UNIVERISITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA INFORMATICA

TESI DI LAUREA MAGISTRALE

3D OBJECT RECOGNITION WITHOUT CAD MODELS FOR INDUSTRIAL ROBOT MANIPULATION

RELATORE: Dott. Stefano Ghidoni

CORRELATORE: Ing. Roberto Polesel

LAUREANDO: Luca Benvegnù

09 Ottobre 2017
ANNO ACCADEMICO 2016/2017

Abstract

In this work we present a new algorithm for 3D object recognition. The goal is to identify the correct position and orientation of complex objects without using a CAD model, input as main current systems. The approach we follow performs feature matching. The characteristics extracted belong only by shape information to achieve a system independent to brightness, colour or texture. Designing opportune settable parameters, we allow recognition also in presence of small deformations.

One of its industrial applications is bin-picking that consists in identification and extraction of article disposed in a box. In this way products can be available for further processing. We tested our algorithm in both ideal and real environments with several object types, analysing quality and precision. We used two different 3D cameras based on structured light: Photoneo PhoXi and Microsoft Kinect v1.

Sommario

In questo progetto si presenta un nuovo algoritmo per il riconoscimento di oggetti 3D. Lo scopo è identificare la corretta posizione e orientazione di oggetti complessi senza usare un modello CAD, input dei principali sistemi attuali. L'approccio seguito è il feature matching. Le caratteristiche estratte riguardano solamente la forma dell'oggetto per realizzare un sistema indipendente dalla luminosità, colore o trama. Progettando degli opportuni parametri modificabili, si permette il riconoscimento anche in presenza di piccole deformazioni.

Una delle applicazioni industriali è il bin-picking che consiste nell'individuazione ed estrazione di articoli disposti all'interno di un cassone. In questo modo i prodotti possono essere resi disponibili per successive lavorazioni. Si è testato l'algoritmo sia in ambiente ideale che reale con diverse tipologie di oggetti analizzando qualità e precisione. Si sono usate due telecamere 3D basate su luce strutturata: Photoneo PhoXi e Microsoft Kinect v1.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the world of automation and robotics there are a lot of applications that cannot be achieved without 3D object recognition. This task allows to identify a target in the scene starting by a model: in this way a robot can acquire informations about the environment or bring object to process or transport and provide it for further manufacturings.

A particular application that is linked with object recognition is bin-picking: targets are scattered in a box as an assembly-chain output and the aim is to pick them out to a predetermined position in the space. In a more complex version, targets are positioned randomly so the problem is called random-bin-picking. This is an open challenge for robotics: even though some companies and researchers have just developed some working systems, there are a lot of aspects that are still unresolved. In detail these software need a CAD model of the target as input and this imposes to create it. In addition the presence of deformations and the absence of a descriptor model, for example in natural products (ex. apples, which no one is perfectly equal to the others), are still been studying. Some systems also take advantage of singular characteristics and are designed to recognize only a precise object type.

In this work we address bin-picking problem to build basis for a general recognition algorithm not sensible to small deformations. The aim is to achieve a system for different object types using only shape information. This allows the robot to work in the dark or in condition of variable brightness and object colour or texture. Another important aspect of the proposed method is that we start from a scan and not from a CAD model. The advantages are linked to fast production of subject description and system versatility: with this work we want to manipulate also objects that have not a model to be set as input.

All the project is designed in cooperation with Euclid Labs. This company, with offices in Nervesa della Battaglia (TV), designs and develops hi-tech solutions for robotics and industrial automation.

1.1 Analysis of task issues

There are several problems linked to recognition aimed to bin-picking, the main are:

- detection of the specific object among others: in the scene there are several elements that can be part of the environment, objects equal or not to the model and so on. A first difficulty is to choice what of these are real targets and discard the subjects that are not;
- computation of its position and orientation: in bin-picking task this is a very crucial point. We need the position to know where is the target and the orientation to understand the pose, perform collision avoidance in path planning and place it correctly;
- identification of the grip point: to grasp an object is essential to find a point where this operation can be executed more easily than in other parts. Quite always this point is established a priori but we have to compute its position in the scene to move robot toward it;
- computation of predicted position accuracy: in particular the identification of grip points, object position and orientation need great precision. In several cases objects are very small or grasping is performed in a specific part to exploit shape characteristics. Knowing recognition reliability, we can pick the object without a doubt or decide to re-do the procedure from another scan to be more precise;
- sensibility to clutters: targets can be partially hidden for lacks of details in scan or undercuts. The object should be recognized as well;
- time execution: all the computations must be done as fast as possible not to slow down the entire production-chain. The efficiency is crucial.

In addition to bin-picking problems, using a scan as model input, we have to manage noise that characterizes data. CAD models are technical drawings that represent precisely all object sides and are not affected by errors. Point cloud noise can be caused by the unlucky combination of 3D camera technology and object materials or by low sensor precision. In order to propose this work to an industrial context it is important to check system robustness to noise: the knowledge of usability bounds is crucial not to have problems on the field.

Also repeatability is a basic property of this system: with same input we must return the same output. These to avoid some uncertainties on results.

This report is organized as follows: in Chapter 2 we will present state of art systems and approaches to object recognition. These are not all the studies that we have analysed to design our algorithm but are the most significant and the ones that give us the insights about the way to forward.

In Chapter 3 we will present the procedure with its algorithms, issues found and solutions adopted.

We will continue in Chapter 4 with implementation details such as libraries, tools, 3D camera and measuring instruments used in system realization and evaluation.

One of the most important part is Chapter 5 where we will discuss our qualitative and quantitative tests that remark pro and cons of this work. We will focus on precision analysis in different contexts and difficulties.

In the end in Chapter 6 we will report conclusions and future improvements.

Chapter 2

State of art solutions

State of art systems adopts mainly four different approaches. They are based on: the creation of a mathematical object model, aspect graphs, machine learning and feature matching. Every solution has some advantages and disadvantages that we are going to explain in the following sections.

2.1 Mathematical model creation

The first approach that we have studied in literature is based on the creation of a mathematical model of the object. This solution is adopted in particular for the reconstruction starting by noisy data but it can be exploited to our aim with proper modifications. Indeed, they use a lot of parameters that can excessively slow down recognition, that is a higher-level process.

An example of these procedures is generalized spline models [1] that can deform locally subject to generic continuity constraints. This appears to be well suited to shape reconstruction but by contrast it needs a drastic information reduction and shape abstraction in order to support efficient matching in object databases of manageable size. For this application cylinder, sphere, pyramid or prism models can be more easily used since they are compactly characterized by a small set of parameters.

In this scenario D. Terzopoulos and D. Metaxas proposed a new modeling, suitable also to recognition of deformable objects called deformable superquadratics [2, 3]. The goal is to create a new family of models that combines membrane splines with parametrized ellipsoids. In this way they obtained great expressive power: splines are free-form and locally deformable, thanks to their shape control variables that provide many local degrees of freedom, and superquadric ellipsoids are globally deformable. The authors also controlled deformities applying physical forces to obtain a sort of object state prediction. Therefore

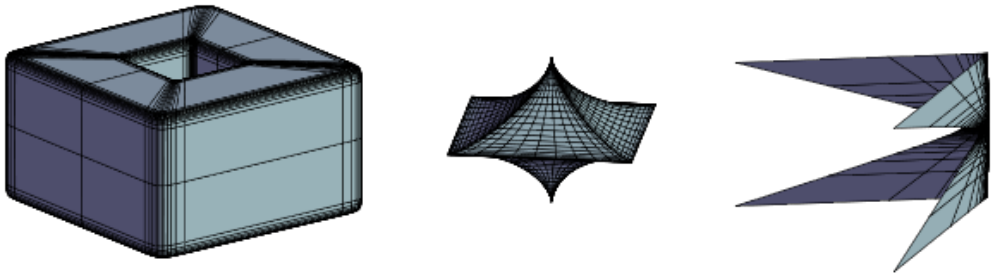


Figure 2.1: Examples of super-quadratics

they must also characterize the model with its physical properties such as flexibility, malleability, compressibility and stiffness as well as its mass and energy. The final result is composed by several degrees of freedom: a translation vector, a quaternion for the rotation, a scale, three radial aspects and two squareness parameters.

It is a powerful instrument but its complexity is a drawback. We have just discussed the importance not to use too parameters in the modeling. Not considering number of degrees of freedom, the prediction of environmental forces needs to a simulation and it is not adaptable to context changes. Furthermore this model solves deformation problem but not the recognition of natural product or other not modelable objects where the deformities are not predictable.

2.2 Aspects graphs

Another solution explored in literature is based on construction of the aspect graph of the object starting by a 3D model [4]. In this data structure nodes are different object aspects or views and edges represent physical nearness. The recognitions are performed executing a sort of feature matching between target and aspects to find the best match. The data are organized in a graph to cleverly research the right view in the database.

The issues linked to this procedure are: how many object aspects we have to acquire to realize an accurate system and how this data can be obtained. The risk is to consume too much memory and build a very slow system.

2.3 Machine learning

One of the most important differences between rigid and deformable objects is that the latter can change their physical state, their grasping point and their pose. Using machine learning we could train a model to recognize them using an off-line simulation.

This is the basic idea of Y. Li C. F. Chen and P. K. Allen's work to estimate category and pose from a set of depth images [5]. It is the first project in the deformable object recognition world that focuses not only on labeling but also on position and orientation identification. Indeed P. F. Felzenszwalb R. B. Girshick D. McAllester and D. Ramanan described an algorithm based on mixtures of multi-scale deformable part models trained using a discriminative procedure [6]; M. Pedersoli A. Vedaldi J. Gonzàlez and X. Roca improved the part-based model by doing a multiple-resolution hierarchical layer structure which increased the detection accuracy [7].

Y. Li C. F. Chen and P. K. Allen's new approach proposes to crate simulated models in order to increase time efficiency and accuracy. In this way data are not affected by errors or noise, therefore there is no need to any preprocessing to clean the input: computations can be executed quickly and in ideal conditions. Another advantage is that this solution is cheaper than to acquire with 3D cameras a lot of object scans: to train a Support Vector Machine numerous frames are necessary. In a bin-picking context, robot could see whatever object side therefore data must be caught from different points of view. The authors report that they simulated 90 cameras positioned on a geodesic dome.

Changing object position also the grasping point changes. To solve this problem researchers establish manually 20-50 gripping points.

After this processing there is feature extraction phase: authors used *densSIFT* [8, 9] (a variant of SIFT algorithm [10]) to identify and describe keypoints not to have sensibility to rotation and scale changes. On these features two-level SVM classifier is trained: one in order to recognize the object category and one to find the grasping point.

Referring also to other works, such as the one of H. Tehrani Niknejad, A. Takeuchi, S. Mita and D. McAllester [11] or the second of B. Kim, S. Xu and S. Savarese [12], we can create a general scheme about this approach:

1. acquire training set to construct a complete model of the object;

2. extract from these data the features that describe the target;

3. train SVM or other machine learning models to obtain a good classifier given an objective function computed from the features;

4. analyse the scene with the classifier to recognize the object and identify its location in 3D space.

The results obtained with this process are fascinating: they can detect targets that are in several positions and with a significant degree of deformity caused by movements or extreme rotations. The first method explained was tested on clothes that are a very difficult target for the infinite different positions that can take and for deformation entities reached. The authors reported that the accuracy of recognition is always up to 70% and in case of shorts they are correctly detected 9 times over 10.

But this method has some disadvantages: to realize a good training, it needs a large quantity of data acquired from different points of view in order to represent every object side. This can be expensive considering execution time and achievement costs. To give a number: in [11] the researchers collected 553 images with 854 vehicles. This can be done once for all if the goal is well defined and it doesn't change over time. But the approach is not applicable to a system that could be used in the industrial world where every company has different targets to recognize. A second drawback is that this technique firstly analyses data in 2D space to obtain a less accurate but faster recognition, and then 3D informations to improve precision. For this reason features are also HOG gradients histograms or SIFT output. These are invariant to image scaling and rotation, and partially to changes in illumination and 3D camera viewpoint. But they are difficultly applicable in a 3D environment like the our because they exploit other information than only shape[1].

2.4 Feature matching

Another approach that several researchers' works describe is feature matching. The literature is full of examples but the following ones capture the attention for some interesting aspects.

The first that we have analysed is the A. K. Jain Yu Zhong and S. Lakshmanan's work [13]. They approached the problem of object localization and identification like a process of matching a deformable template to the object boundary in an input image. The prior shape information is specified as a sketch or binary template. This prototype template is not parametrized, but it contains edges information. Deformed templates are obtained by applying parametric transforms to the prototype, and the variability in template shape is achieved by imposing a probability distribution on admissible mappings. Among all such

[1]Point Cloud library has just realized an implementation of SIFT in 3D but it needs of a intensity field that isn't always available.

transformations, the one that minimizes a Bayesian objective function is selected. The objective function consists of two terms: the first plays the role of a Bayesian data likelihood (it is a potential energy that links edge positions and gradient directions in the input image to the object boundary specified by the deformed template) and the second corresponds to a Bayesian prior that penalizes large deviations from the prototype. Deformable template minimizes the objective function by iteratively updating transformation parameters to alter template shape so that the best match with image edges is obtained.

This was an interesting work since it reaches high accuracy having only edge data. Starting from a sketch or little information as a binary image they can recognize hands, saxophones, towers... on real images. They only concerned 2D cases but some ideas can be transported to 3D space.

A. C. Berg T. L. Berg and J. Malik focused on the correspondence problem: how can we establish algorithmically that two points, one of the model and one of the target, are the same? [14] The particularity of their work is that they didn't start from specific keypoints to realize the matches, but from some points sampled randomly from edges. This is interesting because they skipped the step of a deterministic feature extraction. The description of selected points can be done in several ways: SIFT [10], Shape context [15] or Geometric Blur [16]. They also observed that if i and j are points on the model corresponding to i' and j' respectively, the vector from i to j, r_{ij} should be consistent with the vector from i' to j', $r_{i'j'}$. If the transformation from one shape to another is a translation accompanied by pure scaling, these vectors must be scalar multiples. If the transformation is a pure Euclidean motion, lengths must be preserved. The process ends with the smoothness of the transformation from one shape to the other. This enables to interpolate the transformation to the entire shape, given just the knowledge of the correspondences for a subset of the sample points. In order to characterize the transformations they used regularized thin plate splines.

Another work adds new hierarchical agglomerative clustering method to feature matching to find regions of similar transformations between model and target. M. Cho, Jungmin Lee and K. M. Lee in [17] based their approach on two insights:

- Bottom-up aggregation strategy: they started from confident correspondences and progressively merge them with reliable neighbours to make sure that inliers can be effectively collected in spite of enormous distracting outliers. For example, seed-based exploration methods [18, 19, 20] prove that object recognition performance can be boosted by such a bottom-up aggregation with iterative match-propagation;

- Connectedness between parts: for deformable objects, feature correspon-

dences do not form global compactness in their pairwise geometric similarity owing to deformation, but deformed parts are locally connected by some mediating parts. Thus, a connectedness criterion should be considered for clustering the feature correspondences on deformable objects.

The procedure that they realized uses affine covariant region detectors [21, 22] for feature extraction and SIFT [10] for descriptions as many other works in literature. The matching phase started computing the description differences from target and model features identifying the best matches. Then for each match is built a cluster so that every cluster contains a single match. With an iterative process, at each step two clusters that are the most similar are merged in a single one. The dissimilarity function has two weighted components: photometric and geometric. The first is defined by the Euclidean distance between corresponding SIFT descriptors of model and target feature; the second is the Euclidean distance from feature positions applying an homograph transformation. Setting the stopping condition with a maximum similarity value, they obtained a dendrogram where outliers are never selected for the merging phase. At the end, convex hull or clusters with size lower than a pre-selected threshold are deleted [2].

Their approach provides reliable feature correspondence, object-level multiclass clustering and outlier elimination in an integrated way. Its control parameters are simple and intuitive and it does not require a global energy formulation, strong global constraints, nor a specified number of clusters. Moreover, it is very robust to distracting outliers arising from clutter in real-world images.

From these examples, feature matching approach schema can be written as follows:

1. feature extraction: both model and target are analysed to find some distinctive characteristics such as particular shapes, texture, edges, keypoints and so on. These are useful to summarize object composition;

2. feature description: the points found in the previous step must to be described unambiguously so that the single feature cannot be confused with another;

3. feature matching: here the features that have the most similar description between model and target are matched. This phase recognizes same object parts in the two data;

[2] For cluster size we mean the number of its elements. For convex hull dimension we indicate a geometric measure of the space occupied

4. post process of matches: in this step the matches are analysed to discard errors.

Most of the studied papers concerns about 2D procedures but the ideas can be translated into 3D space with only a few changes. Feature extraction methods exposed works fine on 2D environment but are too expensive in execution time with three-dimensional informations. Our work pretends to not use colour or brightness data so these algorithms like SIFT cannot be used. However the approach is general and not strictly dependent to the specific object type. The model is not a complex representation of the target and does not must be preprocessed with long execution time. The system resulted is very versatile and in agreement with our goals.

Chapter 3

Description of 3D object recognition system

In this section we are going to present our system and the main designed algorithms. The basic idea is founded on a feature matching approach that gives more versatility and less sensibility to brightness or colour changes. Indeed, we use only shape information to be independent of environment and context variations. Using some customizable parameters we can control the difference between descriptions in order to consider two features as correspondent. The result is a system insensible also to little target deformations.

As we reported in Figure 3.1 the algorithm can be divided into 4 steps:

1. scan preprocessing: we prepare the point cloud for the following processes. In this way we can elaborate well structured and less noisy data. The importance of this phase will be clear in following sections;

2. keypoint extraction: to build a system that is less sensible to noise as pos-

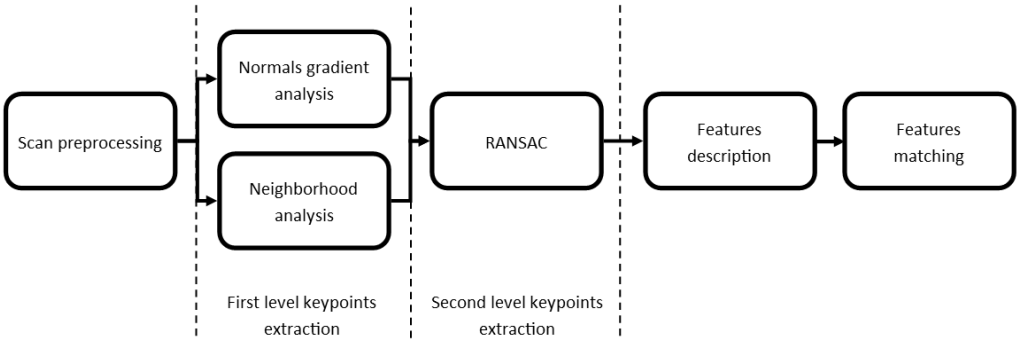


Figure 3.1: Algorithm general steps

sible, keypoint extraction is divided into two levels that are run in cascade. The first part is composed by two different algorithms that we call *Normals gradient analysis* and *Neighbourhood analysis*. Their goal is to focus on two specific shape characteristics: curvature peaks and edges. The latter compacts the information into a few points to reduce matches combinations number;

3. feature description: in this step we describe the characteristic points selected in the previous step. This description is based on local analysis in order to be independent from the point of view. Two features that describe the same object characteristic in several scans must have quite identical descriptions;

4. feature matching: finally we compare the points descriptions to find the best correspondences. The output of the whole algorithm will be the rotation matrix and the translation vector that transport the model to the target.

Points 2 and 3 are executed in the same way for both model and target. There are some differences in the implementation of scan preprocessing caused by the fact that the model is scanned alone, it is just one and it is acquired knowing the scene and its position; but targets are numerous in the same point cloud and we cannot have any other information.

In the following sections we are going to explain in detail each step and used algorithms. To be clearer on the explanation we will start from keypoints extraction skipping the first phase. Indeed, the role of preprocessing could be really understood only by knowing the other phases.

3.1 Keypoint extraction

First level keypoints

The goal of this phase is to sample the scan and return as output only the points that bring some fundamental informations about object shape. In our first approach we identified these characteristics with curvature peaks. The most important advantage is that we can extract features also from curved objects that have not significant edges. We remark that also borders are identified as a discontinuity and will be considered as relevant points. To implement this we design an algorithm that analyses surface normals gradient.

