure concerned with extrapolating data from the covariance structure of a set of variables.

In our case PCA can find out, basing on the 3D dataset, a coordinate system, or the three orthogonal axis in which the data varies more. That is, the output of the procedure is the set of three eigenvectors:

- U that represent the principal direction
- V that is the second most important direction, perpendicular to U
- W the third direction, perpendicular to U and V

Vectors U, V, W represent the orthonormal 3D basis and are called the principal components. If we transform each (X, Y, Z) coordinate into its corresponding (U, V, W) value, the data is decorrelated, meaning that the covariance between each couple of variables is zero.

Summarizing with few words, for a given set of data principal component analysis can find the axis system defined by the principal directions of variance.

With the use of PCA it is possible to find the main directions of the object model cloud AND of the input cluster of the world cloud: then it is possible to compute a transformation that aligns the axis found. The result can be effective especially on particular objects, where a principal direction is clearly distinguishable. For such considerations, this procedure is not so general purpose, introducing a lot of bias due to wrong clustering, and it will be not considered anymore in this work.

3.6 Fine Alignment

The final alignment, that is the 4x4 roto-translation matrix that perfectly matches the point clouds, is computed using object cloud and the world cloud.

Not that the input is the entire world cloud, and not the corresponding cluster found in the initial procedure.

3.6.1 ICP - Iterative Closest Point

When the initial rough alignment has been performed, the algorithm switches to the final procedure that is done by the ICP algorithm, implemented in the PCL library.

ICP works by iteratively minimize distances between internally found correspondences. In each iteration a new transformation matrix is computed. Correspondences used are found only on near points (there is a threshold) and then only small transformation is computed at each iteration.

The algorithm has 3 termination conditions:

- The number of iterations has reached the maximum specified by user
- The epsilon change value between the two last iteration is smaller than a value specified by the user
- A fitness function computed internally has reached a threshold specified by the user

Only the first two condition have been set: 50 maximum iterations and an epsilon equal to 10^{-7}

3.6.2 ICP-NL - Iterative Closest Point - Non Linear

This is an ICP variant that uses non linear Levenberg-Marquardt optimization backend. The results provided by this function did not give appreciable results.

Chapter 4

Recognition Algorithms

Here there are the pseudo-code of the final algorithms using all the best procedures among all the ones described in the previous chapter.

4.1 Algorithm 1: separated FPFH

In this algorithm there are many initial transformations for the same world cluster: the one with best score is elected to be the best initial transformation for that cluster. Each transformation is computed on correspondences coming from a set of FPFH with the same radius.

- The advantage is:
  - we have as many attempts as the number of radii used in the FPFH computations. Example: for cluster A the best correspondences are found using radius X for FPFH, for cluster B the best correspondences are found using radius Y for FPFH. The more number of attempts, the more probability to find a good transformation.
- The disadvantages are:
  - the number of filtered correspondences is low and subject to noise
  - we have more computation complexity because there are 3 nested loops
  - the optimal set of correspondences may be (surely is) made up of correspondences that are found using different radii for FPFH

Algorithm 4.1 Multiple object recognition: algorithm 1

1. Initial alignment: for each cluster of the world cloud:
   (a) find uniform keypoints on object and world cluster cloud (initial leafSize)
   (b) for each FPFH radius:
      i. find FPFH features on object and world cluster cloud
      ii. find correspondences between computed features
      iii. for each SAC threshold:
         A. filter correspondences with RANSAC method
         B. estimate transformation matrix that minimizes distances
         C. compute euclidean distance validation score
         D. keep the transformation with best score
2. keep only initial transformations with a good score
3. Final alignment:
   (a) find uniform keypoints on object and world cloud (final leafSize)
   (b) perform ICP algorithm on keypoints to find the final transformation
   (c) compute percent of outliers validation score
   (d) if the score is better than before, then transform
4. keep only final transformations with a good score
5. compute transformation matrices by multiplying initial and final matrices
6. return matrices and final scores

Obviously some details for performance purposes have been omitted, in order to maintain simplicity in the explanation of the algorithm.

4.1.1 Performance

$$O[n_{clusters} \cdot n_{FPFH.Radii} \cdot n_{SAC.Thresholds} + ICP_{complexity}]$$

Note that there are 3 nested loops, and the performance depends mainly on the settings of the following three parameters: the cluster tolerance that discriminates the number of clusters found, the cardinality of the set of FPFH radii, the cardinality of the set of SAC thresholds.

4.2 Algorithm 2: merged FPFH

In this algorithm there are only one initial transformations for each world cluster.

Each transformation is computed on correspondences coming from sets of FPFH with different radii

- The advantage are:
  - in the initial transformation we have the best correspondences using information about neighbors at different radii
  - the set of correspondences is so big that the noise of outliers is unrelevant
  - there are only 2 innested loops, that improves slightly the computation time
- The disadvantage are:
  - that we have only one try: if the initial step goes wrong we will not be able to continue to final alignment

Algorithm 4.2 Multiple object recognition: algorithm 2

1. Initial alignment: for each cluster of the world cloud:
   (a) find uniform keypoints on object and world cluster cloud (initial leafSize)
   (b) for each FPFH radius:
      i. find FPFH features on object and world cluster cloud
      ii. find correspondences between computed features and add to the total correspondences set
      iii. filter correspondences maintaining the best for the same feature
   (c) (at this point we have only one big set of correspondences)
   (d) for each SAC threshold:
      i. filter correspondences with RANSAC method
      ii. estimate transformation matrix that minimizes distances
      iii. compute euclidean distance validation score
      iv. keep the transformation with best score
2. keep only initial transformations with a good score
3. Final alignment:
   (a) find uniform keypoints on object and world cloud (final leafSize)
   (b) perform ICP algorithm on keypoints to find the final transformation
   (c) compute percent of outliers validation score
   (d) if the score is better than before, then transform
4. keep only final transformations with a good score
5. compute transformation matrices by multiplying initial and final matrices
6. return matrices and final scores

Obviously some details for performance purposes have been omitted, in order to maintain simplicity in the explanation of the algorithm.