Algorithm 1 Normal gradient analysis

```
1:  procedure EXTRACT_KEYPOINTS(point_cloud, s, n)
2:      keypoints
3:      for each point in point_cloud do
4:          peak_index ← findPeak(point, n, 0, s, 0)
5:          if (peak_index = −1) then
6:              keypoints ← peak_index
7:      return keypoints
8:
9:  procedure FIND_PEAK(point, n, α, s, step)
10:     neighbourhood ← getKNearestPoint(n)
11:     pivot_normal ← getNormal(point)
12:     max_angle ← 0
13:     max_angle_point ← −1
14:     for each neighbour in neighbourhood do
15:         neighbour_normal ← getNormal(neighbour)
16:         temp_angle ← computeRotationAngle(pivot_normal, neighbour_normal)
17:         if (|temp_angle| > max_angle) then
18:             max_angle ← temp_angle
19:             max_angle_point ← neighbour
20:     if (step < 1 ∧ max_angle < s) then
21:         return -1
22:     if (max_angle > α) then
23:         step ← step + 1
24:         return findPeak(max_angle_point, n, max_angle, s, step)
25:     else
26:         return point
```

Normals gradient analysis

Pseudo-code is reported in Algorithm 1. The input is composed by the point cloud and two parameters that describe the degree of sensibility (s) and the neighbourhood size in terms of points number (n). The general procedure is quite simple: we focus on each point and, moving on the normal gradient of its area, we find the position where curvature reaches a peak. This is what does the function $findPeak$ in broad terms at line 4. The value -1 is a marker to identify when the point area is flat and there are no curvature peaks. So only when the function result is not equal to -1 it is considered valid and it is added to the final container.

The most important part is $findPeak$ function. Firstly we have to compute

the neighbourhood composed by n points. Then for each neighbour we compare its normal direction with the one of pivot [1] finding an angle that describes the variation. We save the neighbour that produces the maximum angle and we repeat the procedure focusing on this point until one of the following stopping conditions is verified:

- at the first step, before using any recursive calls, the maximum angle found is lower than the sensibility s: to manage scan noise we consider a tolerance threshold for flat regions;
- the direction change registered in the previous step is greater than the current: in this case we find a peak.

Neighbourhood analysis

The drawback of *Normals gradient analysis* is that with subjects composed by constant curved regions or flat surfaces we cannot extract much information. Furthermore, in case of undercuts, normals do not show any direction change: this is a problem because we cannot distinguish real edges and blind spots, and the algorithm does not return any point. Indeed undercuts are empty regions of the scan that correspond to parts of the object that are not visible from the camera viewpoint for the presence of clutters caused by other object sides.

To avoid this problem we consider all these cases interesting and useful for the following computations. In agreement with this decision we designed a second algorithm for the first level keypoints extraction called *Neighbourhood analysis*.

In Algorithm 2 we report the pseudo-code. The input is composed by the point cloud and two parameters that represent the radius search for neighbourhood (Δ_r) and the minimum distance between pivot and neighbourhood centre to establish if it belongs to an edge or not (Δ_p). Initially, for each point we search the other points that are close to it [2] (line 4). Then we compute the centroid position of this group (line 5): if this is distant more than Δ_p, the point in exam is labelled as a keypoint and it is added in the output list (lines 6, 7, 8).

This algorithm works thanks to the observation that when neighbourhood centroid is far from pivot, it means that the majority of its neighbours is placed in a precise direction and is not completely around it. This is the case that characterizes an edge.

[1] We call pivot the focused point

[2] the neighbourhood is defined as the set of points that has a distance lower than Δ_r

Algorithm 2 Neighbourhood analysis

```
1: procedure EXTRACT_KEYPOINTS(point_cloud, Δ_p, Δ_r)
2:     keypoints
3:     for each point in point_cloud do
4:         neighbourhood ← getNeighbourhood(point_cloud, point, Δ_r)
5:         centroid ← computeCentroid(neighbourhood)
6:         distance ← computeDistance(points, centroid)
7:         if (distance ≥ Δ_p) then
8:             keypoints ← points
9:     return keypoints
```

Algorithms comparison

Comparing this solution with the previous one we can observe that:

- the two methods are complementary: while the former fails when objects are regular and sharp-cornered for the presence of undercuts, the latter selects too few points when surfaces are curved and there are no edges;
- neighbourhood analysis does not require normals computation: this is an advantage for execution time;
- both the implementations need a fast algorithm to compute the neighbourhood: KdTree representation of the point cloud can help [23, 24];
- in this work we design a system that uses only one algorithm but not both. A future improvement could be the cooperation of these methods to have more informations about the model.

In Figure 3.2 we report the results obtained applying these two algorithms to the scan in the left: the right-top output derives from *Normals gradient analysis* and the right-bottom one from *Neighbourhood analysis*. It might seem that the second result is the best for definition and precision but this is not true: both the approaches show different object aspects. *Normal gradient analysis* gives more importance to the curvature changes. This can be seen in the bottom part of the pump component where borders start the bulge to reach the central hole. Here a lot of points, that the other algorithm filters out, are maintained. We notice also that some edges are not well defined caused by undercuts. In these regions there is no information about normals direction changes therefore they are considered regular. *Neighbourhood analysis* instead focuses on edges so the output remarks borders.

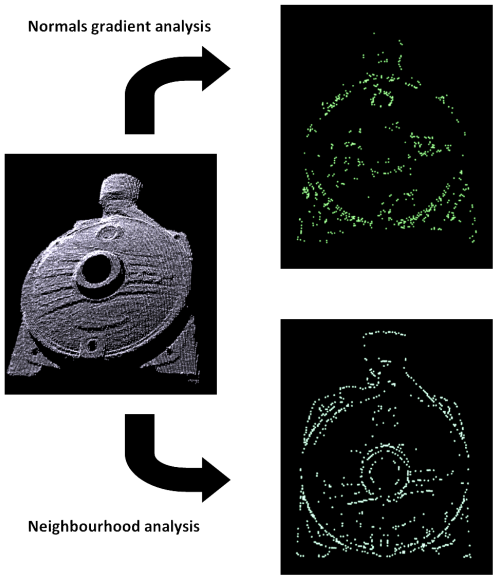


Figure 3.2: Results of both first keypoints extractors

Second level keypoints

With the previous step we have obtained a scan sample with a reduction of around 4-5% points [3] but it is not enough. To realize an efficient feature matching, keypoints must be fewer in order to produce less combinations and have more specific descriptions. This means that a second sampling step is necessary. The idea is to collapse group of first level keypoints into only one point and preserve all the necessary information about the group in its description. Clustering is done in agreement with the previous approaches: each group is a particular line or circle extracted from object shape. We decide to search for these 2D features for the ease of their mathematical model that implies fast research time.

To realize this procedure we use Random Sample Consensus (RANSAC) [25]. Algorithm 3 reports the pseudo-code. Given a model that requires a minimum of n data points to instantiate its parameters and a set of data points P such that the number of points in P is greater than n ($|P| > n$), it randomly selects a subset S_1 of n data points from P and instantiates the model. It uses the instantiated model M_1 to determine the subset S_1' of points in P that are,

[3] This data is acquired in the same context of Figure 3.2 applying both algorithms. The exact result is a sample from 16 816 points to 678 (first algorithm) and to 802 points (second algorithm). This means a reduction respectively of 4.03% and 4.76% that reaches an average of 4.39%. It is important to remark that this is only a general estimation because these values change a lot varying parameters. The aim is to give a quantitative idea of the algorithms effects on real data.

within some error tolerance, inliers of M_1. The set S'_1 is called *consensus set* of S_1. If $|S'_1|$ is greater than some threshold t, which is a function of the estimate of errors number in P, it uses S'_1 to compute (possibly using least squares) a new model M'_i. If $|S'_i|$ is lower than t, it randomly selects a new subset S_2 and repeats the above process. If, after some fixed number of trials, no consensus set with t or more members has been found, it either solves the model with the largest consensus set found, or terminates in failure.

For example, given the task of fitting an arc of a circle to a set of two-dimensional points, the RANSAC approach would select a set of three points (since three points are required to determine a circle), compute the centre and radius of the implied circle, and count the number of points that are close enough to that circle to suggest their compatibility with it (their deviations are small enough to be measurement errors). If there are enough compatible points, RANSAC would employ a smoothing technique such as least squares, to compute an improved estimation for the parameters of the circle, now that a set of mutually consistent points has been identified.

At the end, the points that are not included into any 2D features are deleted. Applying this algorithm to find lines and circles and setting properly the threshold we can obtain a good feature extraction that summarizes the object with at least 60-80 points.

Algorithm 3 Random Sample Consensus

1: **procedure** FIND_MODEL($P, model, n, t, max\_trials\_num$)
2: $model\_found \leftarrow false$
3: M'_1
4: $trials\_num \leftarrow 0$
5: **while** (!$model\_found$) **do**
6: $S_1 \leftarrow randomSelection(P, n)$
7: $M_1 \leftarrow instantiateModel(model, S_1)$
8: $S'_1 \leftarrow exctractPoints(P, M_1)$
9: **if** ($|S'_1| \geq t$) **then**
10: $M'_1 \leftarrow computeConsensusModel(model, S'_1)$
11: $model\_found \leftarrow true$
12: **else if** ($trials\_num > max\_trials\_num$) **then**
13: return $failure$
14: $trials\_num \leftarrow trials\_num + 1$
15: **return** M'_1

3.2 Feature description

Running the previous algorithms in cascade, the scan is filtered and a group of characteristic points called keypoints are returned. The goal of the current step is to describe unequivocally the extracted features. This means that lines and circles are processed differently.

Lines description

Once we find a line with RANSAC we collapse all the points that compose it into a single one that we decide to be its centre. Note that for some clutters or interruptions generated by the previous filters, this point there could not be in the point cloud or could not exist in the real scene. This is not an issue: we delete all line points and replace them with the centre.

We compose the feature description with several fields:

- feature type: we save the information that the point represents a line and not a circle;
- line orientation: this data can be directly acquired from RANSAC model coefficients;
- centre position: to compute this information we consider a point selected randomly in the line. Then, we find the two farthest points, one to the left and one to the right. These are the two line extremities therefore in order to compute the centre we have to average their positions;
- line length: once we computed line extremities, the Euclidean distance between these points is its length.

We use these properties inspired by vector mathematical characterization (module, direction and way).

Circle description

The same process that we designed for lines is implemented for circles. Also, in this case the point that we use to represent the cluster is its centre. The fields that we use to describe circles are:

- feature type: we save the information that the point represents a circle and not a line;
- normal to the surface;

- centre position;
- radius length;

All of these characteristics can be easily retrieved from RANSAC model coefficients. We decide to use them as feature description because they are the circle mathematical model parameters.

3.3 Scan preprocessing

In order of execution this phase is the first. Here the goal is to properly prepare the input data to following processes. This is the unique step that is different between model and target. When we are analysing models, the only thing to do is to remove the floor, planes and everything in the scene that is not the subject. This can be done once manually using some software of 3D computer graphics.

When we are analysing targets there are several problems that are generated by the lack of scene knowledge and the probable presence of more than one object to recognize. In particular the presence of several elements (targets and other in the environment) causes problems to RANSAC. Indeed when it finds a line or a circle in the pointset that contains more points than a threshold, it considers as inliers all the points in the line direction or in the circumference, without checking that they belong to the same object. The major negative effects are seen in lines search and are amplified during the description phase: the length will be distorted by the presence of other inliers and its value will never find a correspondence in the model. This event happens quite always even though it might seem to be sporadic.

For this reason we apply a segmentation at the starting point to produce a point cloud for every element. This is presented as a centre-based clustering problem where final clusters are the objects: this approach is quite reasonable assuming that, when points are close to each other, they are assembling a subject. The three principal centre-based clustering algorithms are Farthest-first traversal, K-means and K-median [26]. Their goal is to find the given number of centres minimizing a specific objective function that characterizes them. The final partition is always obtained associating each point to the cluster of the nearest centre.

Farthest first traversal

The aim of this algorithm is to solve K-centre approach. The objective function that characterizes this method is:

$$\Phi_{kcenter}(C) = \max_{i=1}^{k} \max_{a \in C_i} d(a, c_i). \tag{3.1}$$

The procedure goal is to minimize the maximum distance, computed for all the clusters, between a cluster element and its centre. In order to implement this, we chose as centres the points that are the farthest to each other. The pseudo-code is shown in Algorithm 4. The function at line 4 finds the farthest point from the other centres set. We remember that the distance between a point and a set is the minimum one between the point and an element that belongs to the set. The starting point is selected randomly.

Algorithm 4 Farthest First Traversal

```
1: procedure FIND_CENTRES(P, k)
2:     S ← 0
3:     for i = 0 to k do
4:         c_i ← findFarthestPoint(S)
5:         S ← S ∪ c_i
6:     return S
```

K-means

In this case the objective function is:

$$\Phi_{kmeans}(C) = \sum_{i=1}^{k} \sum_{a \in C_i} (d(a, c_i))^2 \tag{3.2}$$

so the algorithm, called Lloyd's [27, 28], must minimize the sum for all clusters of the sum of square distance between each point and its cluster centres. The procedure is more complex than the previous and it is well explained by the pseudo-code in Algorithm 5.

It is based on the observation, that can be easily proved, that centroid is the point that minimizes the sum of square distance of all the cluster elements. We remember that the centroid c of a pointset P can be computed with the follow equation:

$$c(P) = \frac{1}{|P|} \sum_{x \in P} x. \tag{3.3}$$

The algorithm starts from a random sample of k centres (where k is given clusters number), partitions the pointset with the general procedure explained in the general section (3.3), computes objective function value and the centroids.

Algorithm 5 Lloyd's algorithm

```
1:  procedure FIND_CENTRES(P, k)
2:      S ← getRandomSet(P, k)
3:      Φ ← ∞
4:      stopping_condition ← false
5:      while stopping_condition do
6:          (C_1, C_2, ..., C_k; S) ← partition(P, S)
7:          for i = 1 to k do
8:              c'_i ← getCentroid(C_i)
9:          C ← (C_1, C_2, ..., C_k; c'_1, c'_2, ..., c'_k)
10:         if (Φ_kmeans(C) < Φ) then
11:             Φ_kmeans(C) ← Φ
12:             S ← c'_1, c'_2, ..., c'_k
13:         else
14:             stopping_condition ← true
15:     return C
```

Then, it starts an iterative phase where it computes once again the partition using centroids as new centres and objective function value. It compares this value with the previous one and if it is lower it continues the loop. Otherwise if there is no improvement, the stopping condition is verified and the algorithm returns the last centres.

In the literature we can find several improved versions that obtain better results like K-means++ that does not start from a random sequence of centres. In this last implementation the selection is driven by a distribution function that privileges the farthest points [29, 28].

K-median

The objective function of this approach is the following:

$$\Phi_{kmedian}(C) = \sum_{i=1}^{k} \sum_{a \in C_i} d(a, c_i). \tag{3.4}$$

This problem is quite similar to k-means but it uses distances without squaring them. The points that minimize the function are called medoids and their meaning is almost equal to centroids one with a little difference: centroids can not belong to the pointset but the medoids do. The algorithm that solves this approach is Partitioning Around Medoids and its pseudo-code is reported in Algorithm 6. It starts with a random sequence of centres selected from the pointset

Algorithm 6 Partitioning Around Medoids

```
1:  procedure FIND_CENTRES(P, k)
2:      S ← getRandomSet(P, k)
3:      C ← partition(P, S)
4:      stopping_condition ← false
5:      while stopping_condition do
6:          stopping_condition ← true
7:          for each p ∈ P − S do
8:              for each c ∈ S do
9:                  S' ← (S − c) ∪ p
10:                 C' ← partition(P, S')
11:                 if (Φ_kmedian(C') < Φ_kmedian(C)) then
12:                     stopping_condition ← false
13:                     C ← C'
14:                     exit both for-each loop
15:     return C
```

and computes the partitioning. In the next steps it implements a sort of local search: it tries to substitute each centre with one other point in the pointset. When it finds a replacement that produce a lower objective function value, it save this and restarts searching with new centres sequence. The procedure stops when any improvement is found.

This algorithm is quite inefficient and it is very slow with large pointsets. So there is another algorithm called K-medoids that minimize the same objective function of PAM in the same way as Lloyd's algorithm, with the only difference that it computes medoids and not centroids.

Comparison of clustering algorithms and final implementation

These three algorithms have pros and cons. Farthest-First traversal is very simple to implement and fast with few centres. It is also very sensible to outliers: in presence of isolated points frequently it chooses these as centres for their distance from the others. A characteristic of the partitioning obtained with this procedure is to have clusters with a greater size and some others that contain few points. But if in the pointset there is not noise the result is very good. Another drawback of this technique is that increasing the number of centres, time execution increases exponentially.

Both the issues are almost solved by the second algorithm. Considering all the distances as a sum in the objective function, the outliers contribution is averaged by the other elements. In addition, K-means is less sensible as the centres

number increases, although, when there are less clusters, Farthest First Traversal is faster.

K-median tries to improve independence to noise deleting square elevation. This bestows less importance to large distances and their contribute is mainly averaged by the others. Despite this improvement, the execution is slower because, in k-medoids, to compute a medoid it is necessary to find centroid first, and then to discover its nearest point in the pointset.

For this reason we decide to use k-means approach implementing Lloyd's algorithm with a minor change. At the beginning, we avoid to use a random sequence of points as centres because the result depends on it: each particular initial set produces a different clustering. In addition, starting every time with a different configuration, we introduce a degree of randomness that can have effects on the reproducibility of tests. Observing that usually objects are not so numerous we find the initial sequence with Farthest First Traversal. This choice seems to slow down the execution because it introduces more operations but in practice our test proves that not only it increases algorithm quality performances but also it speeds up the execution and gives more stability.

In Figure 3.3 we report our test results. Here we apply five times k-means clustering in 2 different scenes (one of some cups and one of some pump components; the scans can be seen in Section 5.3) with and without Farthest First Traversal (FFT) in the first step. We obtain that using a random configuration at the beginning, the algorithm is almost always slower. Analysing the average and variance in Table 3.1 we notice that the use of FFT gives more stability also in execution time. The reasons of this improvement are that we have fewer objects so k-centre approach does not show its inefficiency and k-means starts with a good centres sequence, so it only needs a few iterations to verify stopping conditions.

After this analysis we decided to use this k-means new version to perform clustering.

We add also a new procedure to avoid cases in which some output clusters are composed by only one or two points. This happens when a scan is unclear and for the presence of noise some points are too far from the others: in this situation the best way to optimize k-means objective function is to consider smaller clusters that contain only these outliers. Our modification consists of removing elements from the pointset if they compose a cluster with less than 3 points and re-executing clustering from the beginning. This solution is time-consuming but extremely important: a bad clustering can cause wrong recognitions or failures.

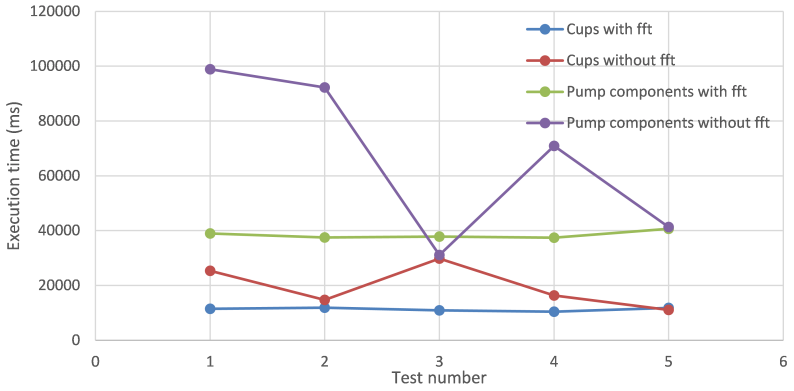


Figure 3.3: Graphic for k-means clustering execution time with and without Farthest First Traversal as first step

Test	Average (ms)	pointset (ms^2)
Cups with fft	11303	301187,8
Cups without fft	19455	48711249
Pump components with fft	38468	1476205
Pump components without fft	66901	$7{,}23 \times 10^8$

Table 3.1: Results of k-means clustering with and without Farthest First Traversal

Algorithm to find object count

An issue of these clustering algorithms is that they require the number of final clusters as input. This means that we have to know the objects number in the scene and this a-priori knowledge is a strong system limitation. When we are analysing a box to perform bin picking is quite impossible to count the products that are seen by the scanner. In addition this knowledge makes less automatic our algorithm and necessities of human interactions. So we implemented a new procedure to count the number of parts to be analysed independently.

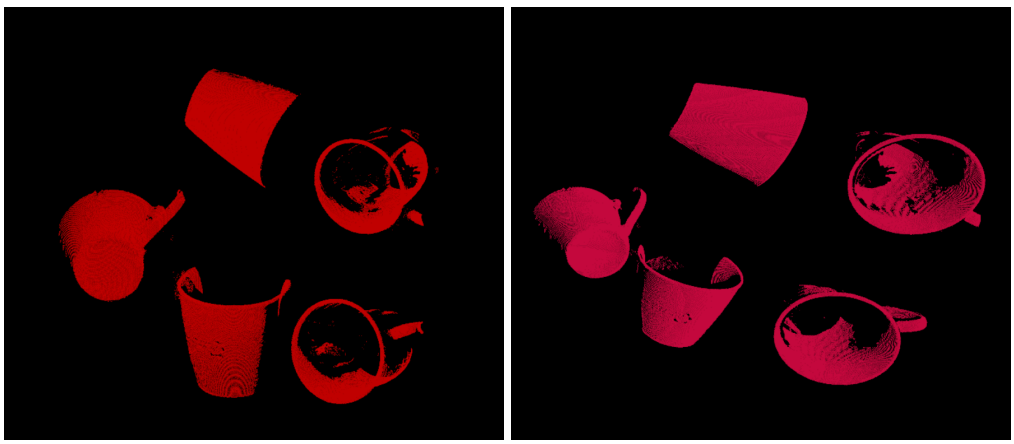
The algorithm is based on the assumption that only close points can represent parts of the same object. This is not always true because in scans there are several undercuts. In this way also points that in the point cloud are far from each other can belong to a single object. It is the case of Figure 3.4a where we can see that the handle is not in close contact with the cup or that there are a lot of empty spaces where there would be cup body.

Our solution is to transform a 3D scene into a 2D one that is seen from the

camera viewpoint. In order to implement this we fix a depth value on z axis [4] building a plane. Then for each point we compute the line that links this point with camera position and we select the intersection with fixed plane. In this way we find point position for 2D point cloud. This must delete completely undercuts and return a point cloud where only close points belong to the same object. In the next step with a simple hierarchical clustering, implemented using Euclidean distance, we can discover the right number of clusters to have in each of them a single subject. Counting clusters with a lot of elements we compute objects number. In Algorithm 7 is reported the pseudo-code.

This works fine with time of flight camera but there is an issue in structured light technology: in these instruments the sources are two and undercuts can be caused by both occlusions to the projector light propagation and to the camera view. Indeed, camera cannot acquire data if it cannot see a particular region or if there is no known light pattern on it. The correct undercuts elimination is obtained having a perfect subject knowledge to compute clutters direction, but we do not have this information. We partially solved this problem deleting the greatest undercuts that we notice are caused by the farthest source. So the change of algorithm described above is line computation: its direction is from focused point to the farthest source.

The results are shown in Figure 3.4b where we report the 2D transformation of 3D cups scene. The greatest effects are visible on the handles that now are in close contact with the other object part and the point of view does not allow to see some undercuts.



(a) 3D cups scene (b) 2D cups transformation

Figure 3.4: Transformation from 3D scene to 2D scene deleting undercuts

[4] we decide to use centroid z value but every other choice works

Algorithm 7 Objects number computation

```
1: procedure COMPUTE_OBJECTS_NUMBER(r, camera_position,
   projector_position, min_cluster_size)
2:     z ← centroid.z
3:     2D_point_cloud ← ∅
4:     for each point in point_cloud do
5:         camera_distance ← computeDistance(point, camera_position)
6:         projector_distance ← computeDistance(point, projector_position)
7:         if (camera_distance ≥ projector_distance) then
8:             line ← computeLine(point, camera_position)
9:         else
10:            line ← computeLine(point, projector_position)
11:        intersection ← findIntersection(line, z)
12:        2D_point_cloud ← 2D_point_cloud ∪ {intersection}
13:    clusters ← hierarchicalClustering(2D_point_cloud, r)
14:    count ← ∅
15:    for each cluster in clusters do
16:        if (cluster.size ≥ min_cluster_size) then
17:            count + +
18:    return count
```

This is a good input for an hierarchical clustering that creates group of points that are distant less than a threshold (a good value could be 1 or 1.5 mm). This technique is totally different from centre-based clustering explained in the previous subsections. Firstly it does not need the final groups number and does not necessarily build spherical regions. In the agglomerative approach it starts with a cluster for every points and merge the closest until the minimum distance between cluster is greater than the threshold. A downside of this procedure is that it is computationally onerous.

We remark that this method does not guarantee to compute perfect clusters. For the presence of some other undercuts the output is also composed by little groups of few hundred points that corresponds to isolated objects parts. To count the right targets number we have to consider only the greatest clusters that have thousands of elements. Once we return this count we can perform centre-based clustering.

3.4 Feature matching

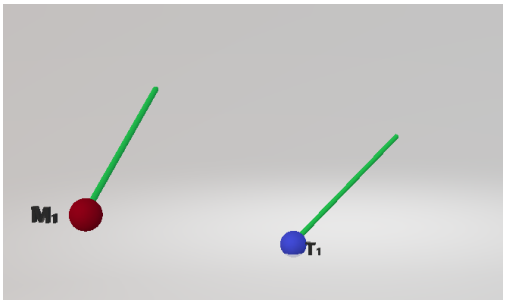
In this phase the input is composed by the features and their descriptions of both model and target. The goal is to find correspondences between them to compute objects similarities and identify their position and orientation in the scene. The output is a rotation matrix and a shifting vector that describe the model transformation to reach the same pose of the recognized subject.

We remember that there are two different features extracted: lines and circles. Their properties are just described in Section 3.2. In what follows we are going to explain the procedure implemented in this work assuming that we are matching lines. This is to simplify the description: there are no differences in lines or circles matching but the first is more intuitive and the figures are more easily understood. In the end we will present also the second case.

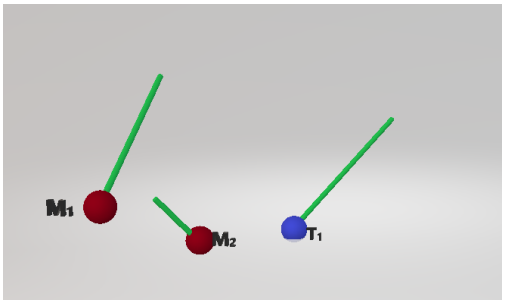
In Figure 3.5 there is a schema of the matching phase: spheres represent keypoints and segments their orientation. Firstly we select a model keypoint that we call M_1 (Figure 3.5a) and we search in the target descriptions list a keypoint with similar characteristics (Figure 3.5b). In detail we analyse the type and line length. When we find a target keypoint with a difference in the description lower than a threshold, we consider it as a possible match. Therefore we call it T_1 and go on with the analysis extracting a second model keypoint, M_2 (Figure 3.5c). At this moment we compute a first rotation matrix to find the transformation for M_1 orientation to reach the displacement vector between M_1 and M_2. In this way we can compute the direction to move T_1 to reach T_2: once we have the displacement module we can find its position. But if we use this transformation in the target scene we cannot be sure to find the correct region for a second keypoint that matches with M_2: we have another degree of freedom that is line roll. Therefore, once we have applied the transformation to T_1 orientation, we must consider a toroid, centred on the line and dimensioned to touch the just computed point, as a region where all the points, that belong to it, are valid for a match (Figures 3.5d and 3.5e). When we have selected target keypoints in this area, we compare their description with M_2 and we elect as correspondent the one with the smallest difference (Figure 3.5f). Considering not only the type and length but also the orientation, we compare these values with some thresholds to be sure that the two points are similar (the nearer description could be extremely different from the model point one because it is relative to the value of the other target keypoints description). Once we have executed these operations we block all degrees of freedom and we can find the transformation to obtain the target points position starting by the model (Figure 3.6)

This procedure must be applied to every model keypoint and for each of them, changing the first match with the other target keypoints. In this way the output is composed by several rotation matrices, so the computation of a score is

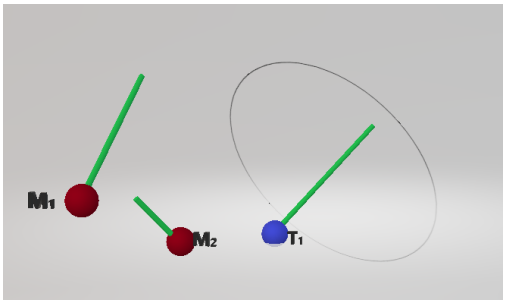
(a) Starting with a model keypoint M_1



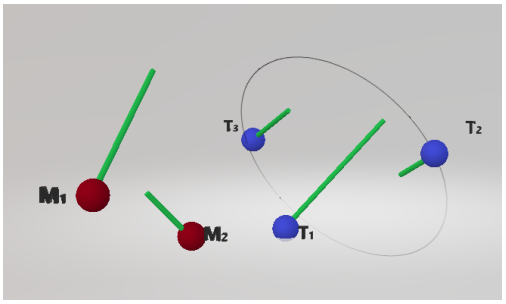
(b) We find a target keypoint T_1 with similar description



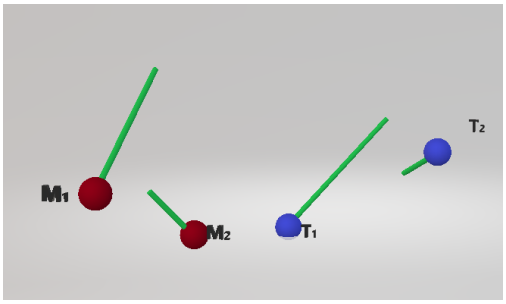
(c) We select another model keypoint M_2



(d) Analysing the rotation and shifting transformation between M_1 and M_2 we find a toroidal region where could be target keypoint T_2



(e) In this region we could find several target keypoint



(f) We select the one that has most similar description to M_2

Figure 3.5: Features matching schema

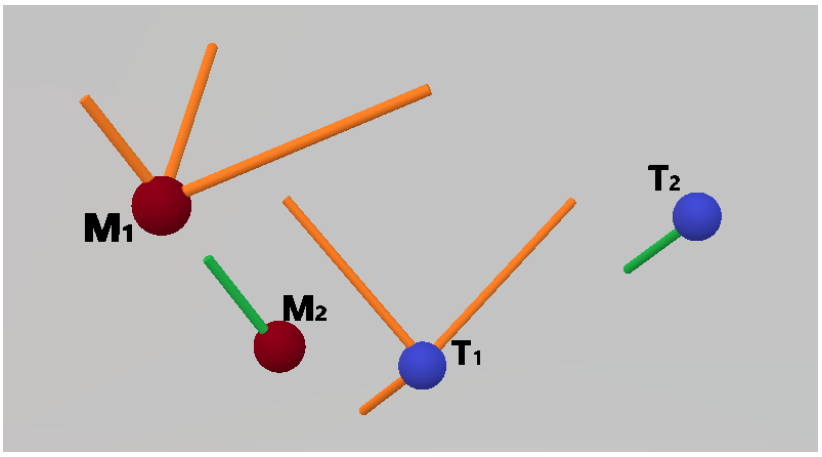


Figure 3.6: Identification of the reference system orientations

necessary to evaluate the best one. Finally we apply the transformation to each model keypoint to find its corresponding position in the target and we search around this area a target keypoint with similar characteristics. The final score is the number of matches over the number of the model features.

The complete pseudo-code of the algorithm is reported in Algorithm 8. We can find two shifts at lines 4 and 8: they are necessary to quickly compute rotation matrices.

When we are matching circle features, the procedure is the same: the only difference is that the orientation is the circle normal and the length is the radius size.

In order to solve only the detection problem, we can introduce a score threshold. In this way all the final transformations with a number of matches lower than the set parameter can be considered invalid or wrong and discarded. If there is no valid result the object is not detected so there is no subject in the scene similar to the model.

As we can see from the explanation, a crucial point of this algorithm is the computation of rotation matrices. In the next section we are going to describe how to calculate these and how we use them.

Rotation matrix computation

A rotation matrix R is a matrix used to perform a rotation in Euclidean space. Its main property is that $R \times R^t = I$ so this means that $R^t = R^{-1}$. One of its effects is that its determinant is 1 or -1 [5]. In case of det(R) = -1 the transformation that it represents is a rotation plus a reflection.

[5] $\det(R) \cdot \det(R^t) = \det(I) \Rightarrow \det(R) \cdot \det(R) = \det(I) \Rightarrow \det(R)^2 = 1 \Rightarrow \det(R) = \pm 1$

Algorithm 8 Features_Matching

procedure FIND_TRANSFORMATION ($descriptions_{model}, descriptions_{target}, \Delta_p, \Delta_l, \Delta_o$)
  $M \leftarrow \emptyset$
  for each M_1 **in** $descriptions_{model}$ **do**
    $shift(model, M_1)$
    for each T_1 **in** $descriptions_{target}$ **do**
      if ($M_1.type \neq T_1.type \vee |M_1.lenght - T_1.length| > \Delta_l$) **then**
        continue
      $shift(target, T_1)$
      for each $M_2 \neq M_1$ **in** $descriptions_{model}$ **do**
        $displacement \leftarrow M_2.position$
        $rotation_1 \leftarrow computeRotation(M_1.orientation, displacement)$
        $rotation_2 \leftarrow computeRotation(M_1.orientation, M_2.orientation)$
        $module\_displacement \leftarrow computeNorm(M_2.position)$
        $position \leftarrow rotation_1 * T_1.orientation * module\_displacement$
        $orientation \leftarrow rotation_2 * T_1.orientation$
        $possible\_matches \leftarrow \emptyset$
        for $angle = 0$ **to** 2π **do**
          $new\_position \leftarrow rotate(angle, T_1.orientation, position)$
          $point \leftarrow findNearKeypoint(position_1, \Delta_p)$
          $T_2 \leftarrow getKeypointDescription(point)$
          if ($checkSimilarity(T_2, M_2, \Delta_l, \Delta_o)$) **then**
            $closeness \leftarrow computeCloseness(T_2, M_2)$
            $possible\_matches \leftarrow possible\_matches \cup [T_2, closeness]$
        $T_2 \leftarrow findMaxCloseness(possible\_matches)$
        $transformation \leftarrow computeTransformation(M_1, M_2, T_1, T_2)$
        $score \leftarrow computeScore(transformation)$
        $M \leftarrow M \cup [transformation, score]$
      $shift(target, -T_1)$
    $shift(model, -M_1)$
  return $getBestScoredTransformation(M)$

Focusing on our context, this matrix has 3x3 size and can be generally expressed in the following way:

$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) - u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_x u_y(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

where u is rotation axis and θ is rotation angle [30]. The most complex situation to compute this matrix is when we know two vectors and we want to find the transformation between them. In this case we do not know both the axis and the angle but they can be easily computed: the first can be obtained with the cross product between the vectors, the module of this result is $\sin\theta$ and the module of the dot product is $\cos\theta$. This is what the function *computeRotation* at lines 11 and 12 in Algorithm 8 does.

Rotate at line 18 is simpler because as input it takes the first vector, the angle and the axis. θ and u are known, it computes rotation matrix and multiplies this with the vector. The output is vector rotated of an angle θ with respect to the axis u.

In the explained procedure we do not treat the transfer of rotation data from the model reference system to the target one. This change complicates the problem because the rotation axis direction varies quite often. We start by computing the angle ($\theta_{M_1 disp}$) and the axis ($u_{M_1 disp}$) between M_1 orientation and M_1-M_2 displacement vector. Then, we do the same with M_1 and T_1 orientation. Using the last transformation we can rotate $u_{M_1 disp}$ to obtain its correspondent vector in the target context ($u_{T_1 disp}$). At this point we can rotate T_1 orientation respect to $u_{T_1 disp}$ of $\theta_{M_1 disp}$. To find T_2 orientation we can do the same than the first step computing axis and angle between M_1 and M_2 orientation.

We have to remark two points:

- all the rotations described must be done with a vector of norm equal to 1 (direction without module information);
- the previous procedure finds a possible T_2 position and orientation but we have to consider also the line roll. Therefore, in the following steps we have to check in the toroid centred on the first computed position.

The last computation that we are going to explain produces the rotation matrix that describes the transformation to overlap the model to the recognized object in the target scene (line 25). As input we have four keypoints matched in pairs: $M_1 \rightarrow T_1$ and $M_2 \rightarrow T_2$. The final transformation (R) is computed by adding the contributes of two factors: the rotation between M_1 and T_1 orientation (R_1) and the one between the new M_2 orientation (obtained multiplying R_1

for M_2 original direction) and T_2 orientation (R_2):

$$R = R_1 * R_2.$$

This can be applied to both positions and orientations because all the object transformations in exam are rigid. If there are some small deformations we can insert a threshold for the position and the orientation differences to add a degree of tolerance.

3.5 The algorithm execution flow

In the previous sections we analysed in detail all the algorithms that compose our work. In order to review what we have explained until now, we are going to show all the processing phases that are applied to a point cloud to recognize an object model in its scene.

Firstly, we have to create a model of the object that we are recognizing. In a point cloud that contains it, we select its area and remove any other particulars that not belong to it. This phase must be performed manually with a 3D graphic software. Then we continue analysing this selection: we find the first level key-points with *Normals gradient analysis* or *Neighbourhood analysis* and among these points we find circles and lines with RANSAC. Then, we save the centre of every feature and its description that is composed by length, direction and position for the lines and radius size, the normal to the surface and the position for the circles.

Now we have to obtain the same information in the target scene, therefore we acquire a scan and we remove box floor (if it is visible) with RANSAC. To correctly analyse every object we must to create a single point cloud for each of them. Therefore we count the targets simulating a 2D view of the scan to reduce as much as possible the undercuts and finding, with hierarchical clustering, groups of close points of a reasonable size. This count is used as input of a centre-based clustering that splits the scene. For each point cloud we apply the procedures just explained for the model.

Once we have computed the feature description for both model and target we continue with the matching phase in order to find correspondences between them. Based on these matches we compute a transformation to overlap the model point cloud on a target: according to the number of correspondent points that are closed each other we establish a score value that describe the accuracy of this transformation. Choosing the best score and verifying that this score is greater than a threshold, we return the position of recognized object.

Repeating this last phase for each element we can find all the occurrences of the model subject in the scene.

Chapter 4

Implementing details and tools used

In this chapter we present the instruments used in acquisition and measuring phase and we describe some implementing details.

The main part of this code is produced in Visual Studio environment using C#, a programming language object-oriented developed by Microsoft within its .NET initiative and later approved as a standard by Ecma and ISO.

4.1 Point Cloud Library

In order to achieve this work we needed to a library where we could find the basic data structures and some operative algorithms. For this aim the ideal tool is Point Cloud Library (PCL) [31] a standalone open project for 2D/3D image and point cloud processing. It is developed in C++ so we have designed a Wrapper to use this library in our working set.

Thanks to this tool we can exploit its data structure (point cloud), all the input/output methods (ex. to load and write point cloud, to convert the different extensions such as ".pcd" or ".ply"), RANSAC and algorithms to compute normals.

In particular RANSAC implementation has played an important role for the number of available mathematical models, both 2D and 3D. At this moment the features searched among first level keypoints are only two-dimensional but we have just designed the structure for future introduction of three-dimensional models. Another advantage is that several versions of RANSAC are implemented. We tried three of them: Maximum Likelihood Estimation SAmple Consensus (MLESAC)[32], PROgressive SAmple Consensus (PROSAC)[33] and the standard RANSAC.

In order to compare these versions we use a scan acquired for bin-picking of a component pump that connects two tubes. This data belongs to a project

that is been developing by Euclid Labs company and for this reason was easily available. All experiments in this Section are executed considering this context. The same scene will be used in Section 5.3 to make other qualitative tests.

MLESAC

This algorithm is a generalization of RANSAC estimator. It adopts the same sampling strategy as RANSAC to generate putative solutions but chooses the one that maximizes the likelihood rather just the number of inliers.

We have tested this version comparing it with original RANSAC. The results are very good in time execution because both for lines and circles MLESAC spends only 1 or 2 milliseconds instead of 7 or 9 milliseconds for lines and 50 or 60 milliseconds for circles. But analysing the other performances we decide to not use it. In particular with the same parameters value [1] the original algorithm finds 30 lines and 19 circles against 1 and 18 of the new version. Another qualitative comparison is between goodness of extracted features. There is not an objective evaluation but we can see that the underlined geometric forms have a greater meaning in RANSAC output than in MLESAC. The only way to justify this sentence is to show the results: in Figure 4.1 we report first line and circle that the two methods found (underlined points over white scan).