4.2.1 Performance

$$O[n_{clusters} \cdot (n_{FPFH.Radii} + n_{SAC.Thresholds}) + ICP_{complexity}]$$

Note that there are only 2 nested loops, and the performance depends mainly on the settings of the following parameters: the cluster tolerance that discriminates the number of clusters found and the cardinality of the set of FPFH radii.

4.3 Free parameters

Here are now described the numerical parameters that have been chosen, but can be modified as the programmer wants.

1. Resolution of the scan (initial voxel grid filter) = 1.0 [mm]
2. Cluster tolerance, as percentage (>100%) of the resolution of the scan = 150%
3. Cluster min size, as percentage (<100%) of the object size = 30%
4. Cluster max size, as percentage (>100%) of the object size = 300%
5. Initial leafSize to calculate keypoints for initial transformation = 3.0 [mm]
6. Radii of FPFH to try = $R = \{10, 15, 20, 25, 30\}$ [mm]
7. Normal radius to compute FPFH, as percentage (<100%) of the FPFH radius = 80%
8. RANSAC inlier thresholds to try = $T = 1, 2, 3, ..., 18, 19, 20$ [mm]
9. Euclidean distance score threshold for initial alignment = 100.0 [no-dimensional score]
10. Final leafSize to calculate keypoints for initial transformation = 2.0 [mm]
11. ICP termination conditions= 50 iterations or 10^{-7}epsilon
12. Percent of outliers score threshold for final alignment = 20%

With these parameters set, the algorithm is really general purpose for many small objects of a volume around $10^{-3}m^3$.

If the object increases in size, different (greater) parameters for 1. 4. 5. 7. 9. of the list (metric measures) should be set.

4.4 Parallelization with OpenMP

The grade of parallelism that can be achieved in this work is very high, both because we are working on point clouds (that are structures based on sets of not ordered N-dimensional points) and we are dealing with clusters that do not require to be computed sequentially.

4.4.1 The OpenMP API

The OpenMP Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms. Jointly defined by a group of major computer hardware and software vendors, OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer.

4.4.2 The parallel for loop constructs

Algorithm 4.3 How does the code change between a sequential and a OpenMP parallel for loop

the case of the sequential for loop

1. for (int i=0 ; i<N ; i++)
   (a) Do stuff...

The case of the OpenMP parallel for loop

1. #pragma omp parallel for
2. for (int i=0 ; i<N ; i++)
   (a) Do stuff...

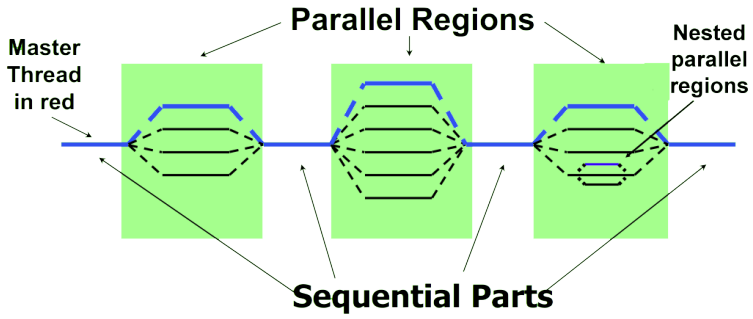

Figure 4.1: Workflow of a parallel algorithm with OpenMP.
Note the master thread (blue) that is always active and manages the assigned team (black)

4.4.2.1 Parallel

An OpenMP program begins as a single thread of execution, called the initial thread. The initial thread executes sequentially, as if enclosed in an implicit task region, called the initial task region, that is defined by an implicit inactive parallel region surrounding the whole program.

When any thread encounters a *"parallel"* construct, the thread creates a team of itself and zero or more additional threads and becomes the master of the new team. A set of implicit tasks, one per thread, is generated. The code for each task is defined by the code inside the *parallel* construct. Each task is assigned to a different thread in the team and becomes tied; that is, it is always executed by the thread to which it is initially assigned. The task region of the task being executed by the encountering thread is suspended, and each member of the new team executes its implicit task. There is an implicit barrier at the end of the *parallel* construct. Only the master thread resumes execution beyond the end of the *parallel* construct, resuming the task region that was suspended upon encountering the *parallel* construct. Any number of *parallel* constructs can be specified in a single program.

4.4.2.2 For Loop

The loop construct specifies that the iterations of one or more associated loops will be executed in parallel by threads in the team in the context of their implicit tasks. The iterations are distributed across threads that already exist in the team executing the parallel region to which the loop region binds.

There are several scheduling options, but leaving the settings as "auto" makes the compiler and the runtime environment choose how to distribute the iterations across threads. Most of the time all the n iterations are divided into the m number of processors and each block of the loop is executed at the same time, giving a best-case speedup factor equal to the number of processors.