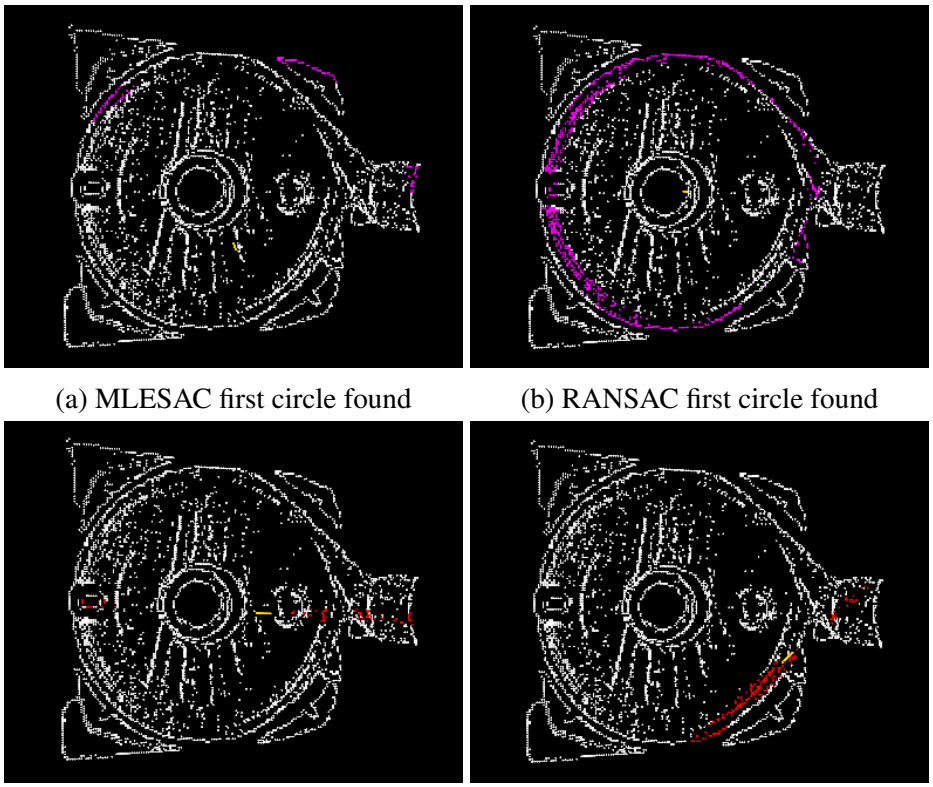
PROSAC

The goal of PROSAC algorithm is to obtain the same result of RANSAC in a faster way. So to better understand its improvements we have to dedicate few lines on a possible RANSAC implementation (to a general algorithm description we refer to Section 3.1). It is viewed as a black box that generates N tentative correspondences, the error-prone matches are established by comparing local descriptors. The set U of tentative correspondences contains an a priori unknown number I of correct matches (inliers). The inliers are consistent with a global geometric model that is found by fitting a model to a randomly selected subset of U. The hypothesize-and-test loop is terminated when the probability of finding a superior solution falls below a pre-selected threshold.

The time complexity of RANSAC depends on N (number of tentative correspondences), I, and the complexity m of the geometric model. The average number of samples drawn is proportional to $(\frac{N}{I})^m$.

PROSAC introduces a new sample-and-test matching approach. The method achieves large computational savings (with speed-up factors of the order of 10^2

[1] we fixed threshold for the minimum number model points and error toleration to be consider inlier



(a) MLESAC first circle found (b) RANSAC first circle found

(c) MLESAC first line found (d) RANSAC first line found

Figure 4.1: Comparison between RANSAC and MLESAC in goodness of output

compared to RANSAC) by exploiting the linear ordering structure of U. The ordering is defined at least implicitly in all commonly used local matching methods, because the set of tentative correspondences is obtained by first evaluating a real-valued similarity function $q(\cdot)$ that is subsequently thresholded to obtain the N correspondences. Correlation of intensities around points of interest [34], Mahalanobis distance of invariant descriptors [35] or the ratio of distances in the SIFT space of the first to second nearest neighbour [10] are commonly used examples of $q(\cdot)$. In PROSAC, samples are semi-randomly drawn from progressively larger sets of tentative correspondences. The improvement in efficiency rests on the mild assumption that tentative correspondences with high similarity are more likely to be inliers. More precisely, it assumes that the ordering defined by the similarity used during the formation of tentative matches is not worse than random ordering.

The main difference between RANSAC and PROSAC is the first step sampling: unlike in RANSAC in PROSAC the samples are not drawn from all data, but from a subset of the data with the highest quality. The size of the hypothesis

generation set is gradually increased. The samples that are more likely to be uncontaminated are therefore examined early. In fact, PROSAC is designed to draw the same samples as RANSAC, only in a different order.

The comparison between these two algorithms almost confirms the just explained theoretical aspects. As test we execute the feature extraction phase on pump components: we note execution time, features extracted and final result of recognition task.

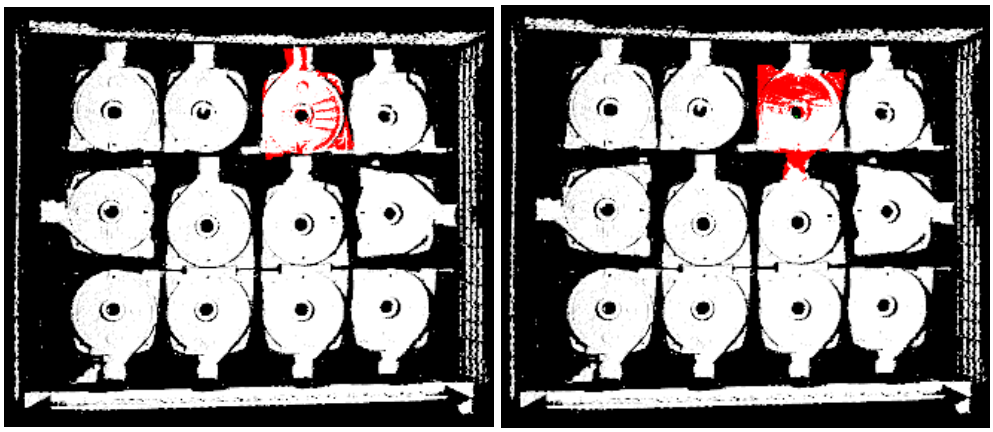
In Table 4.1 we report execution time comparison: data are expressed in milliseconds and represents the time spent to extract the features. This test can be executed in this way thanks to the fact that the features found with both the versions are exactly the same. We can see first five lines and circles: the greatest differences are in time to find circles. Analysing all the data (25 lines and 17 circles) and using PROSAC we compute an average improvement of 4 ms that reaches 23.4 ms considering only circles. Until now, this is in agreement with the algorithm goal but we successively find a discrepancy: studying accurately the results we discover that RANSAC finds more features then the other (30 lines and 19 circles) setting the same parameters value[2]. These more extractions seem to have greatly effects on final results. Indeed in a test, that we will describe in detail in Section 5.3, we compare the recognition of a pump component using both RANSAC and PROSAC: the former well recognizes 10 of 12 objects but the latter only 8. In Figure 4.2 we show a case where RANSAC identifies correctly subject position and orientation but PROSAC does not. In white we can see the box scan with the objects inside it; in red there is the projection of the model as result of the final transformation computed.

At the end of this analysis we decide to continue to use RANSAC privileging precision and accuracy despite execution time.

Feature	RANSAC	PROSAC	Feature	RANSAC	PROSAC
Line 0	7	10	Circle 0	67	22
Line 1	9	8	Circle 1	57	52
Line 2	6	8	Circle 2	49	23
Line 3	6	7	Circle 3	45	3
Line 4	6	7	Circle 4	40	41

Table 4.1: Time execution comparison between RANSAC and PROSAC (times expressed in milliseconds)

[2] we fixed threshold for the minimum number model points and error toleration to be consider inlier



(a) Recognition with RANSAC (b) Recognition with PROSAC

Figure 4.2: Comparison between RANSAC and PROSAC in recognition

4.2 3D Cameras

In this section we are going to describe 3D cameras that we use to test our algorithm performance. All are based on structured light projection: one is an industrial camera (Photoneo [36]) and the other a user-consumer (Microsoft Kinect v1 [37]).

This technology is composed by two physical sensors: a projector that throws known pattern in the scene, and a scanner that recognizes the pattern and analysing light deformations allows shape objects reconstruction [38].

In the following there are all sensors details.

Photoneo

Figure 4.3: Photoneo sensor

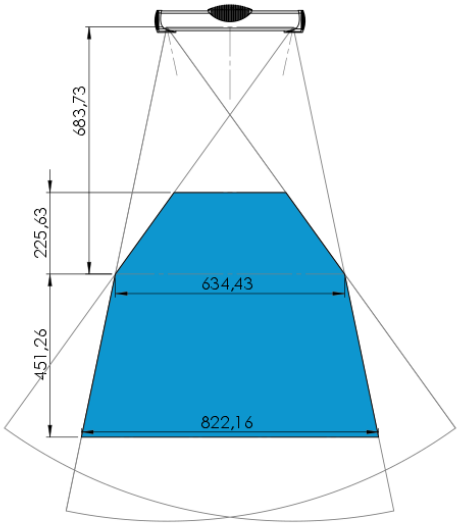


Figure 4.4: PhoXi 3D Scanner M range values

A conventional structured light technique is derived from a stereo vision. This method, that is inspired by the nature of a human vision, exploits data acquired in the same moment from two close cameras positioned in a known place. Finding the correspondences of points in the two frames, it is possible to compute their position in 3D space by triangulation. While the idea is very simple, the technique is computationally complex and requires a well textured object. Due to the real-world restrictions, the technique is not suitable for industrial use.

For these reasons many companies focuses on different sensor based on structured light. As we have just explained, a camera is interchanged by a pattern projector which emits a well defined structure illumination. Source of light pattern can be a conventional 2D projector similar to the ones used for multimedia presentations. Both the sensors must be focused on the same scanning area and posed in well known positions, determined by a system calibration.

In Photoneo the projector emits a set of coding patterns projected in a succession, one after another. These coding patterns encodes a spatial information. Camera captures the scanning area once per every projected pattern. For every image point A, the algorithm decodes the spatial information encoded in the succession of the intensity values captured by this point. This spatial information encodes the corresponding “image point B” in the view of the projector, as in stereo video approach. With the correspondence, the algorithm computes an exact 3D position of the object point.

The advantages of this technique are that it works irrespectively of object

Mode	High res mode	High speed
Depth Map resolution	3.2M points	0.8M points
Point size	0.33 mm	0.66 mm
Absolute accuracy	≤ 100 μm	≤ 100 μm
Z noise	≤ 100 μm	≤ 100 μm
FPS	2.5 fps	5 fps
Data acquisition time	400 ms	200 ms
Data acquisition time	4 s	2 s
3D points throughput	16 Million points per second	
GPU	NVIDIA Maxwell 1 TFLOPSwith 256 NVIDIA CUDA Cores	

Table 4.2: PhoXi 3D Scanner M characteristics

texture, high 3D reconstruction and speed of acquisition.

One of the greatest problem of light structured based systems is the complexness of pattern projectors. The conventional design based on DMD (Digital Micromirror Device) or LCD technology is expensive and optically ineffective. The most of the light created is lost in the system due to the complex optical set-up. This light is than converted to heat. Due to the nature of light sources used in these systems, the depth of field is very limited.

The improvements of Photoneo PhoXi family is that they use an own projection system based on coherent laser radiation. They emit light of a specified wavelength in addition to high quality glass bandpass filters used in a camera optics. Moreover the system delivers outstanding resistance to other light sources even in challenging indoor environments with multiple lights. The result is high depth of field similar to laser line triangulation systems [39].

The hardware specifics are:

- latest CMOS sensors from Sony to deliver an excellent performance with a small energy footprint;
- industry leading camera sensing performance made by Ximea;
- NVIDIA Jetson platform, to make system as enough powerful to execute advanced reconstruction algorithms with high resolution and fps;
- PhoXi capturing and processing pipeline is capable of delivering 16 million measurements per second, either in 3.2 megapixels at 5 fps, or 0.8 megapixels at 20 fps.

Figure 4.4 shows the range values of a specific product of Photoneo PhoXi family, PhoXi 3D Scanner M, and in Table 4.2 we report its performances. The data meanings are [40]:

- Depth map resolution: maximum number of measured points (resolution of camera sensor);
- Point size: distance between two measured points on the plane perpendicular to the camera in the focus distance from the sensor;
- Absolute accuracy: precision of point measurement. It is the standard deviation of the measurement error in the whole measuring range of the device;
- Z noise: standard deviation of the noise (measured on a diffuse surface with 80 percent albedo). The noise level describes the quality of the sensor to capture local surface details. The noise distribution of the sensor is similar to Gaussian;
- FPS: maximum number of triggered frames per second, in fastest acquisition mode;
- Data acquisition time - best case (white diffuse objects): fastest possible acquisition time;
- Data acquisition time – worst case (dark objects): longest expected acquisition time;
- 3D points throughput: number of 3D points that can be reconstructed in a second in sequential scans.

Microsoft Kinect v1

Microsoft Kinect, announced in June 2009 under code name ”Project Natal”, is based on structured light built for commercial use. Its production changed 3D camera trade because it combines medium-high precision with low costs.

It is a mix of Microsoft built software and hardware. Kinect v1 hardware includes a range chip set technology PrimeSense, which developed a system consisting of an infra-red projector, camera and a special microchip that generates a grid from which the location of a nearby object in 3 dimensions can be ascertained. The depth sensor combines projector with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions. The sensing range of the depth sensor is adjustable [37]. In the bar there is also a low definition RGB camera: for this reason Kinect is considered a RGB-D sensor.

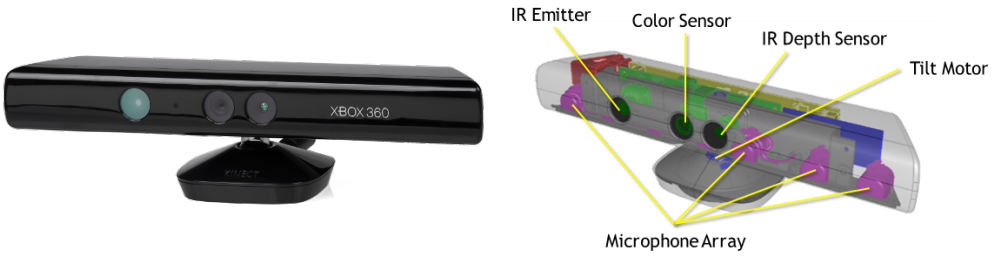


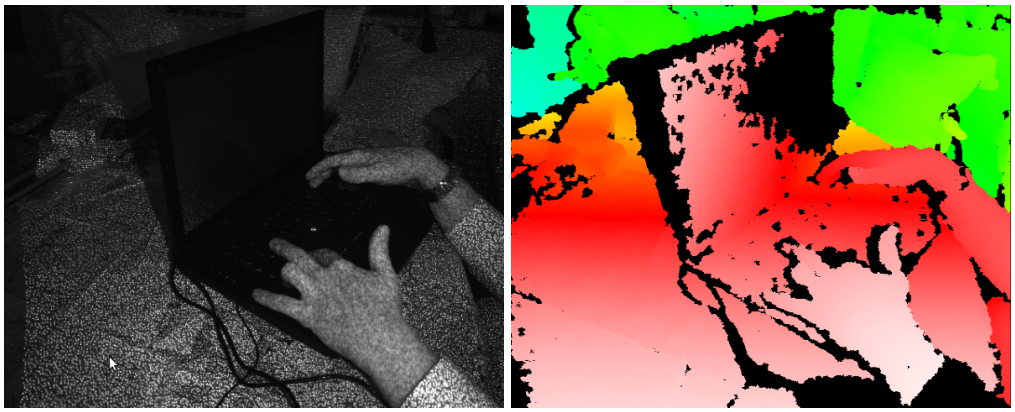
Figure 4.5: Microsoft Kinect v1 sensor

In Figure 4.6 we report a view of the infra-red points that are projected in the scene and the reconstruction of the same environment as a depth-map. It is interesting to see how the sensor works and what performances produces. In Table 4.3 there are its characteristics [41].

A considerable improvement in resolution and 3D image quality is given by Fusion mode. This system reconstructs a single scanning starting by multiple acquisitions from different viewpoints tracking objects and camera position and orientation. Steps to realize this procedure are [42]:

1. depth map to point cloud conversion: here it computes depth float value of each point starting from colour and calibration informations;

2. it computes global/world camera pose to track sensor position and orientation. To do this it can use two different algorithms of tracking: *NuiFusionAlignPointClouds* that allows to align two point clouds even if they are acquired from two viewpoints or from the same view but not in the



(a) Infra-red points projection (b) Kinect depth-map

Figure 4.6: Kinect results both in infra-red projection and depth-image computation

same instant; *AlignDepthToReconstruction* that provides a more accurate camera tracking but it is sensible to object movements on the scene;

3. fusion on the same volumetric scene vision of the environment, thanks to position data computed in the previous step. In this way empty region in a frame can be filled from informations acquired in the next moments. An essential procedure is to compute average point position to reduce noise. It is important to fuse these data continuously;

4. as last step it establishes a reference system (that quite always corresponds to the current sensor pose) and shares point cloud with values referred to it.

Limitations to this reconstruction are linked to the only depth stream use. Indeed, if we are scanning a quite planar scene or objects with little variations in depth, tracking performs with fusion algorithm fails and results are very bad [42]. For this reason it is important to not position the camera to have a top view of the environment. In our tests we place the sensor in a top position but we orient it with an angle of around 45 degrees respect the scene.

Using this acquisition mode we reach a resolution of around $1 \sim 2$ mm per voxel.

Parameter	Value on Microsoft Kinect v1
Color camera	640 x 480 [640 x 480 at 30fps]
Ir camera	1280 x 1024 [640 x 480 at 30fps]
Operation Range	0.8 m - 3.5 m
Field Of View	58° H, 45° V , 70° D
Spatial resolution	3 mm (at 2 m distance)
Depth resolution	1 cm (at 2 m distance)

Table 4.3: Microsoft Kinect v1 characteristics

4.3 Kreon Baces arm

Figure 4.7: Bases measuring arm

Another important instrument used to test our algorithm precision is Kreon Baces arm. It is a measuring arm with six axis that has a tip with three buttons to allow software control. The configuration reported in Figure 4.7 is used to take measures touching the point: arm ends with a sphere of 4 mm of diameters. There is another configuration that inserts an handle to add new axis (they becomes seven) and Kreon scanners: this allows to take measures without touching points.

The conveniences of this arm in measuring are its lightness to easily move it in every environment, its precision that allows a point repeatability lower than tenth of millimetre, and the presence of buttons on the tip. This last feature is essential to take measure alone, programming an opportune control software. Another plus is the use of a foot pedal to do the same tasks as tip buttons but leaving hands to move the arm without pressing need [43]. This guarantee more freedom in measuring.

Kreon provides different versions of this arm that differences mainly in dimensions and materials. Some products are made in aluminium some others in carbon-fibre: in this way they can offer light and stable instruments that are ideal for metrological applications, reverse engineering, digitization, inspection, rapid prototyping and so on.

Chapter 5

Results

In this chapter we are going to explain our tests to evaluate the performances of this new approach. We reproduce multiple scenes with different objects using two 3D cameras. We work in both ideal and real environment to compare the theoretical results and the practical ones.

It is crucial to say that in every case we use the same configuration with Neighbourhood analysis to extract first level keypoints. The only parameters that we can change from an example to the other are:

- *Ransac size*: this is the lower threshold given to RANSAC algorithm. It means that the lines and circles that it returns must have at least this number of inliers;
- *Ransac threshold*: it measures the tolerance to consider a point as an inlier of a geometric model;
- *Neighbourhood position range*: it is an input of the algorithm for first level keypoints extraction. It is the threshold that manages the displacement of the neighbourhood centre to consider the pivot as an edge point or not;
- matches parameters: these are the tolerance on position, orientation, length, radius size and score used in the feature matching phase.

These values change in order to match the camera characteristic (as resolution), object shape complexity and scan goodness. This last variable is determined by the object physical properties such as reflection, transparency and so on.

In the following sections we analyse singularly each test, specifying set-up and results obtained.

5.1 Test in ideal environment

Recognition with clean data

The first test goal is to check if the feature matching phase is designed and implemented correctly. We build a model in simulation composed by two circles and two lines: this is the ideal output of a perfect first level keypoints extractor made from a scan without noise. In this way we delete all the components that could prevent the recognition. With this test we can also implement a first qualitative estimation of precision. The created model can be seen in Figure 5.1.

After the model creation, we also build a target scene. Our aim is to check some different transformations in terms of angle size and axis direction. Therefore we rotate the model point cloud using singularly x, y and z axis and for each of them we vary the angle in $\{\frac{\pi}{5}; \frac{2\pi}{5}; \frac{3\pi}{5}; \frac{4\pi}{5}\}$. We save all the results and put them in the same data structure adding appropriate translations. Finally, we perform the recognition, skipping the first level keypoint extraction step, with the set-up shown in Table 5.1.