Chapter 5

Tests & Results

5.1 Objects and datasets

5.1.1 Description of the objects

5.1.1.1 Object A

It is a sheet of steel bent strongly to form angles close to 90°. Height, width and depth are comparable.

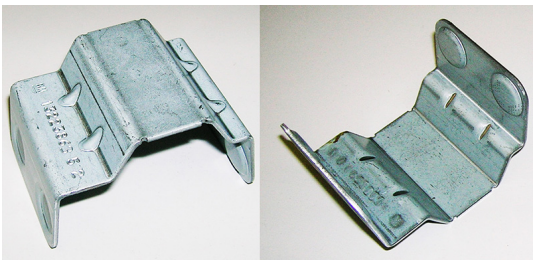

Figure 5.1: Object A

The models used for the matching are the following: one is the entire view of the object from the top, the other is the cut of the minimal visible part that has to match with the world cloud

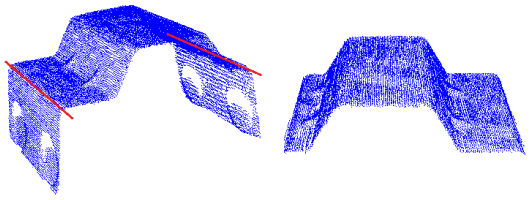

Figure 5.2: Model used for object A in the algorithm:
(left) Entire object viewed from the top
(right) Cut of the object that represents the minimal part that has to match

5.1.1.2 Object B

It is a sheet of steel slightly bent, it can be considered as a planar object. depth is minimal.

Figure 5.3: Object B

The model used for the matching is the following.

Figure 5.4: Model used for object B in the algorithm:

5.1.1.3 Object C

It is a pipe of steel bent in two points.

Figure 5.5: Object C

The models used for the matching are the following: one is the entire view of the object from the top, the other is the cut of the minimal visible part that has to match with the world cloud. Note that with the entire object used as model, the algorithm cannot align the objects posed on the other side: in fact the model is taken only from a point of view. For this special case, the pick is good even if the object is posed on the other side, hence the symmetrical model has been obtained by cutting the edges of the entire one.

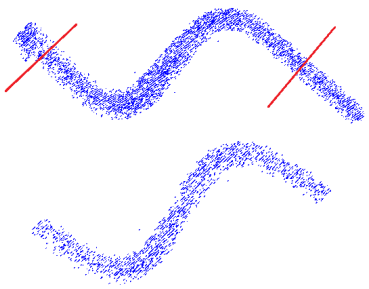

Figure 5.6: Model used for object C in the algorithm:
(top) Entire object viewed from the top
(bottom) Cut of the object that represents the minimal part that has to match

5.1.2 Description of the dataset

5.1.2.1 Conveyor belt scans

- 10 scans for object A
- 10 scans for object B

- 10 scans for object C
- 10 scans for object A + B
- 10 scans for object B + C
- 10 scans for object C + A
- 10 scans for object A-Decr
- 10 scans for object B-Decr
- 10 scans for object C-Decr

Totaling 90 scans.

The operator "+" means that the objects of different types are posed together under the laser system.

The tag "-Decr" means that consecutive scans are taken decrementing the number of objects by one at each scan.

5.1.2.2 Robotic arm scans

For these scans object B has been ignored: we concentrate on the objects A and C, so B is used only to provide a random basement for the other objects.

- 10 scans for object A
- 10 scans for object A over B
- 10 scans for object A over B and C
- 10 scans for object C
- 10 scans for object C over B
- 10 scans for object C over B and A
- 20 scans for object A+B+C mixed
- 20 scans for object A+B+C-Box mixed inside a box

Totaling 100 scans.

Each object has 70 input clouds that contain several instances of it.

5.2 The machine used for tests

The physical machine used has the following specifics:

- AMD Phenom II x4 core, 3.0Ghz
- 4GB RAM 1666Mhz
- Hard Disk drive 160Gb 7200rpm
- OS Microsoft Windows 7 Professional

Graphic card is not used for computation, but only for the visualization widget, so the details of GPU is omitted.

Then a virtual machine has been created on this host system using VMWare Player tools. The assigned resources to guest system are the following:

- 4 Processors (sharing with the host)
- 2GB RAM
- Hard Disk drive 16Gb
- OS Ubuntu Linux 12.04 LTS with XFCE, a lightweight Desktop Manager

Test have been performed on the virtual machine, keeping the host system as idle as possible.

5.3 Determining the best cluster parameters and the best algorithm

In this section I will present the tests conducted on objects A and B to find out the best parameters to use with clusters and the best algorithm among 1 and 2.

The dataset used is only the one taken with the conveyor belt.

5.3.1 The cluster threshold

One of the most important parameters in the overall work is the cluster threshold: setting it too small will split the same objects into different parts, setting it too large will merge different objects in the same calculation with high probability of false positives.

This parameter must be greater than the resolution of the cloud when it is loaded.

Since this resolution has been set to 1.0, the study of the parameter has been conducted in the range [1.1 ; 1.9] with a precision of 0.1.