In Figure 5.2 we report the results of this test. Points in red represent the projection of the model on the target environment obtained multiplying it with the computed rotation matrix. In white there are the target points. In all the recognitions we notice a perfect match between the projection and target points:

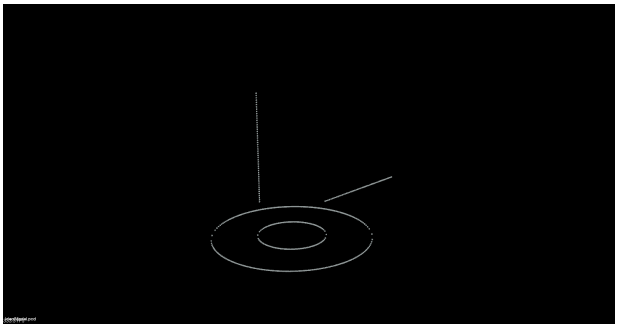
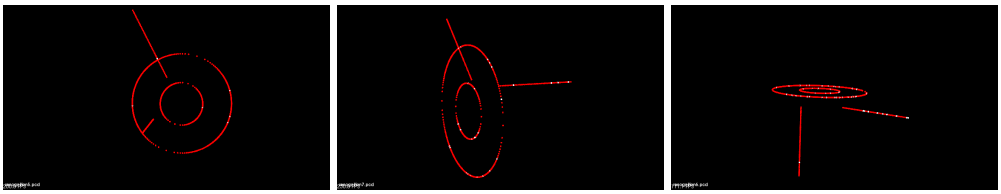


Figure 5.1: Ideal model created for test in ideal environment

Parameter	Value	Parameter	Value
Ransac size	40	Matching length	10 mm
Ransac threshold	$1/10^{-3}$ mm [circles/lines]	Matching radius size	10 mm
Neig. position range	-1	Matching position	2 mm
Score	-	Matching orientation	0.25 rad

Table 5.1: Set-up for test in ideal environment without noise



(a) Rotation axis = x (b) Rotation axis = y (c) Rotation axis = z

Figure 5.2: Results of test in ideal environment without noise

changing axis or angle size the results remain equal. The proof of coincidence is the presence of some white points between the two red ones. We reported only 3 results obtained with different axis and rotation angles to have some examples.

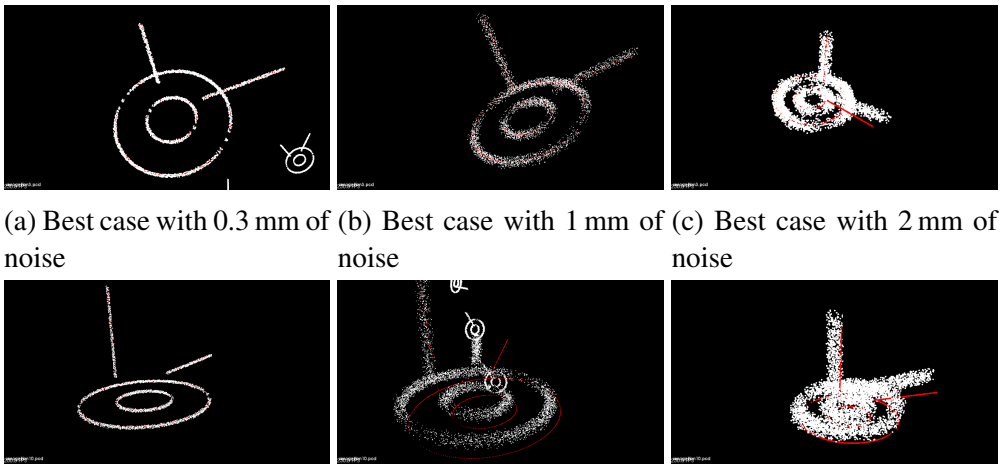
System behaviour in presence of noise in target point cloud

The second test in ideal environment checks the robustness to noise. To do this we adopt the same model of Figure 5.1 but we insert a settable degree of error in the target scene. In detail we replace the position of each target point with another, selected randomly from a neighbourhood with size equal to the input value. Our goal is to repeat the experiment with several degrees of error to find the threshold beyond which the system does not work correctly. For this check we use the model feature description computed in the previous test to start from clean informations. The parameters set-up is described in Table 5.2 and they are not changed during trails.

Parameter	Value	Parameter	Value
Ransac size	40	Matching length	10 mm
Ransac threshold	1/0.5 mm [circles/lines]	Matching radius size	10 mm
Neig. position range	-1	Matching position	2 mm
Score	-	Matching orientation	0.25 rad

Table 5.2: Set-up for test in ideal environment with noise

The results are illustrated in Figure 5.3 and are divided into 2 parts. We remember that the target scene is composed by the model copied and rotated into twelve different positions in the same point cloud. The first result part (first line of images) represents three recognitions applied to the same model transformation with different degrees of error (0.3 mm; 1 mm; 2 mm); the second part (second line of images) represents the same recognitions with the same three degrees of error but applied to another model transformation. We show these ex-



(a) Best case with 0.3 mm of noise (b) Best case with 1 mm of noise (c) Best case with 2 mm of noise

(d) Average case with 0.3 mm of noise (e) Average case with 1 mm of noise (f) Average case with 2 mm of noise

Figure 5.3: Results of test in ideal environment with noise

amples because the first is the best in terms of precision and the latter explains an average behaviour. Indeed in images 5.3a,5.3b and 5.3c we can notice that until the noise is lower than 2 mm, the model is perfectly contained in the target. In 5.3d, 5.3e and 5.3f model is not well recognized already with an error degree of 1 mm. In general we obtained 12 good identifications of position and orientation over 12 with 0.3 mm of noise, 9 with 1 mm and 2 with 2 mm.

We analyse this result to try to understand why the recognition fails. This is caused prevalently by RANSAC that, having as input unclean data, does not find the same lines and circles of the model in the target. In this way there are no common extracted features and the matching phase does not return enough correspondences to perform the recognition. This means that the first level keypoint extraction is crucial: its role is to clean the input point cloud and allow to focus only on a few but important points.

This is a qualitative analysis and its goal is only to understand the behaviour of the presented system. To find an exactly error and precision evaluation we do specific tests visible in Section 5.5.

5.2 Recognition of the model in multiple object types scene

Here we focus on the detection task: in a scene with different object types we have to identify the one that corresponds to the model. So we will not analyse the

precision for the position and orientation estimation but only for the recognition of the right element.

To test this capability we create a point cloud with the scan of 3 different subjects: a jar, an ornament with a robot shape and a support. We analyse the scene in several executions changing the model given as input: in the first we recognize the support, in the second the robot and in the last the jar. In Table 5.3 there is the set-up. In the case of the jar, we have to consider a greater threshold score (precisely 0.4) because it is a simpler shape than the others. This fact produces only a few features therefore the recognition is more sensible to errors: only 2 or 3 matches are needed to label an element as our target because they exceed 10% of keypoints. Increasing this threshold the algorithm must to find more matches to select a subject so few errors are irrelevant. In Figure 5.4 we can see that giving in input a particular object with the right parameters in the same scene, only the corresponding target is detected. It is important to remark that the scans used as model are different than the ones used to compose the environment.

They are all acquired using Photoneo camera.

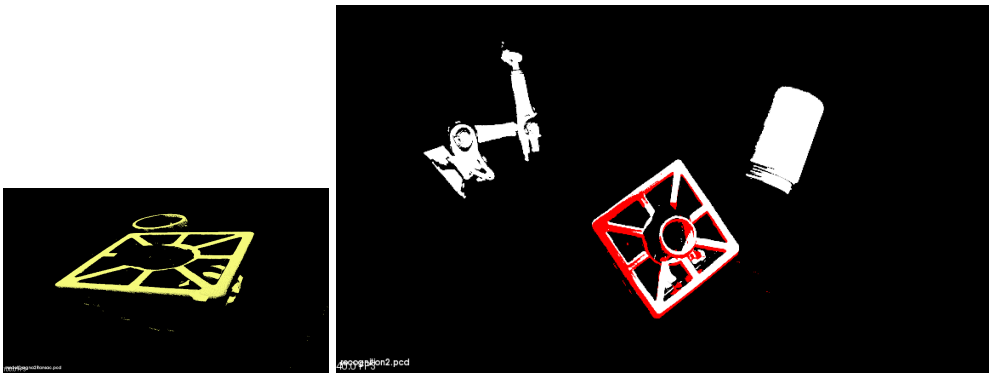
5.3 Tests on real objects recognition

In this section we are going to present some examples of recognition in real-life contexts. To underline our system versatility we recognize other two objects: pump components and cups. In the previous tests we have just handled 3 different subjects and the transition from one to the other caused only one parameter change. In this case more settings must be modified because we have not only to detect the right object but also identify its exactly position and orientation. In addition the selected models are rather symmetrical: this complicates a lot the identification of the right orientation.

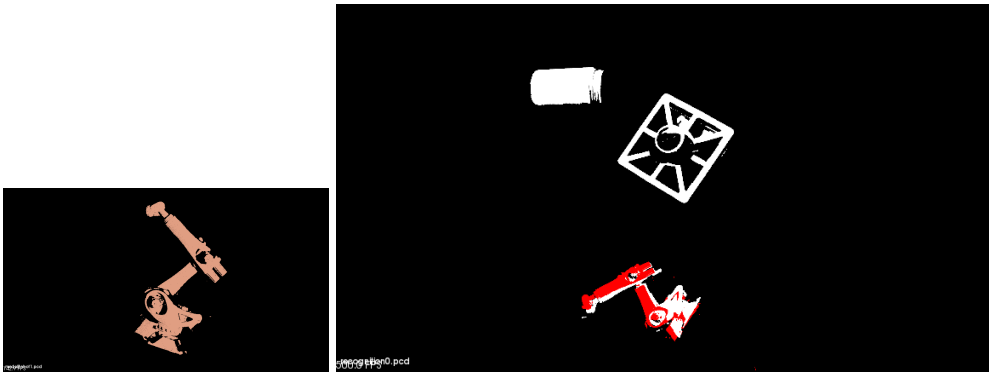
In the next subsection we are going to explain the set-up and the results.

Parameter	Value	Parameter	Value
Ransac size	40	Matching length	15 mm
Ransac threshold	5 mm	Matching radius size	15 mm
Neig. position range	0.5 mm	Matching position	10 mm
Score	0.1	Matching orientation	0.25 rad

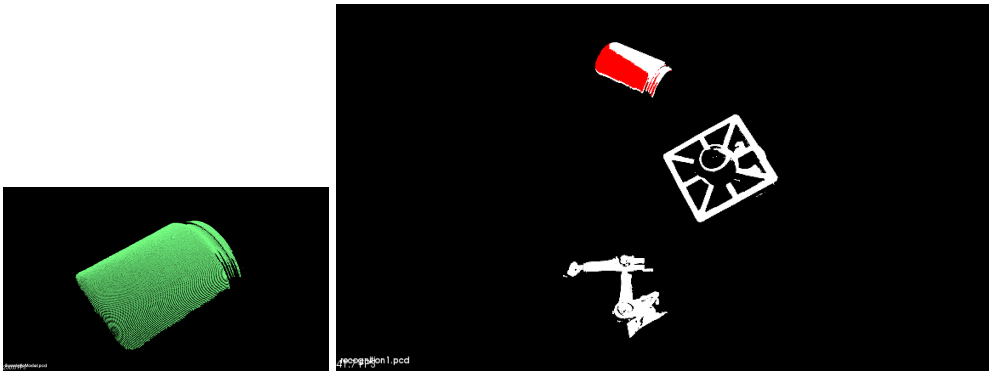
Table 5.3: Set-up for support and ornament detection



(a) Support detection



(b) Robot detection

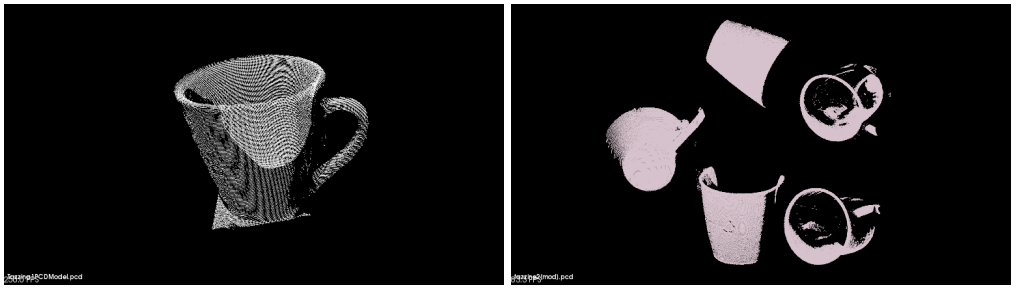


(c) Jar detection

Figure 5.4: Results of test for detection task

Cup recognition

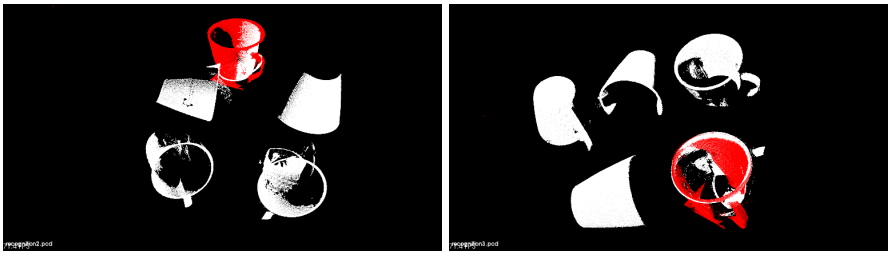
The test on cups is quite interesting because it shows three important aspects of our work performances:



(a) Cup model (b) Scene of recognition

Figure 5.5: Model and target used in cup recognition test

(a) Item 0 (b) Item 1



(c) Item 2 (d) Item 3

(e) Item 4

Figure 5.6: Results of cup recognition test

Parameter	Value	Parameter	Value
Ransac size	40	Matching length	10 mm
Ransac threshold	2 mm	Matching radius size	10 mm
Neig. position range	0.5 mm	Matching position	10 mm
Score	0.1	Matching orientation	0.25 rad

Table 5.4: Set-up for cup recognition

- its behaviour in case of symmetrical objects: only the handle can establish a direction of the subject;
- its behaviour in case of similar but not equal items: there are two different cup types. One is smaller and more curved, the other is taller and narrower;
- its behaviour in case of clutters: cups are positioned randomly and show different sides to the camera. This is similar to the problem of clutters because not all the parts are visible (for example in several cups there is not an handle).

As we can see in Figure 5.6 with the set-up of Table 5.4 all cups are well recognized. When an handle is visible, the model is also positioned in the right orientation. An example is Figure 5.6b where we can see that the handle, in red, of the model overlaps the white one of the target. In some cases like Figure 5.6a or 5.6e model completes the lack of information of target and matches perfectly with the known parts.

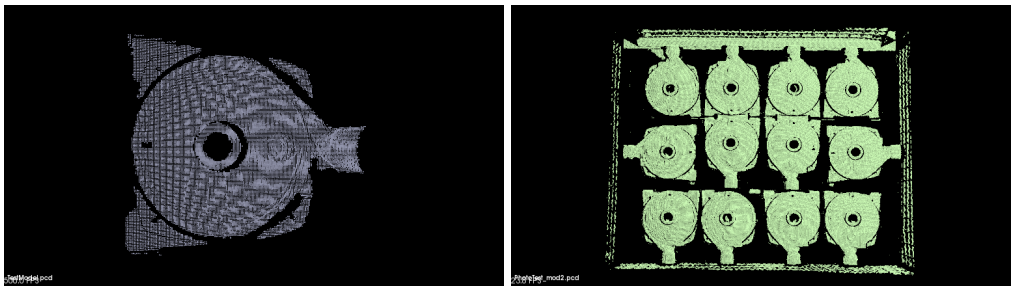
Pump component recognition

Here we retrieve scans used in RANSAC versions evaluation in Section 4.1. It is an example of application in actual industrial context: Euclid Labs has to realize a bin picking system to extract a series of pump components that link two tubes for a company of Caldiero (VR). To study our algorithm performances we select scans acquired here and we apply these as input. The camera used is always the Photoneo and the parameters set-up is reported in Table 5.5.

As we can see in Figure 5.8 there are some recognitions that perfectly compute both position and orientation. Items 0, 1, 3, 6 are examples of this precision: the shifted and rotated model overlaps completely the object. Other results are less accurate but can be considered correct for the error entity: items 4, 5, 8, 9, 10 and 11 register right position but little wrong orientation. This inaccuracy would be reduced applying Iterative Closest Point (ICP) or other algorithms to

Parameter	Value	Parameter	Value
Ransac size	50	Matching length	60 mm
Ransac threshold	5 mm	Matching radius size	60 mm
Neig. position range	0.6 mm	Matching position	20 mm
Score	0.1	Matching orientation	0.25 rad

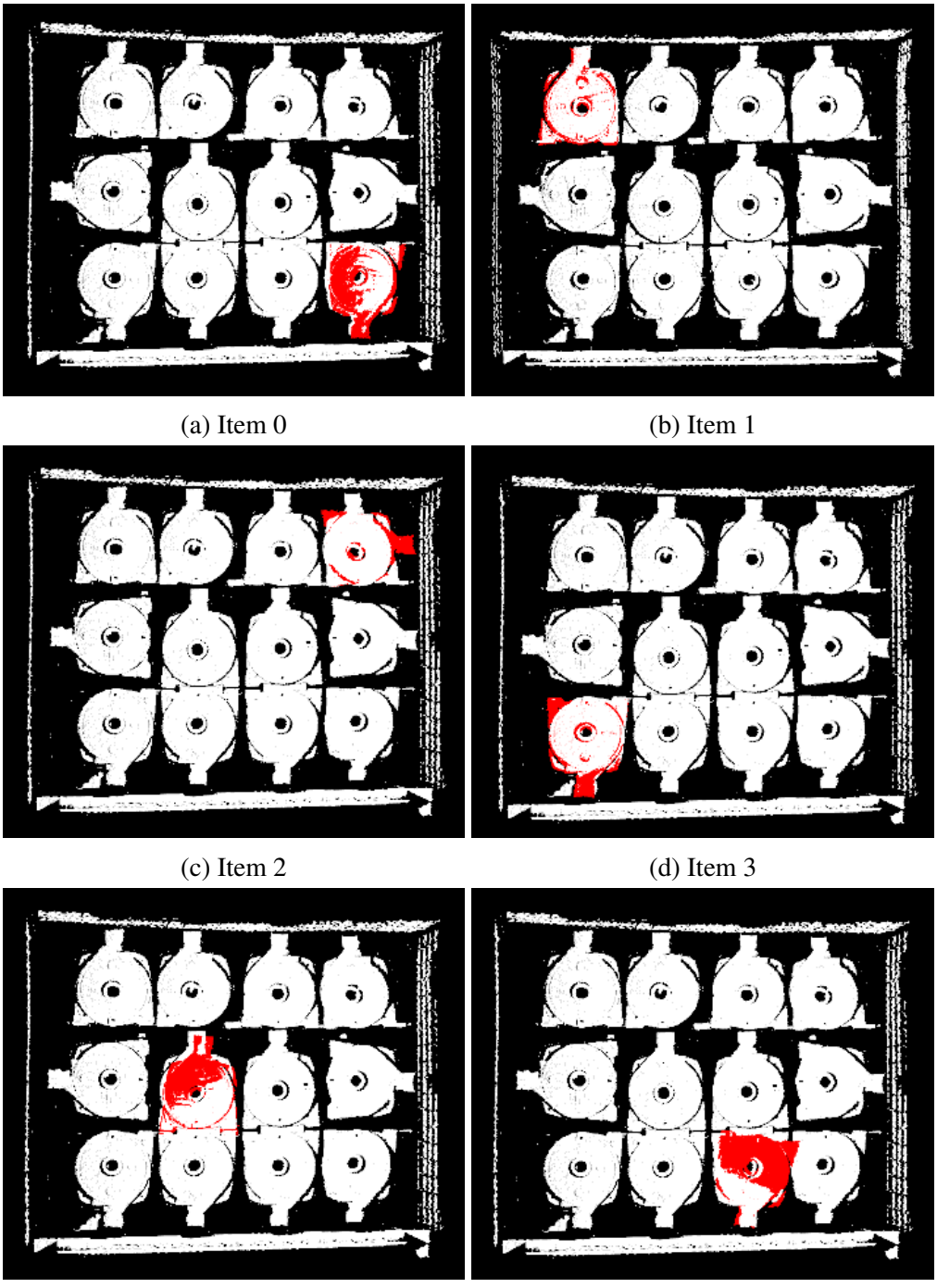
Table 5.5: Set-up for pump component recognition



(a) Pump component model (b) Scene of recognition

Figure 5.7: Model and target used in pump component recognition test

align the point cloud. Items 2 and 7 are examples of wrong object direction computation. These results are consistent with score values: the first groups show a match between more than 40% of keypoints, the second in a range of 33% and 40%, the last less than 33%. It means that we can set a threshold to select only the elements with an high accuracy and pick these: successively we can acquire another scan and re-execute the procedure to recognize also the others. Indeed the errors can be caused by scan defects that compromise RANSAC output. In some cases it happens that, for the presence of other points that represent noise, lines and circles with more inliers are positioned too differently than the model so our algorithm cannot perform matches.

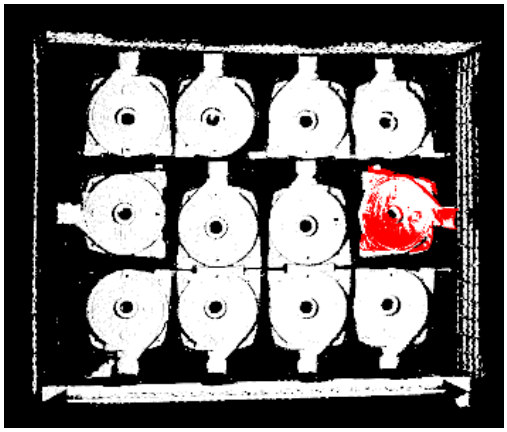


(a) Item 0 (b) Item 1

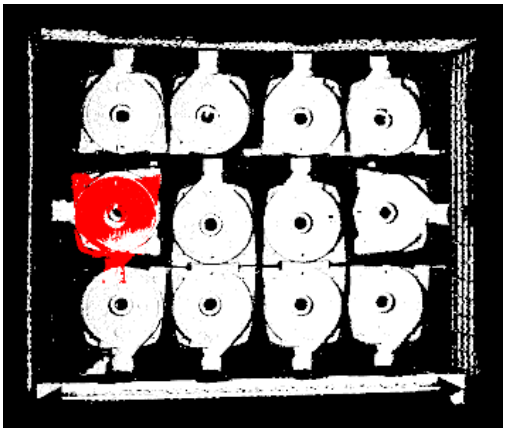
(c) Item 2 (d) Item 3

(e) Item 4 (f) Item 5

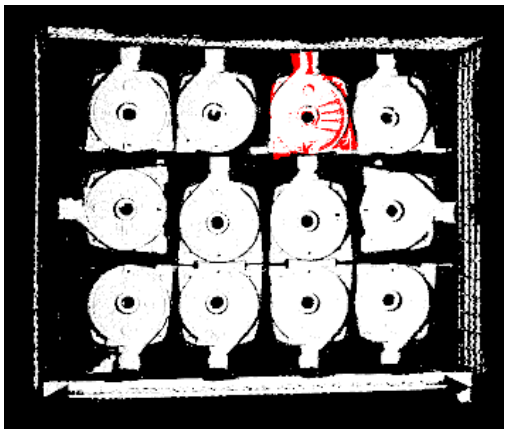
Figure 5.8: Results of pump component recognition test [first part]



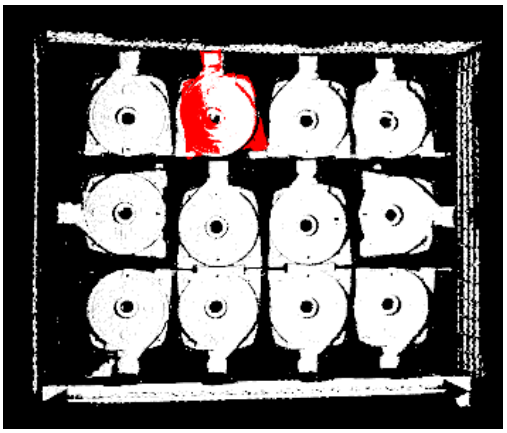
(g) Item 6



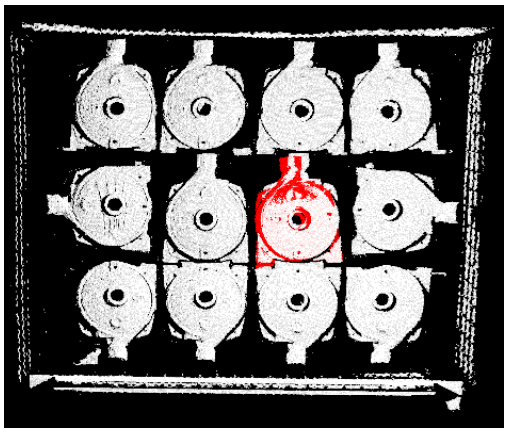
(h) Item 7



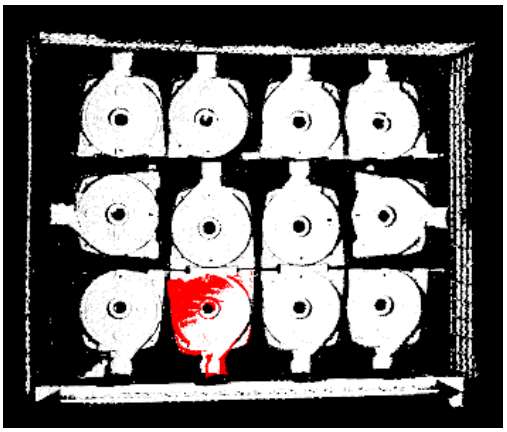
(i) Item 8



(j) Item 9



(k) Item 10



(l) Item 11

Figure 5.8: Results of pump component recognition test [second part]

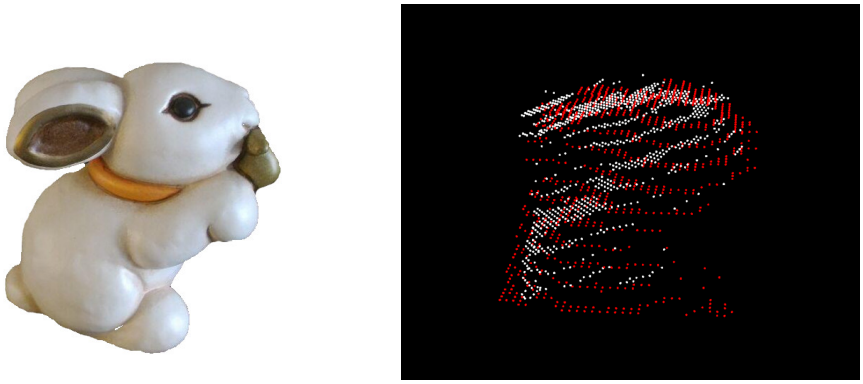
5.4 Tests with Microsoft Kinect v1

To give more completeness to our work we try to use our algorithm with other sensors. The selected alternative is Microsoft Kinect v1 because it is a largely used camera and it was easily available. In addition it is an interesting product for its low price and medium resolution.

The goal of these recognitions is to show the independence of the system to high resolute scans and its applicability with other sensors, different than Photoneo. Obviously the scanning quality has effects on results precision and it is important to choose correctly the sensor according to the task characteristics. We also take the chance to underline the versatility by using other object types as model and to test the recognition of deformable subjects: as a rigid model, we study the algorithm behaviour with an ornament shaped like a rabbit and as a non-rigid the one with pears.

Considering the parameter configuration, we use for both tests the values reported in Table 5.6. The only considerable changes comparing the set-up with Photoneo are about Ransac size and Neighbourhood position range: these are the most strictly linked to sensor resolution. The first represents a points number so if point cloud has less resolution it means that there are less points to describe the scene and consequently the inliers number of a particular mathematical model decreases. The second is a distance of neighbourhood centroid so if neighbours are farther also this parameter must increase: if it was not all points would be so far from the other to be considered in the edge. So these changes are necessary and quite intuitive.

In Figure 5.9 are reported the results of the rabbit ornament recognition. Model and target belong to different point clouds but are referred to the same



(a) Rabbit shaped ornament (b) Recognition result

Figure 5.9: Recognition of a rabbit shaped ornament

Parameter	Value	Parameter	Value
Ransac size	0	Matching length	60 mm
Ransac threshold	5 mm	Matching radius size	60 mm
Neig. position range	3 mm	Matching position	10 mm
Score	0.1	Matching orientation	0.25 rad

Table 5.6: Set-up for Kinect recognition

specimen in distinct places. As we can see the recognition produces good results: both position and orientation are computed correctly and the model overlaps the target. This precision can be verified for the rabbit ears and nose where the shapes are more clearly identifiable with human eyes.

The pear recognition is more interesting for the analysis of the system behaviour in presence of deformations. To remark diversities of every sample we report in Figure 5.10 targets used. In this case the fruit is not elongated or stretched and we not mechanically cause deformities: as deformation we mean that it does not exist a pear that is identically equal to another. These natural discrepancies can be considered as shape changes from the software point of view. This test means that our algorithm adapts well to the recognition of objects that are similar but not perfectly equal to the model.

In Figures 5.11 and 5.12 we report two results obtained analysing two different scenes: one with a pear randomly positioned on a table and one with targets well placed in a stand up position. It is clear that model does not overlap the elements in all its points but every object is not perfectly equal to the others therefore the opposite would be impossible. What is important is that the system matches features correspondent but not identical and computes correctly the position and orientation. Red points are sufficiently close to the white ones to try to perform grasping. With this recognition we can also present the partial nosensibility to distinct point of view between model and target. Partially because if a target shows a side about which we do not have any information our system will not recognize it.

The model is extracted from the first scene and used equally in the first and second scan.

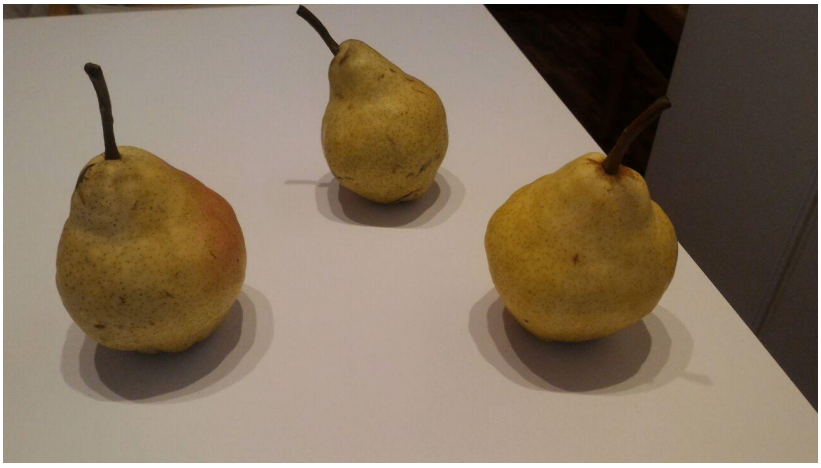


Figure 5.10: Scene of pears recognition

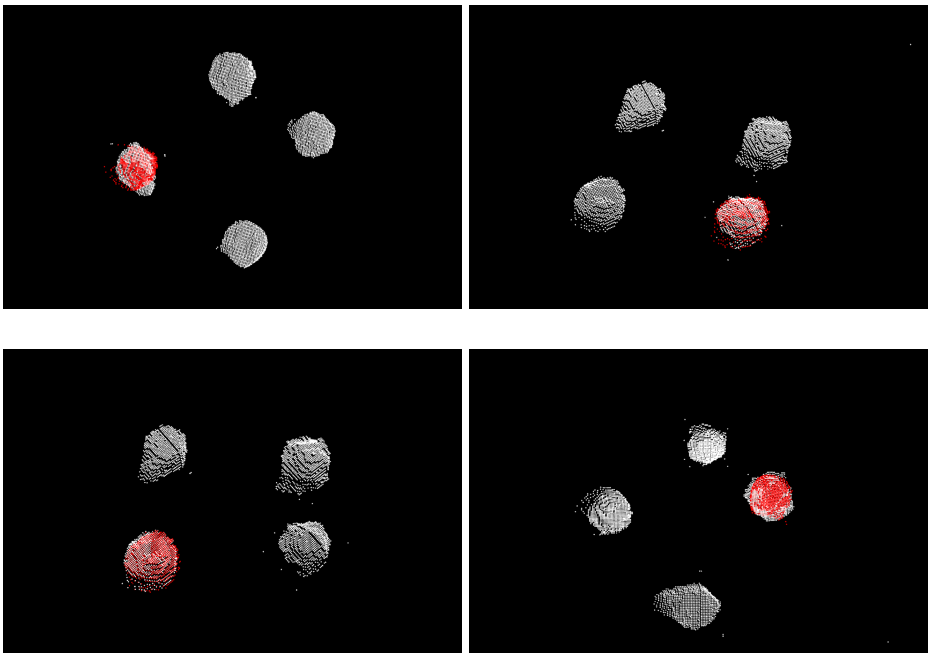


Figure 5.11: Recognition of pears (first scene)

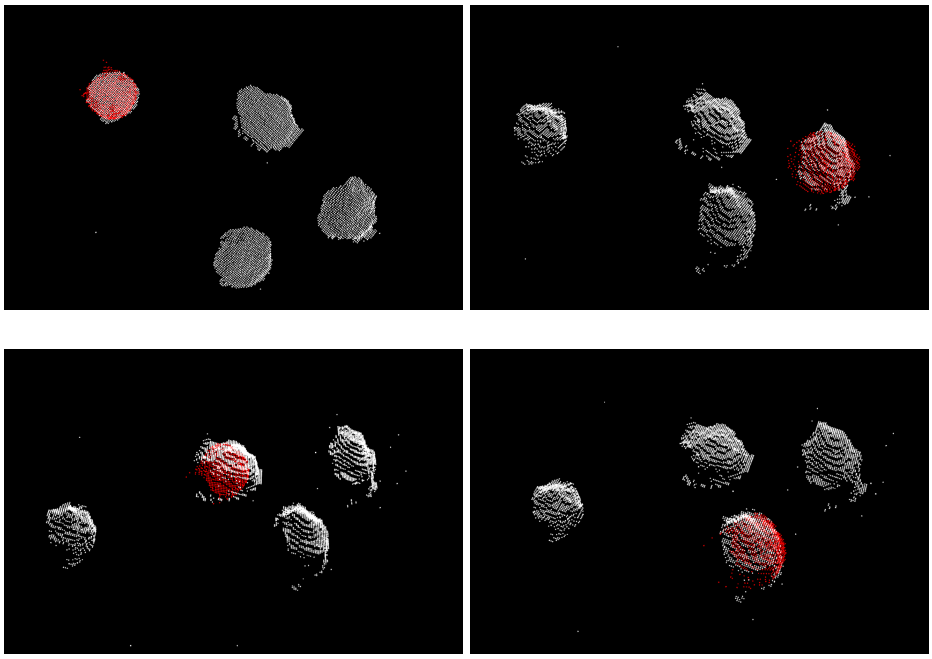


Figure 5.12: Recognition of pears (second scene)

5.5 Measures of precision

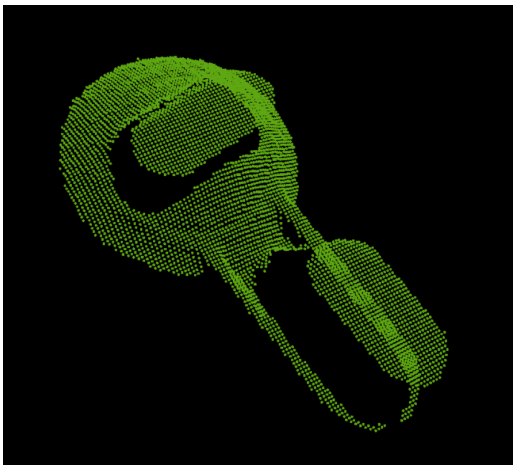


Figure 5.13: Coat rack model

This section is about quantitative tests to analyse our system precision in different situations and tasks. While the other trials are presented to only show qualitative performance changing object model and adding noise or clutters, this one gives precise information of error measures about the reccognition of both position and orientation, repeatability and tolerance to the displacement.

For this test we use two instruments that we have just presented in Sections 4.2 and 4.3: Photoneo to acquire scans and Baces Arm to take measures. An issue is represented by the fact that these two products return data referring to their own reference system: to allow their cooperation is necessary to calibrate one respect to the other. Our choice is to calibrate Baces respect Photoneo. To do this we scan the working area marking with a black marker four points on the environment. It is better if these points are not coplanar, therefore three of them belong to a table plane and the last to a cardboard box positioned on the table. Using *PCL visualizer* we extract them and compute homogeneous matrix to translate Photoneo coordinates values into Baces ones. To achieve correctly all the procedure we must to consider also the sphere diameter of the arm tip: this inserts an approximation that we try to reduce as much as possible. During measures acquisition we position the tip in a perpendicular direction to x-y plane and we subtract 2 units to the final z value[1].

All the following recognitions are realized given as input a little coat rack with a suction cup to apply it on a bath wall. The object model is seen in Figure

[1] remember that sphere is 4 mm of diameter and that the measure represents its centre position

(a) Template (b) Template application on coat rack

Figure 5.14: Template and its use

5.13. We select this subject to remark the versatility of our system and to have more agility during test: for its size and lightness it can be easily handled and moved around the environment. In addition, the suction cup is used to fix objects on the table: in this way it will not change its position during scans acquisition.

It is important also to mark two points on coat racks to measure precisely displacements and errors. These marks must have two basic characteristics: they have to represent the same relative position in the object and they must be easily visible in scans. To solve these problems we realize a wood template with two holes used to apply two adhesive felts. This mould is handmade milling wood and reproducing the exactly shape and size of the coat rack: to prove template precision we apply it on the subject and ascertain that all is blocked and nothing can move. In Figure 5.14 we report mould and its application.

We perform three tests:

1. we scan the same scene with the object in a fixed place several times to analyse precision in position and orientation estimation and repeatability;

2. we acquire several scans in the same environment moving the object to 3 known poses to see sensibility to position changes and to measure precision in displacement computation;

3. we perform recognition to bin-picking placing randomly three coat racks on a table.

Parameters configurations can be seen in Tables 5.7 and 5.8. We use two settings because one has more relaxed constraints and can be used in several

Parameter	Value	Paramter	Value
Ransac size	10	Matching length	10 mm
Ransac threshold	2 mm	Maching radius size	10 mm
Neig. position range	0.5 mm	Matching position	5 mm
Score	0.1	Matching orientation	0.25 rad

Table 5.7: Set-up for coat rack recognition (configuration 0)

Parameter	Value	Paramter	Value
Ransac size	10	Matching length	10 mm
Ransac threshold	2 mm	Maching radius size	10 mm
Neig. position range	0.5 mm	Matching position	2 mm
Score	0.1	Matching orientation	0.25 rad

Table 5.8: Set-up for coat rack recognition (configuration 1)

different contexts, the other reaches the greatest precision and shows the best system behaviour. The first two tests are made with both the configurations; the latter is executed only with less strict constraints to allow the system to be more robust to noise and position changes.

In the following subsections we are going to present the details of every quantitative test.

Position and orientation precision

In this test we block one coat rack in a position and acquire 22 scans of the same scene. We extract from the first 3D image the object model and we execute the recognition on all the others. To measure the precision of the computed object position and orientation we consider the two points marked with the adhesive felts described above. We save their position manually in the model exploited *PCL visualizer* and then we use the transformation computed by the system to find their position in the scene. The next step is to apply an homogeneous matrix to the result to translate the values in Baces arm coordinate system. Comparing these final positions with the ones measured by Baces we can compute the error entity. To evaluate also the orientations we must to consider that scans are acquired with the object positioned on a table so z value might not change from a point to the other and we can simplify the problem in a 2D space. We compute the line that links both marked points and its grade in x-y plane. Comparing final values obtained with found and acquired position we can also

evaluate orientation error.

The goal is to compute the precision of our algorithm regarding the object position and orientation estimation. The repetitions of the procedure for 22 scans helps to remove systematic errors in the measurements. Considering only a single object without changing its place in the environment we simplify the task to do a first test of precision; in the others also this simplification will be overtaken. Another thing that is evaluated in this phase is the sensitivity to scan quality because we use as input data all the acquisitions without even discarding one.

To be clearer and more precise in the results explanation, we add some graphs that show different peculiarities of our experiments.