5.3.1.1 Tests on object A

This test is performed on 40 input samples: A, A+B, C+A, A-Decr with the following settings:

- input Voxel Resolution= 1.0
- min/max Cluster Size relative to object size= 0.2 / 10.0
- initial/final ScoreThreshold= 200.0 / 20.0
- initial/final Resolution= 3.0 / 2.0
- Radii FPFH $R = \{10, 15, 20\}$
- SAC Thresholds$T = 1, 3, 5, ..., 15, 17, 19$
- OpenMP parallelism: active

The results are presented in a graph form:

- The total computation time includes loading of PCD files (approx 10Mb each) and complete registration process for the entire set of 40 input clouds.
- The recognition rate is given by the division of the number of objects totally recognized by the number of files processed.
- The average initial score is the Euclidean Distance validation score computed at the end of the initial alignment
- The average final score is the Percent of Outliers validation score computed at the end of the final alignment

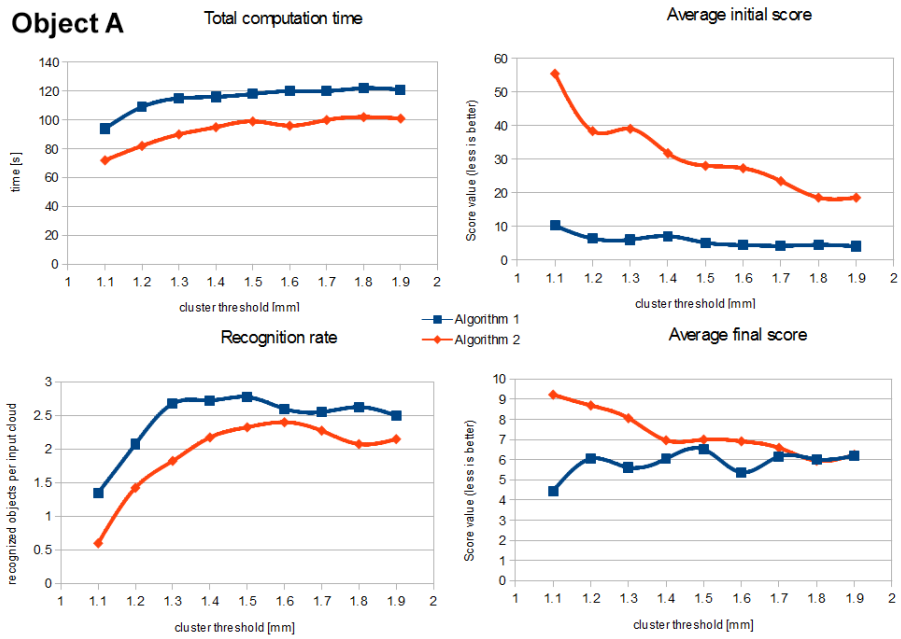

Figure 5.7: Test results to determine the cluster parameters and the best algorithm:
(top left) Algorithm 2 is faster
(bottom left) Algorithm 1 finds more objects
(top right) Algorithm 1 is more accurate in the initial phase
(bottom right) There is no significant difference between final scores

5.3.1.2 Tests on object B

This test is performed on 40 input samples: B, A+B, B+C, C-Decr with the following settings:

- input Voxel Resolution= 1.0
- min/max Cluster Size relative to object size= 0.4 / 10.0
- initial/final ScoreThreshold= 200.0 / 20.0
- initial/final Resolution= 3.0 / 2.0
- Radii FPFH $R = \{10, 15, 20\}$
- SAC Thresholds$T = 1, 3, 5, ..., 15, 17, 19$
- OpenMP parallelism: active

The results are presented in a graph form:

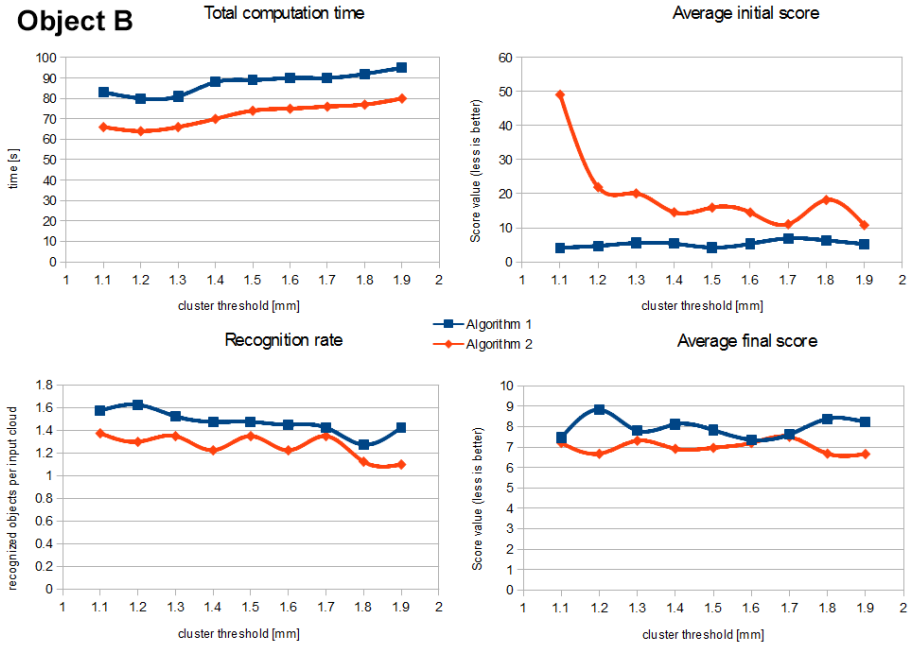

Figure 5.8: Test results to determine the cluster parameters and the best algorithm:
(top left) Algorithm 2 is faster
(bottom left) Algorithm 1 finds more objects
(top right) Algorithm 1 is more accurate in the initial phase
(bottom right) There is no significant difference between final scores

5.3.1.3 Conclusions

The threshold too close to 1 produces high initial scores, which means poor registration quality. On the other hand the more it increases, the more time is consumed and the less recognition rate is shown.