In Figure 5.15 we represent the output of the recognition task: with light colours we indicate computed positions of two grip points (that are the points selected with *PCL visualizer*), red for the first and blue for the second, and, with the correspondent dark tonalities, real point position acquired by Baces. Here, we notice that in all the trials the object is well identified both in position and orientation. The errors are almost always lower than 2 mm, except in one case where the distance with the real position reachces 7 mm. In addition, our system estimates better grip point 1 than 2: this is an effect of the orientation computation. The first one is closer to the object centre therefore it suffers less direction error repercussions. In order to better analyse the inaccuracies and their distribution we report two other graphs shown in Figure 5.16. The upper function shows the times that a certain error is registered and the other its distribution with the average for both grip point 1 and 2. An important characteristic is that the largest part of the differences respect to real positions is grouped very

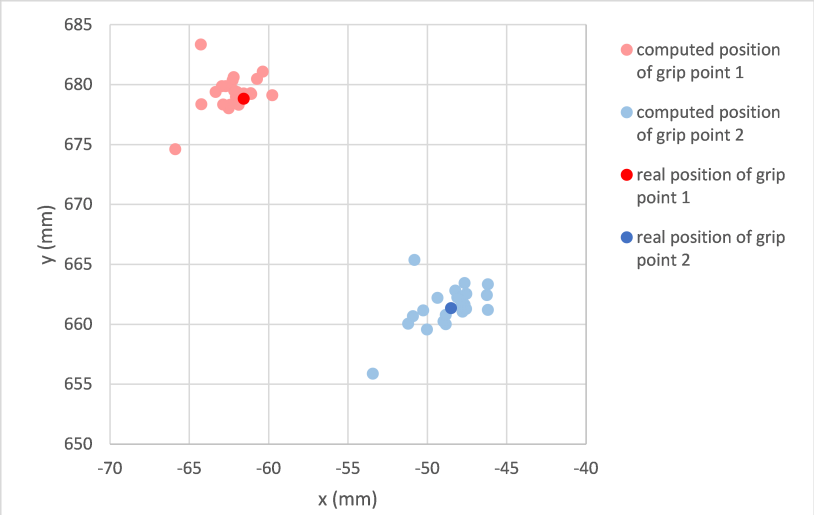


Figure 5.15: Positions computed with configuration 0

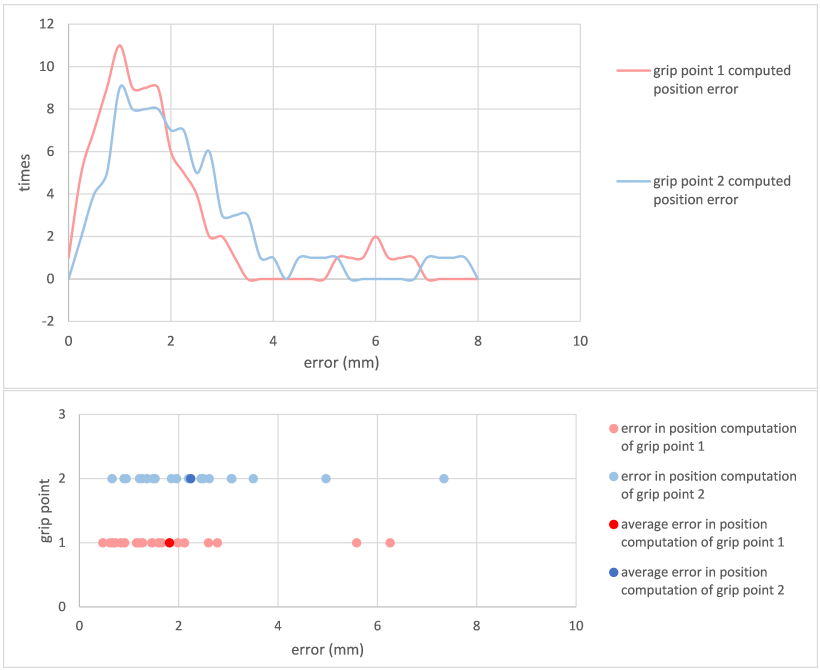


Figure 5.16: Error distribution in computed position with configuration 0

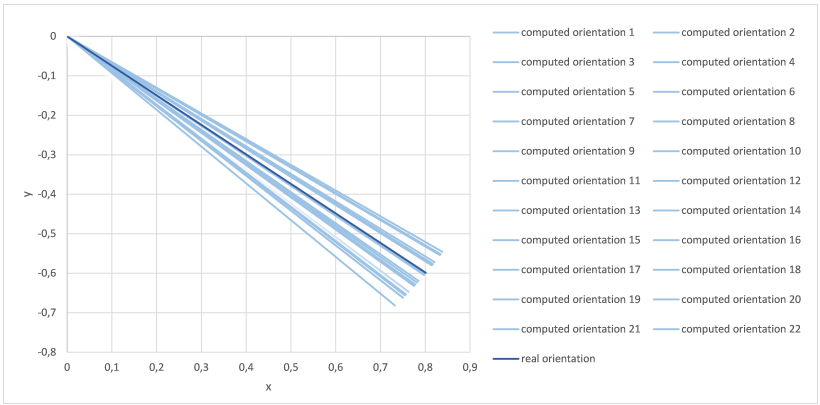


Figure 5.17: Orientations computed with configuration 0

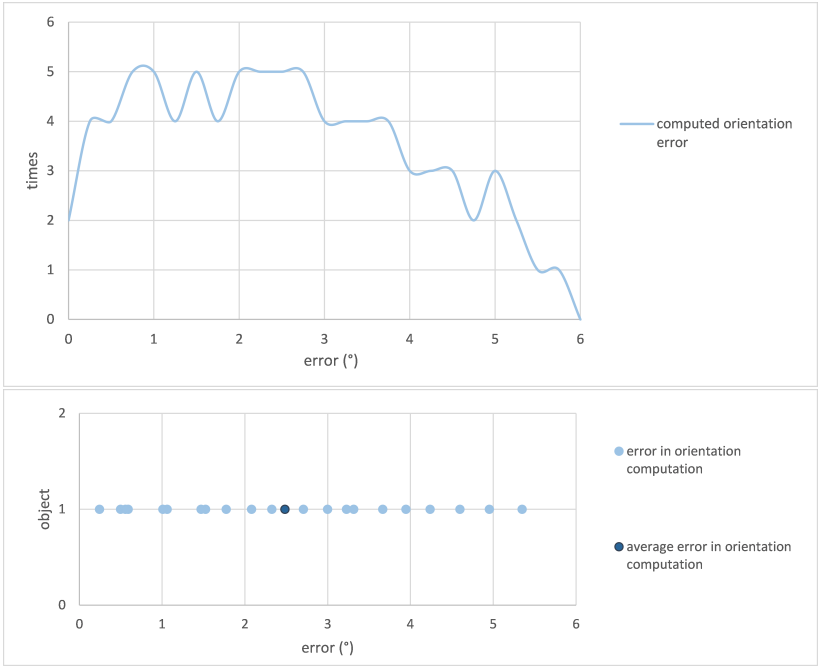


Figure 5.18: Error distribution in computed orientation with configuration 0

close to the average. This allows to recognize a sort of a Gaussian distribution, that is clearer in first representation with the bell. In this case we cannot see it perfectly for the low number of repetitions but it is quite identifiable. This has an important consequence: it means that our algorithm returns with a high probability a position close to the average.

Considering all the errors independently for grip point 1 and 2 we compute an average of 1.82 mm for the first and 2.09 mm for the second. The correspondent variance is 2.24 mm^2 and 2.28 mm^2. This is not a great precision but the quality of a system is always linked to its application context. We can observe that our algorithm cannot be used for too small objects but it has other advantages that we have just explained in other results.

We have to make different considerations about orientation errors. The same graphs as for positions are shown: Figure 5.17 represents computed object direction for each scan analysis and Figure 5.18 error distribution. We observe that all the outputs are concentrated near the real value with no evident outliers. Function development recalls a Gaussian bell even though there is not an evident peak at average value that is 2.48 °. The variance is 2.27 $°^2$ and the maximum registered error is 5.34 °. As a final note, we can repeat that there is no perfect recognition but it is enough in contexts with not too small objects.

All these measurements are acquired with parameters configuration of Ta-

ble 5.7. As we have just said we try to see the best behaviour of our system re-executing the test with more strict constraints. Graphs in Figure 5.19,5.20, 5.21 and 5.22 are referred to settings in Table 5.8. For the largest part of results, positions are computed correctly with a higher precision (see Figure 5.19): this is an obvious consequence to the fact that the algorithm searches compatible keypoints in a smaller area near the predicted place than the previous case. Observing distribution error in Figure 5.20 we can easily identify Gauss bell for the first grip point. Indeed, in the second there is a great global maximum but the presence of another local peak at 3 mm gives to the function a strange shape. It means that there are some outliers all concentrated in a small error range. Anyway, they are only four outliers in over 22 different elements.

Bells are narrower than in the previous case (Figure 5.16) and this is in agreement with the computed variances: 0.37 mm^2 for point 1 and 1.52 mm^2 for point 2. Also the averages show an improvement: 1.23 mm for the first and 1.66 mm for the second.

Orientation estimation is penalized by the four outliers in grip point 2 position. Indeed, we can notice in the error distribution (Figure 5.22) two first peaks at the beginning and then some lower ones that extend the bell base of over 6 °. All these observations are coherent with the computed data: even though the average improves reducing at 2.22 °, the variance reaches 2.60 $^{\circ^2}$.

The system behaviour concerning the orientation estimation is the same for both the configurations. We cannot say the same for the position computation that increases of around 0.5 mm.

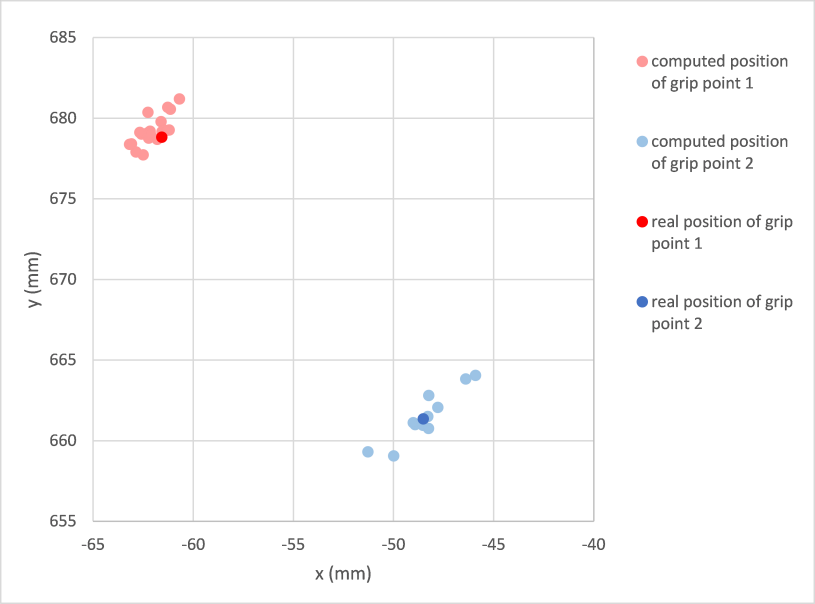


Figure 5.19: Positions computed with configuration 1

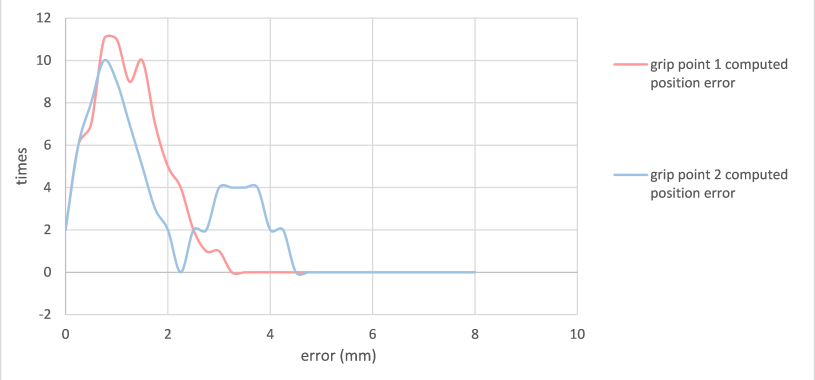


Figure 5.20: Error distribution with configuration 1

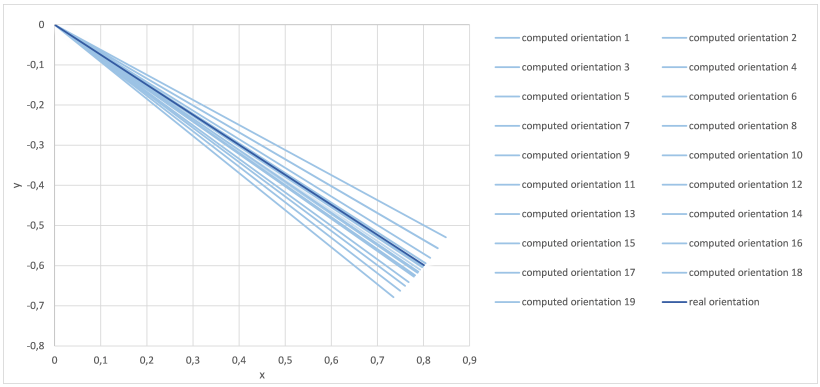


Figure 5.21: Orientations computed with configuration 1

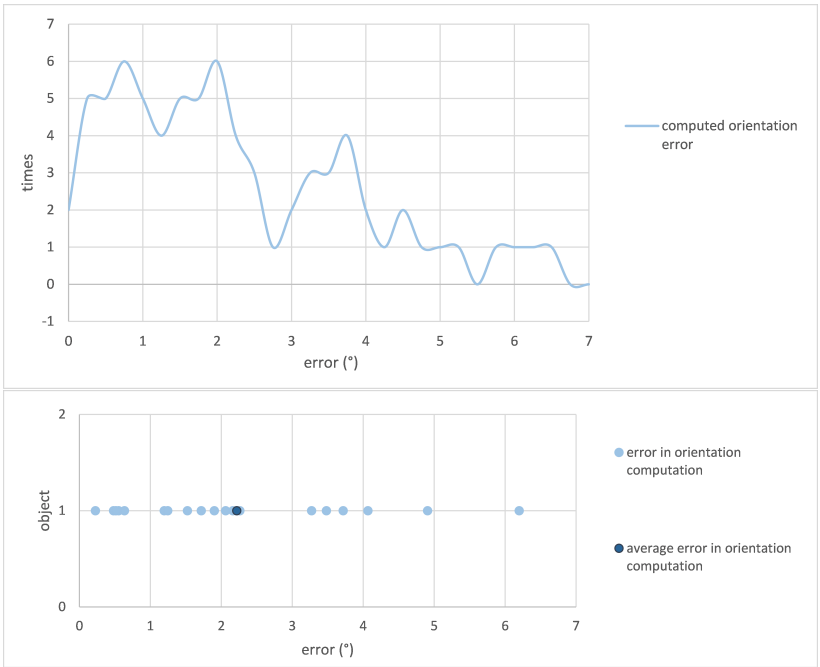


Figure 5.22: Error distribution in computed orientation with configuration 1

Tolerance to displacements

Here we are going to analyse the precision for relative positions and displacements estimation. Therefore we choose the same coat rack of the previous test as the object to recognize and we shift it into 3 different positions. The displacements are of known and precise quantities: 10 cm along x-axis, 10 cm along y-axis and 10 cm along both. To guarantee the correctness of the direction and the movement size, we use a graph paper positioned according to the Baces arm reference system. In this way, shifting the coat rack, for example only along the longest sheet margin, we register changes only in the y-axis value.

We acquire six scans in the first and second position and seven in the third. As a model we use the point cloud of the coat rack acquired before moving it. In this case we cannot use data of another context because the position is essential in order to correctly compute the displacements.

The test is executed in the following way: with our algorithm we estimate the positions of the two grip points and with Baces we measure their real one. Comparing the known displacements quantities with the ones computed subtracting the estimated object positions with the ones of the model, we measure our system precision.

We propose two graphs to explain the results. The first one (Figure 5.23) is

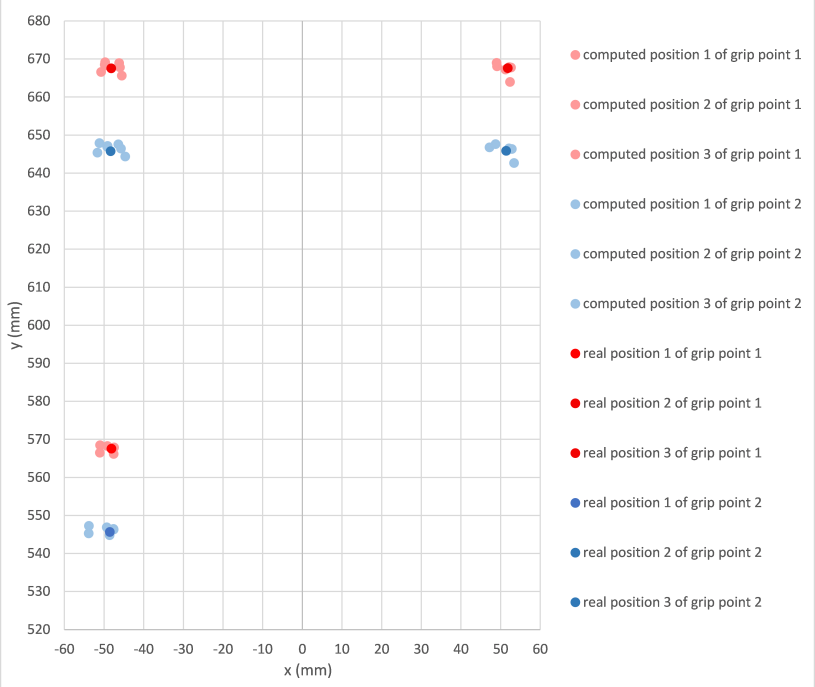


Figure 5.23: Positions computed with configuration 0

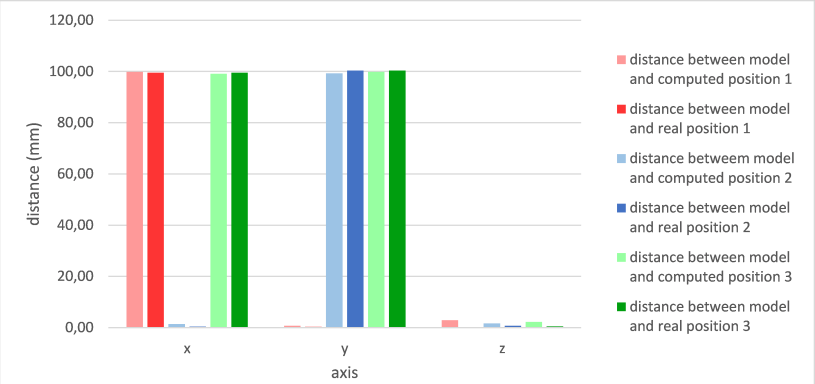


Figure 5.24: Displacement size computed with configuration 0

prone to similar observations of the previous test: it reports the positions computed with the procedure considering all displacements. In this case statistics are worse: average error for grip point 1 is of 2.5 mm and for grip point 2 of 3.66 mm with a variance respectively of 1.55 mm^2 and 4.35 mm^2.

Another aim of this graph is to contextualize the experiments and gives to the reader a clearer idea of what are the displacements and their directions. Model was placed in bottom-left corner.