The best setting for this parameter is 1.5.

5.3.2 The cluster min-max parameter

These parameters affect the number of cluster tried in the initial alignment and consequently the time of the computation.

The parameters have these constraints: $0 < min_{cluster} < 1$ and $max_{cluster} > 1$ and can be chosen according to the object type:

- $min_{cluster}$: if the object is clearly distinguishable from the others and does not have overlapping/hidden parts it is better closer to 1. Elsewhere

if the object is more "difficult" to segment clearly it is better closer to 0. In this test: $min_{cluster}objectA = 0.2$ and $min_{cluster}objectB = 0.4$.

- $max_{cluster}$: with this parameter set as low as possible (recommended at least 2 or 3) we can avoid doing calculation on too large clusters. In this test the parameter has been set to 10.

5.3.3 The best algorithm

The difference between the two algorithms consists on the initial alignment, and there is no doubt that the Algorithm 1 has a better performance in terms of recognition rate that is due to the big difference of initial score between the two. Obviously this performance requires an average of 20% more computation time.

5.4 False negative / False positives

We are now going to evaluate the performance of the algorithm basing on the rates of false positives and false negatives.

5.4.1 False positives

There is a false positive when the algorithm classifies as good an alignment that is not.

In our case a good alignment is able to overlap correctly the object with an instance of it in the world cloud. On the other hand a false positive can show a wrong rotation of 180° or any other wrong alignment.

There are some particular alignments that find an object that cannot be picked, that is hardly occluded by other objects. In the test I have split the classification between false positives and non-pickable objects (because they are substantially correct)

5.4.2 False negatives

There is a false negative when the algorithm classifies as bad a registration on a cluster which contains a visible object. That is, when a visible object is skipped by the algorithm.

5.4.3 Classification by cycles

A cycle of computation is defined by the run of the algorithm on a particular world cloud: the acquisition system provides to the algorithm a point cloud on which the algorithm is performed.

5.4.3.1 Good cycle

A cycle is defined good if the algorithm is able to recognize a pickable object in the world cloud, labeling it with the best score threshold.

5.4.3.2 Bad cycle

A cycle is defined bad if the algorithm enter one of these 2 states:

- False best: The object with best final score IS a false positive or a not pickable object: in this case the picking system or the final product would be subject to damage.
- Starvation: NO object among all the "present" can be recognized: in this case the picking system would be lead to starvation.

When no objects of the world cloud are pickable, then the classification is NOT considered as a bad cycle.

5.5 Final tests on Conveyor Belt Data

For this section the following parameters remain fixed:

- Algorithm = 1
- input Voxel Resolution= 1.0
- cluster threshold relative to input voxel resolution= 1.5
- min/max Cluster Size relative to object size= 0.3 / 10.0
- SAC Thresholds$T = 1, 3, 5, ..., 15, 17, 19$
- OpenMP parallelism: active

5.5.1 Configuration for tests

5.5.1.1 On object A

This test have been made using the model for object A (not cut).

The dataset used is taken with the conveyor belt system: A, A+B, A+C, A-Decr

- A:
  - initial/final Resolution= 3.0 / 2.0
  - initial/final ScoreThreshold= 200.0 / 20.0
  - Radii FPFH $R = \{5, 10, 15, 20\}$

5.5.1.2 On object B

This test have been made using the only model available for object B

The dataset used is taken with the conveyor belt system: B, A+B, B+C, B-Decr

- B:
  - initial/final Resolution= 3.0 / 2.0
  - initial/final ScoreThreshold= 200.0 / 20.0
  - Radii FPFH $R = \{5, 10, 15, 20\}$

5.5.1.3 On object C (C1 - C2 - C3)

The following tests have been made using the model for object C (not cut). Symmetric objects are not considered good.

The dataset used is taken with the conveyor belt system: C, A+C, B+C, C-Decr.

- C1:
  - initial/final Resolution= 3.0 / 2.0
  - initial/final ScoreThreshold= 200.0 / 20.0
  - Radii FPFH $R = \{5, 10, 15, 20\}$

This model used is the same.

The dataset used is the same.

- C2:
  - initial/final Resolution= 2.0 / 1.5
  - initial/final ScoreThreshold= 200.0 / 25.0
  - Radii FPFH $R = \{5, 9, 13, 17\}$

The following test has been made using the CUT model for object C. Consequentially the number of pickable objects grows.

The dataset used is the same.

- C3:
  - initial/final Resolution= 2.0 / 1.5
  - initial/final ScoreThreshold= 100.0 / 15.0
  - Radii FPFH $R = \{3, 6, 9, 12\}$

5.5.2 Test results

Results are presented here in absolute numeric form:

Property	Symbol	A	B	C1	C2	C3
N. of files processed	F	40	40	40	40	40
N. of pickable objects in the cloud	P	102	79	80	80	149
Subset of the pickable objects recognized	R	88	58	46	64	104
False Negatives (not recognized)	FN	14	21	34	16	45
Files with at least one false positive (wrong alignment)	FP	1	0	19	28	17
Files with at least one unpickable objects (aligned correctly)	NP	9	5	6	11	10
Bad cycle due to false best (wrong best object)	FB	0	0	9	7	6
Bad cycle due to starvation (no objects)	S	1	3	4	2	1

Then the compared results in percent form are:

Property	Formula	A	B	C1	C2	C3
Recognition rate	$\frac{R}{P}\%$	86%	73%	58%	80%	70%
False Negatives (not recognized)	$\frac{FN}{P}\%$	14%	27%	42%	20%	30%
Percent of clouds that experience false positives	$\frac{FP}{F}\%$	2%	0%	47%	70%	42%
Percent of clouds that experience unpickable objects	$\frac{NP}{F}\%$	22%	12%	15%	27%	25%
Bad cycle due to false best (wrong best object)	$\frac{FB}{F}\%$	0%	0%	22%	17%	15%
Bad cycle due to starvation (no objects)	$\frac{S}{F}\%$	2%	7%	10%	5%	2%

Note that with different values of the parameter, the last row will converge to 0%: that is some object will be surely found, but the risk to find false positive increases, just like the computation time.

5.6 Final tests on Robotic Arm Data

For this section the following parameters remain fixed:

- Algorithm = 1
- input Voxel Resolution= 1.0
- cluster threshold relative to input voxel resolution= 1.8
- min/max Cluster Size relative to object size= 0.3 / 5.0
- SAC Thresholds$T = 1, 3, 5, ..., 15, 17, 19$
- OpenMP parallelism: NOT active

Note that:

1. The cluster threshold has been increased because the resolution given by the robotic arm scans is much smaller than the one provided by the conveyor belt scans.
2. The max cluster size has been decreased to 5.0 for speed (too big clusters are ignored).
3. OpenMP is not active because we want to measure the effective time complexity of each part of the registration process.

5.6.1 Configuration for tests

5.6.1.1 On object A

The following test has been made using the model for object A (not cut).

The dataset used is taken with the robotic arm system: A, A overB, A overBC, ABC, ABC Box

- A:
  - initial/final Resolution= 3.0 / 2.0
  - initial/final ScoreThreshold= 100.0 / 20.0
  - Radii FPFH $R = \{5, 10, 15, 20\}$

5.6.1.2 On object C

The following test has been made using the CUT model for object C. The symmetry of the object makes each side recognizable.

The dataset used is taken with the robotic arm system: C, C overB, C overAB, ABC, ABC Box

- C:
  - initial/final Resolution= 2.0 / 1.5
  - initial/final ScoreThreshold= 100.0 / 15.0
  - Radii FPFH $R = \{3, 6, 9, 12\}$

5.6.2 Test results

Results are presented here in absolute numeric form:

Property	Symbol	A	C
N. of files processed	F	70	70
N. of pickable objects in the cloud	P	234	263
Subset of the pickable objects recognized	R	198	151
False Negatives (not recognized)	FN	36	112
Files with at least one false positive or unpickable	FP	14	40
Bad cycle due to false best (wrong best object)	FB	2	7
Bad cycle due to starvation (no objects)	S	0	5

Then the compared results in percent form are:

Property	Formula	A	C
Recognition rate	$\frac{R}{P}\%$	85%	57%
False Negatives (not recognized)	$\frac{FN}{P}\%$	15%	43%
Percent of clouds that experience false positives	$\frac{FP}{F}\%$	20%	57%
Bad cycle due to false best (wrong best object)	$\frac{FB}{F}\%$	3%	10%
Bad cycle due to starvation (no objects)	$\frac{S}{F}\%$	0%	7%

Note that the quality of the scan for robotic arm was not so good, so the object C has poorer results due to the fact that it is smaller.

Moreover, Object C even if smaller, it is heavier so during the random movements of different object it tended to place beneath the other objects, making it more occluded.

5.7 Time Performance Tests

This tests have been made by changing the number of processors assigned to the virtual machine where the software runs:

- Single processor: there is only one processor assigned to the VM. Normals, FPFH features, are all computed in a single thread.

- Multi processor (x4) without parallelism: the VM has 4 processors but the OpenMP feature is disabled. Hence Only normals and FPFH features are computed using all processors.
- Multi processor (x4) with parallelism: with this feature enabled, each cluster is processed by a separate thread, that is assigned to a different processor. Hence the overall computation time is reduced a lot, while the normals and FPFH features time consumption increases because other processors are busy with their assigned cluster.

Note that each time presented here contains also the computation of keypoints, FPFH of the model: this data can be precomputed offline, reducing the computation time of the algorithm.

5.7.1 On conveyor belt data

This calculation has been performed basing on 20 different clouds of object A with the same configuration chosen in the final tests.