The most interesting graph is the one in Figure 5.24 that shows the displacements compute by our algorithm: red for the first position, blue for the second and green for the third. The operations solved to obtain presented data are shown in the following equations:

$$displacement_x = \frac{1}{|S|} \sum_{s \in S} |b_x - a_x|$$

$$displacement_y = \frac{1}{|S|} \sum_{s \in S} |b_y - a_y|$$

$$displacement_z = \frac{1}{|S|} \sum_{s \in S} |b_z - a_z|$$

where S is the set of scans, b is grip point 1 position measured by Baces and a is the one computed by our algorithm. We decide to split measurements in each component value to underline the precision reached. As we can see by vertical bars, our computations is completely in accordance with the real one. The greatest differences between Baces and our system computations are registered along z axis and are about 1.7 or 2.7 mm. In detail they are:

- in position 1 about 0.32 mm on x, 0.39 mm on y and 2.74 mm on z;

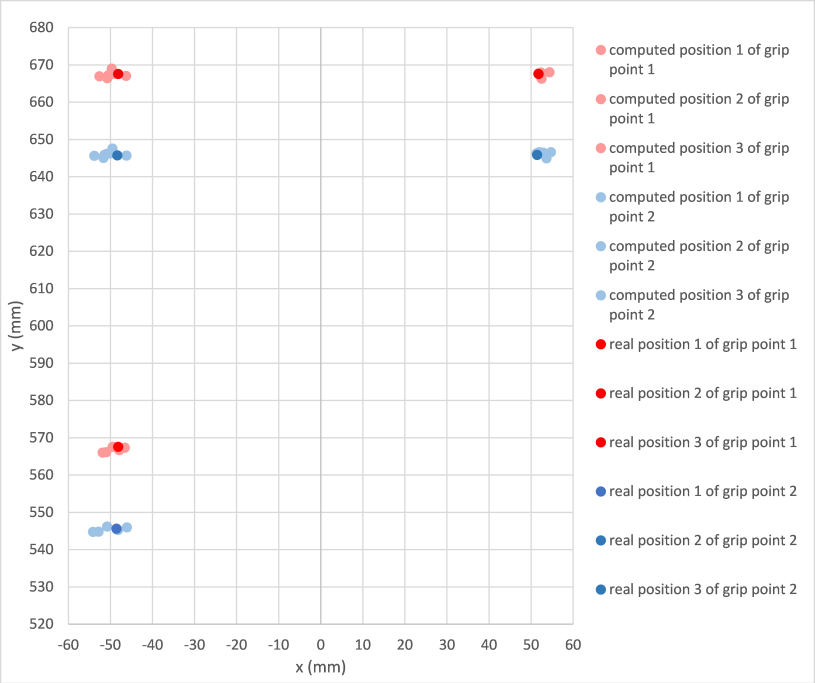


Figure 5.25: Positions computed with configuration 1

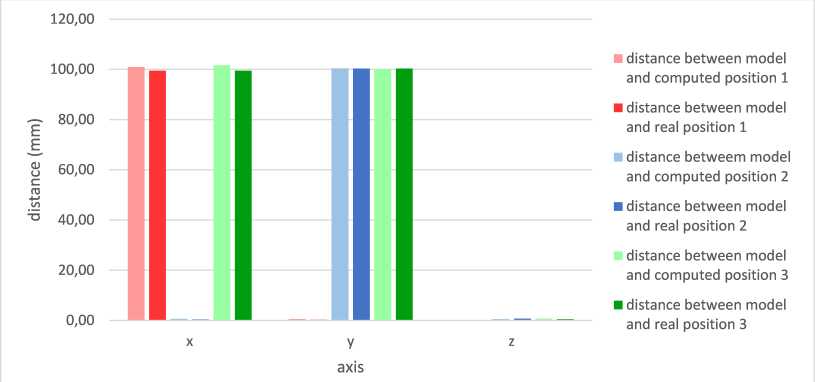


Figure 5.26: Displacement size computed with configuration 1

- in position 2 about 1.02 mm on x, 1.08 mm on y and 0.88 mm on z;
- in position 3 about 0.44 mm on x, 0.52 mm on y and 1.68 mm on z.

All these data are referred to the results obtained using parameters configuration of Table 5.7; trying to increase precision we use second settings (Table 5.8). We can see improvements just in position graphics (Figure 5.25) where all points are close each other and to the real positions. There are no outliers and

the averages are lower: 1.74 mm and 2.56 mm. Variance decreases only for the second point to 2.58 mm^2 while for the first it remains stable around 1.6 mm^2.

Observing displacements size (Figure 5.26) the improvement is immediately clear: all the errors around zero values are deeply reduced or disappeared. This is remarked also by the data that show differences of about 1 or 2 mm in only one case: the others are all lower than 0.3 mm. In detail we find errors:

- in position 1 about 1.4 mm on x, 0.23 mm on y and 0.03 mm on z;
- in position 2 about 0.3 mm on x, 0.02 mm on y and 0.18 mm on z;
- in position 3 about 2.08 mm on x, 0.14 mm on y and 0.21 mm on z.

The presented results underline that the system registers with great precision the position changes and can measure well relative distances between objects in the scene. About the estimation of absolute subject place they do not add any information to our previous knowledge.

Bin picking precision

The last test is about bin picking. To simulate this task, we place randomly three coat racks of the same type on a table. Computing the two marked points position in the recognized targets we can compare these poses with the ones measured by Baces and find the error value. We perform the recognition using the model acquired in the first of these three tests: in this way we evaluate our algorithm giving as input a subject acquired in a different moment. Another complication in addition is the use of a scene composed by more than one target randomly positioned (it includes also orientation changes). Together these difficulties reduce the gap with real applications.

We repeat this procedure with two different object dispositions and for every configuration we acquire 7 scans.

Figures 5.27, 5.29, 5.28 and 5.30 represent results obtained in this test and are organized as follow: the first two show grip points computed and real positions, the last two evaluate identified object orientations compared with the real ones. Graphs are two for each measurement type because we evaluate two different targets dispositions.

Starting by the position analysis we can confirm what we have just explained in the other tests: to be more clear we report two tables that list single and total averages and variances (Tables 5.9 and 5.10). We note that grip point 1 position is always more precise than grip point 2 that also registered a greater variance. The total statistics are in perfect agreement with data shown in Subsection 5.5.

Grip point 1				Grip point 2			
mm	object 1	object 2	object 3	mm	object 1	object 2	object 3
scene 1	1.97	1.91	2.77	scene 1	3.1	3.29	4.29
scene 2	1.97	2.5	1.59	scene 2	3.2	4.13	2.7
total	2.02			total	3.16		

Table 5.9: Averages of position error

Grip point 1				Grip point 2			
mm^2	object 1	object 2	object 3	mm^2	object 1	object 2	object 3
scene 1	0.96	1.42	1.63	scene 1	4.78	1.422.83	5.57
scene 2	1.71	1.1	0.83	scene 2	5.73	4.57	3.54
total	1.37			total	4.22		

Table 5.10: Variances of position error

Analysing the orientations we have some good estimations accompanied by others less accurate. In Figure 5.28 that describes first disposition, we find two objects that are well recognized also in their directions: the blue real measure is perfectly contained in a group of very close computed orientations except for an outlier; the green one is the same but with a set of outliers with a bigger cardinality. Red measurements are not as good as the other because the real value is not in the middle of a group of samples: it seems to be an outlier. Similar considerations can be done for the second dispositions: green elements in Figure 5.30 are close to real value; in red region Baces measurement is an inlier of the group but there are not so many neighbours; the blue case is the same of the red elements in the first disposition.

All these qualitative observations are in agreement with extracted statistics. In the first disposition the worst orientation computation is of object 1 (colour red in the image) with an average error of 4.8 °. The best is object 2 (blue) with an average error of 2.7 ° and the last is object 3 (green) with 4 °. In the second displacement the precision is higher but there is not so much difference concerning general behaviour. The worst case is object 2 with an average error of 4.8 °; the best is object 3 with 1.5 °; in the middle there is object 1 with 3.4 °.

The average of all errors is 3.54 °.

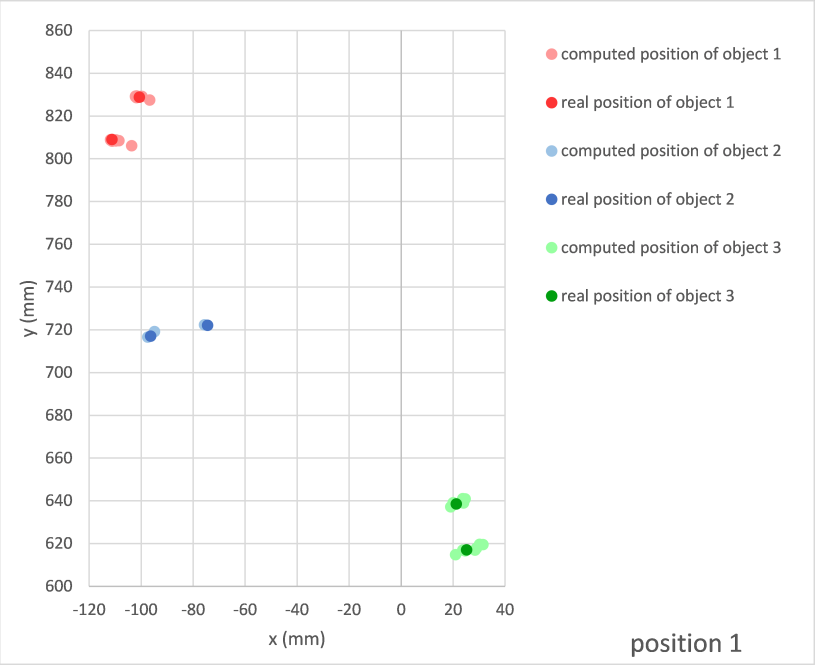


Figure 5.27: Positions computed in disposition 0

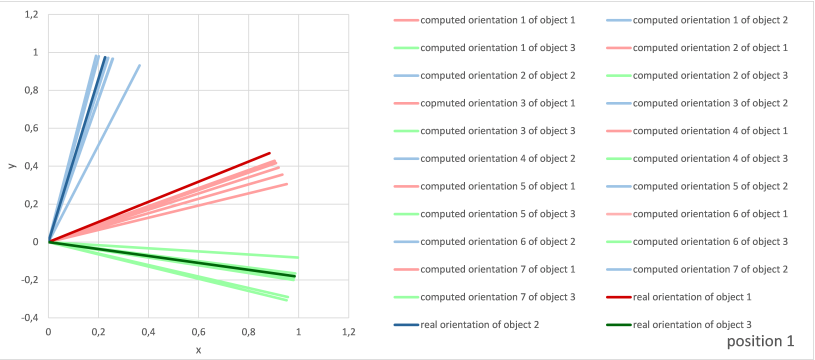


Figure 5.28: Orientations computed in disposition 0

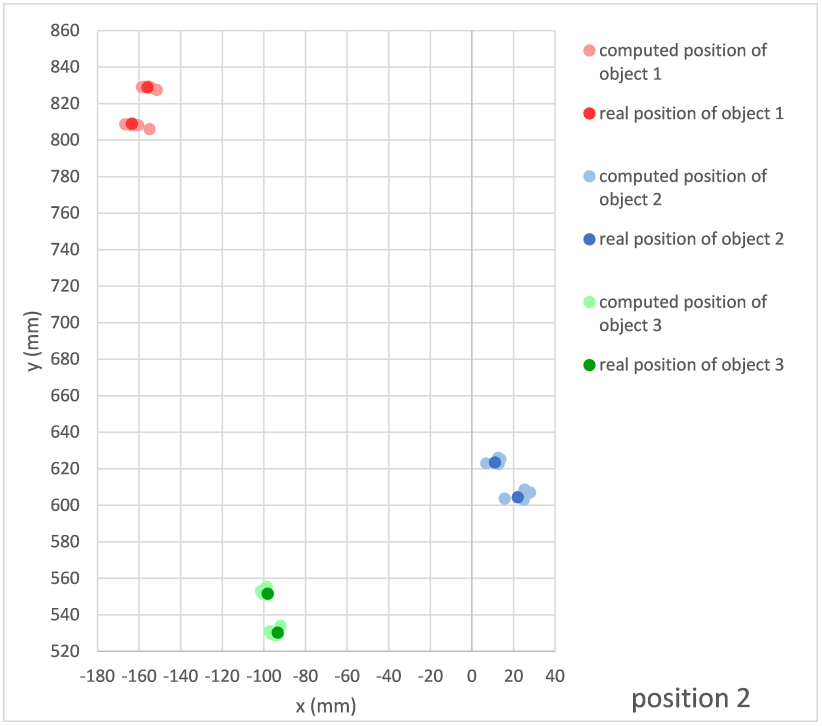


Figure 5.29: Positions computed in disposition 1

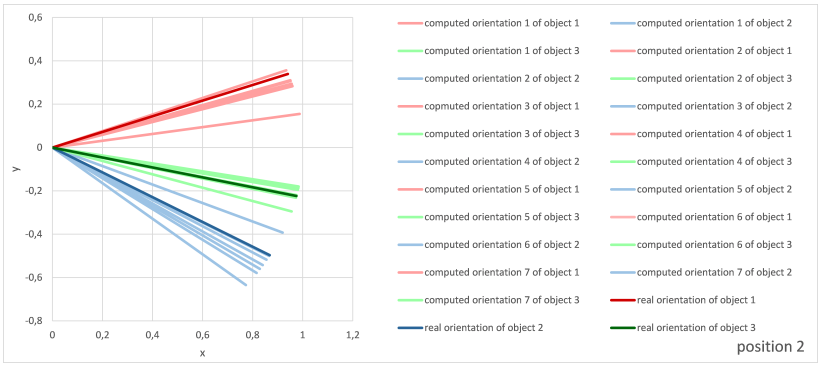


Figure 5.30: Orientations computed in disposition 1

Chapter 6

Conclusions

With this work we introduce a new approach for 3D object recognition. Its peculiarity is that we use as input an object point cloud as model and not its CAD model. This includes noise management and less precise information: for this reason the presented method needs of several steps that must be executed on cascade. Starting with box floor removal, we analyse the scene to find objects number, then we cluster the scene to obtain a point cloud for each of them. Further steps are executed for every cluster and consist in selection of points that represent normals gradient peaks or edges, lines and circles fitting among them and their reduction to the centre and finally the description of these last features. The recognition ends with feature matching and the choice of the best computed transformation to overlap model on the target.

We use two 3D cameras to acquire scans, both based on structured light: Phoxi Photoneo and Microsoft Kinect v1. Thanks to the presence of a projector and a camera that recognizes the pattern emitted, this technology allows its use also in dark environments and it is not sensible to brightness changes.

Using only shape information we realize a system that can work in different contexts. Its results quality does not depend on several changes such as light or colour intensity and object texture.

Our tests show that this approach is very versatile since we recognized several objects with different characteristics. We make trials with square supports, ornaments shaped like robot and rabbit, jar, cups, pump components, pears and coat racks: they have different dimensions and shapes, some more complex than the others. Cups were of two not equal types and they were acquired with a lot of undercuts that played the role of clutters. Pump components are symmetrical adding difficulties to the orientation estimation. Pears are natural products and their model does not exist: every sample has dissimilarities respect the others. Coat racks have different colours and are smaller that the other objects.

We perform our tests in several contexts: some are executed in ideal environ-

ment to check correctness of only specific algorithm parts, some other in lab and one represents a real industrial problem. The results prove insensibility to partial clutters and small objects deformations in addition of object type, dimension and texture. The symmetries generate little inaccuracies in orientation estimation but they are overtaken with successively scan and execution repetitions.

Precision measurements reveal that our system can reach an error in position estimation of 1.23 mm but the average is around 2 mm. Concerning orientation the average error is around 3,5 ° with a best case of 2,2 °. These results describe correct recognition but not so precise to manage very small objects or to work in particular contexts that require high accuracy. This does not allow its application to solve all industrial problems. Despite this inaccuracy in absolute positions estimation, we find great precision computing displacements entity. Our tests measure errors lower than 0.5 mm.

Future improvements will be focused on execution time and precision.

We could increase speed adding parallel code in Cuda exploiting that in the algorithm there are few specific operations that are run for each point of the scan. A procedure designed like this is suited to parallelization. Also clustering could be improved in execution time: instead of use Farthest First Traversal to find starting sequence of centres we could design an opportune distribution function to bring points scanning the data structure only once, inspiring by K-means++. This would be more efficient also in presence of many elements to cluster.

The introduction of an alignment algorithm, like Iterative Closest Point or others, could improve precision and reach higher level of accuracy also with symmetric and small objects. The procedure choice will be done considering trade-off between performance in execution time and results quality. We could also exploit the search of 3D mathematical models in second level keypoint extraction to describe better some particular object shapes. In addition introducing cooperation between *Normals gradient analyis* and *Neighbourhood analysis* we could acquire information in case of both too regular and complex objects.

Adding these advances we will expand the applicability of this project in many other industrial contexts.

References

[1] S. Wold. Spline functions in data analysis. *Technometrics*, 16(1):1–11, 1974.

[2] D. Terzopoulos and D. Metaxas. Dynamic 3d models with local and global deformations: deformable superquadrics. pages 606–615, Dec 1990.

[3] Wikipedia. Superquadrics — wikipedia, the free encyclopedia, 2017. `https://en.wikipedia.org/w/index.php?title=Superquadrics&oldid=770878683`; [Online; accessed 26-August-2017].

[4] C. M. Cyr and B. B. Kimia. 3d object recognition using shape similiarity-based aspect graph. 1:254–261 vol.1, 2001.

[5] Y. Li, C. F. Chen, and P. K. Allen. Recognition of deformable object category and pose. pages 5558–5564, May 2014.

[6] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, Sept 2010.

[7] M. Pedersoli, A. Vedaldi, J. Gonzàlez, and X. Roca. A coarse-to-fine approach for fast deformable object detection. *Pattern Recognition*, 48(5):1844 – 1853, 2015.

[8] Ce Liu, Jenny Yuen, Antonio Torralba, Josef Sivic, and William T. Freeman. *SIFT Flow: Dense Correspondence across Different Scenes*, pages 28–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[9] C. Liu, J. Yuen, and A. Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):978–994, May 2011.

[10] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[11] H. Tehrani Niknejad, A. Takeuchi, S. Mita, and D. McAllester. On-road multivehicle tracking using deformable object model and particle filter with improved likelihood estimation. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):748–758, June 2012.

[12] B. Kim, S. Xu, and S. Savarese. Accurate localization of 3d objects from rgb-d data using segmentation hypotheses. June 2013.

[13] A. K. Jain, Yu Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):267–278, Mar 1996.

[14] A. C. Berg, T. L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. 1:26–33 vol. 1, 2005.

[15] S. Belongie, J. Malik, and J. Puzicha. Matching shapes. 1:454–461 vol.1, 2001.

[16] A. C. Berg and J. Malik. Geometric blur for template matching. 1:I–607–I–614 vol.1, 2001.

[17] M. Cho, Jungmin Lee, and K. M. Lee. Feature correspondence and deformable object matching via agglomerative correspondence clustering. pages 1280–1287, Sept 2009.

[18] Vittorio Ferrari, Tinne Tuytelaars, and Luc Van Gool. *Simultaneous Object Recognition and Segmentation by Image Exploration*, pages 40–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[19] J. Kannala, E. Rahtu, S. S. Brandt, and J. Heikkila. Object recognition and segmentation by non-rigid quasi-dense matching. pages 1–8, June 2008.

[20] Minsu Cho, Young Min Shin, and Kyoung Mu Lee. *Co-recognition of Image Pairs by Data-Driven Monte Carlo Image Exploration*, pages 144–157. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[21] J Matas, O Chum, M Urban, and T Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761 – 767, 2004. British Machine Vision Computing 2002.

[22] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, Oct 2004.

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

[24] A. Nuchter, K. Lingemann, and J. Hertzberg. Cached k-d tree search for icp algorithms. pages 419–426, Aug 2007.

[25] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[26] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Classification: basic concepts, decision trees, and model evaluation. *Introduction to data mining*, 1:487–568, 2006.

[27] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[28] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. *Proc. of ACM-SIAM SODA*, pages 1027–1035, 2007.

[29] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, March 2012.

[30] Wikipedia. Rotation matrix — wikipedia, the free encyclopedia, 2017. "https://en.wikipedia.org/w/index.php?title=Rotation_matrix&oldid=794324319"; [Online; accessed 24-August-2017].

[31] Pcl home page, August 2017. http://pointclouds.org/.

[32] P.H.S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138 – 156, 2000.

[33] O. Chum and J. Matas. Matching with prosac - progressive sample consensus. 1:220–226 vol. 1, June 2005.

[34] Zhengyou Zhang, Rachid Deriche, Olivier Faugeras, and Quang-Tuan Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, 78(1):87 – 119, 1995. Special Volume on Computer Vision.

[35] Tinne Tuytelaars and Luc Van Gool. Wide baseline stereo matching based on local, affinely invariant regions.

[36] Photoneo. Photoneo focused on 3d. http://www.photoneo.com/; [Online; accessed 25-August-2017].

[37] Wikipedia. Kinect — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kinect&oldid=788947752; [Online; accessed 25-August-2017].

[38] Wikipedia. Structured-light 3d scanner — wikipedia, the free encyclopedia, 2017. https://en.wikipedia.org/w/index.php?title=Structured-light_3D_scanner&oldid=800713424; [Online; accessed 20-September-2017].

[39] Photoneo. Phoxi 3d scanner technology overview. http://www.photoneo.com/phoxi-3d-scanner/; [Online; accessed 25-August-2017].

[40] Photoneo. 3d scanning knowledge base photoneo wiki. http://wiki.photoneo.com; [Online; accessed 25-August-2017].

[41] C. Andujar. Kinect. *Universitat Politecnica de Catalunya, Barcelona Tech*; http://www.cs.upc.edu/~virtual/RVA/CourseSlides/Kinect.pdf; [Online; accessed 25-August-2017].

[42] Microsoft. Kinect fusion, 2017. https://msdn.microsoft.com/en-us/library/dn188670.aspx; [Online; accessed 17-September-2017].

[43] Kreon. Baces measuring arm, 2017. https://kreon3d.com/scanning-arms-portable-cmm/baces-measuring-arm-portable-cmm/; [Online; accessed 14-September-2017].