5.7.1.1 Single processor Virtual Machine

The average time for an input sets is 7.0 s

Percentage are shown in the pie chart

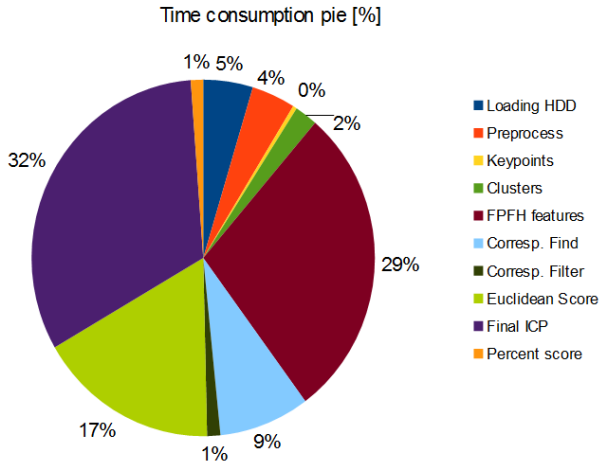

Figure 5.9: Pie chart with the time consumption percentage for each step of the algorithm

5.7.1.2 Multi processor (x4) VM without parallelism

The average time for an input sets: 5.6 s

Percentage are shown in the pie chart

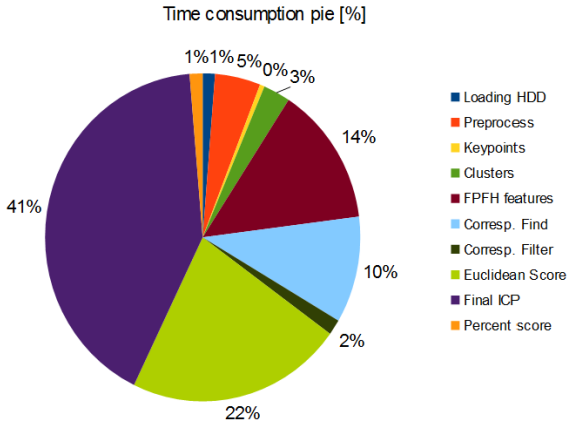

Figure 5.10: Pie chart with the time consumption percentage for each step of the algorithm

5.7.1.3 Multi processor (x4) VM with parallelism

The average time for an input sets: 2.9 s

It is not possible to see each phase of the algorithm because they are mixed together. Hence in this pie chart i had to merge and see the overall time percentage of the initial alignment AND the final alignment.

- Initial alignment contains: clustering, keypoints, FPFH features, Corresp. find, Corresp. filter, Euclidean score.
- Initial alignment contains: Final ICP, Percent score.

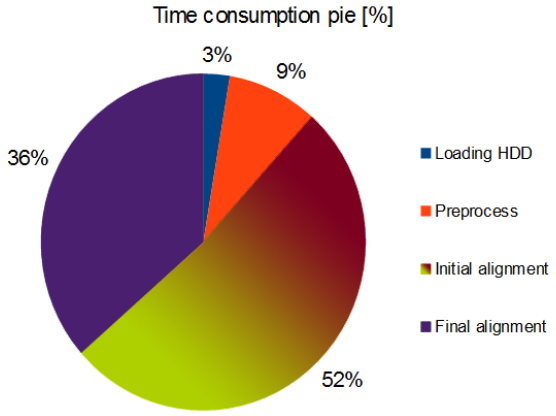

Figure 5.11: Pie chart with the time consumption percentage for the main steps of the algorithm

5.7.2 On robotic arm data

This calculation has been performed basing on 20 different clouds of object A with the same configuration chosen in the final tests.

5.7.2.1 Single processor Virtual Machine

The average time for an input sets is 14.1 s

Percentage are shown in the pie chart

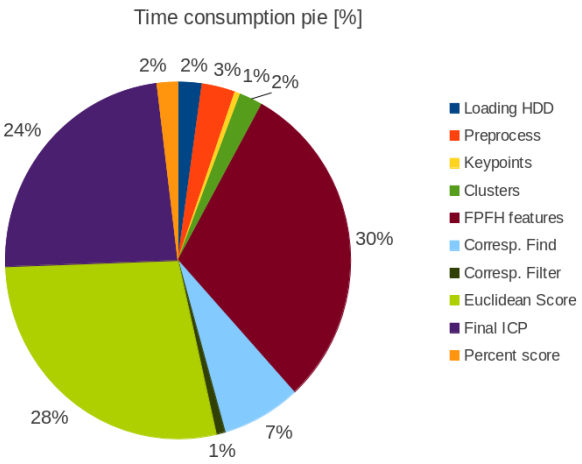

Figure 5.12: Pie chart with the time consumption percentage for each step of the algorithm

5.7.2.2 Multi processor (x4) VM without parallelism

The average time for an input sets: 12.5 s

Percentage are shown in the pie chart

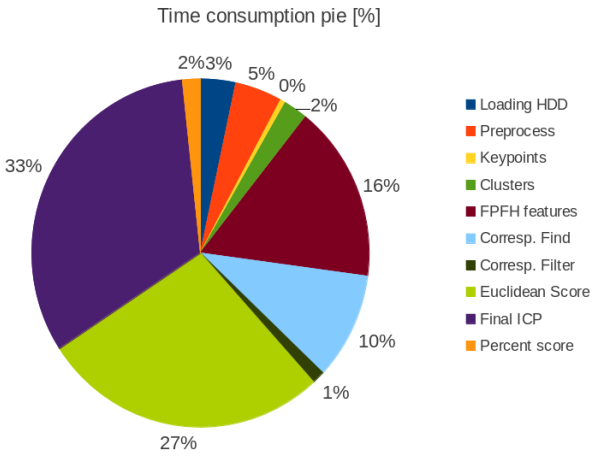

Figure 5.13: Pie chart with the time consumption percentage for each step of the algorithm

5.7.2.3 Multi processor (x4) VM with parallelism

The average time for an input sets: 5.3 s

It is not possible to see each phase of the algorithm because they are mixed together. Hence in this pie chart i had to merge and see the overall time percentage of the initial alignment AND the final alignment.

- Initial alignment contains: clustering, keypoints, FPFH features, Corresp. find, Corresp. filter, Euclidean score.
- Initial alignment contains: Final ICP, Percent score.

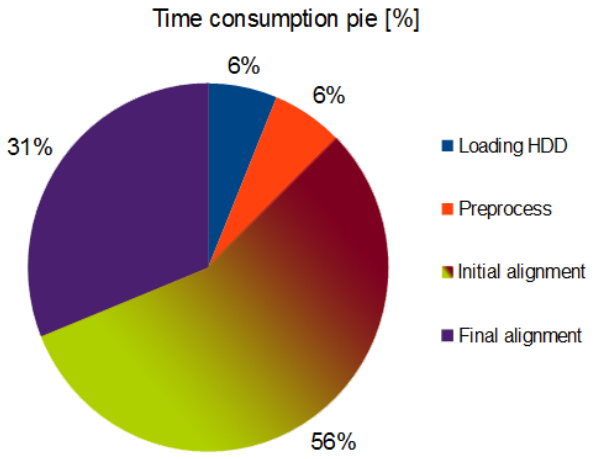

Figure 5.14: Pie chart with the time consumption percentage for the main steps of the algorithm

Chapter 6

Conclusions

In this work I have developed a robust and high configurable algorithm, able to find all the occurrences of a pattern in a point cloud dataset using Point Cloud Library framework.

Applying such algorithm on 3D scans of a bin, taking a single object as the model or pattern, the system can autonomously find the 6-Degrees-Of-Freedom poses of almost all the present objects. In this context the recognition is given by moving the model on the input point cloud, in a way that most of the points are aligned.

This process is guided by validation functions that give a value of the "pickability" of the objects, that measures the level of occlusion and the accuracy of the guess. An important parameter to evaluate such goodness is the distance between the actual object in the bin and its expected position, given by the alignment.

The following three parameters have always been taken as strict requirements for the entire work, due to the high efficiency and reliability that the industrial target needs. In the end of the project they have been all successfully achieved.

- The algorithm is **general purpose**: it could not be meant to recognize one particular object, failing with other ones. To achieve this feature, the initial guesses are obtained by correspondences between FPFH descriptors, that really care only about local geometry. Parameters are highly configurable, in order to fit any particular case.
- The algorithm is **reliable and robust**. Multiple iteration over the steps of the procedure have been implemented, so the probability to skip a good solution falls down. The user can configure how many iterations perform and on which parameters.

- It is **fast**, and gives an answer in 1 to 10 seconds. I have chosen fast keypoints and fast descriptors, implemented parallelism on CPU and given control to the user between speed and accuracy.

Considering the overall work, this system has been proved to be a successful solution to the given problem, and it can be directly applied in industrial contexts, with the modifications of the specific cases.

Bibliography

[1] Chris Harris, Mike Stephens. A combined corner and edge detector. Plessey Research Roke Minor, UK, 1988.

[2] R.B. Rusu, G. Bradski, R. Thibaux, J. Hsu. Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram. In Proceedings of International Conference on Intelligent Robots and Systems (IROS) Taipei, Taiwan, October 18-22 2010.

[3] R.B. Rusu, N. Blodow, M. Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, May 12-17 2009.

[4] R.B. Rusu, A. Holzbach, N. Blodow, M. Beetz. Fast Geometric Point Labeling using Conditional Random Fields. In Proceedings of the 22nd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, MO, USA, October 11-15 2009.

[5] R.B. Rusu, N. Blodow, Z.C. Marton, M. Beetz. Aligning Point Cloud Views using Persistent Feature Histograms. In Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26 2008.

[6] R.B. Rusu, Z.C. Marton, N. Blodow, M. Beetz. Learning Informative Point Classes for the Acquisition of Object Model Maps. In Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV), Hanoi, Vietnam, December 17-20 2008.

[7] Martin A. Fischler, Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, 1981

[8] Paul J. Besl. A Method for Registration of 3-D Shapes. IEEE Trans. on Pattern Analysis and Machine Intelligence, Los Alamitos, CA, USA, IEEE Computer Society, vol. 14, no 2, p. 239–256, 1992

[9] Stefano Tonello, Emanuele Menegatti. Bin picking robot: algoritmi di visione e framework software. Università di Padova, Italy, 2010.

[10] CAD-Model Recognition and 6 DOF Pose Estimation A. Aldoma, N. Blodow, D. Gossow, S. Gedikli, R.B. Rusu, M. Vincze and G. Bradski ICCV 2011, 3D Representation and Recognition (3dRR11) workshop Barcelona, Spain, (2011)

[11] Pearson Karl. Principal Component Analysis: On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine 2 p.559–572. London, UK, 1901

[12] M. Pham, O. Woodford, F. Perbet, A. Maki, B. Stenger, R. Cipolla: A New Distance for Scale-Invariant 3D Shape Recognition and Registration. Cambridge, UK, ICCV 2011

[13] http://opencv.org/

[14] http://pointclouds.org/

[15] http://www.ros.org/

[16] http://eigen.tuxfamily.org/

[17] http://www.openmp.org/

[18] http://www.willowgarage.com/pages/research