POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master of Science in Computer Engineering

Geometry alignment for collaborative robots

Supervisor:

Prof. Alessandro Rizzo

Company Tutor:

Dr. Robert Bohlin

Candidate:

Matteo Canavero

Ai miei nonni

Francesco, Giovanni, Lucia e Maria

Acknowledgements

First of all I want to thank you my thesis advisor, Prof. Alessandro Rizzo for his willingness, the interest shown in this project and his support.

I would also like to thank you my tutor at FCC, Dr. Robert Bohlin for assistance during the whole project and for all time and efforts spent in helping me. Without his guidance this thesis would not have been possible.

Thank you to all the employees at FCC, in particular to my thesis colleagues Christian, Hanna, Joakim and Fabian.

Thanks also to my family, my wonderful girlfriend Elena and all my friends for supporting me during these months of work and along all these years. Without you it would have been difficult to achieve this goal.

Abstract

In the last years the growth of the robotic and industrial automation market has been exponential and all the projections indicate that this market and, more in general, the level of automation in manufacturing industry will further increment in the near future. For this reason, the robot motion planning process is playing an increasingly important role in the production engineering field and therefore, software applications for robot path planning simulation and optimization are becoming essential in manufacturing.

One of the issues in the virtual simulation of robot motions lies in the fact that the virtual environment doesn't always correspond to the real one and this discrepancy can be critical, especially in designing collision free paths. For this reason, it's necessary to align the virtual geometry with respect to the real environment in order to compute consistent robot paths.

This work aims to solve the problem of aligning virtual environments to match their real counterparts. Here is presented a method that uses a collaborative robot to get a set of inspection points by letting the robot touch certain surfaces while monitoring the joint values, and then updates the positions of corresponding virtual objects. In this thesis is also introduced a variation of that method, capable of handling large inspections sets, such as point cloud generated by 3D scanners.

Contents

1. Introduction 13

1.1 Background 17

1.1.1 Fraunhofer-Chalmers Centre (FCC) 17

1.1.2 UNIFICATION Project 19

1.2 Outline 19

2. Geometry Alignment using collaborative robot 20

2.1 Algorithms 22

2.1.1 Point Matching process 22

2.1.2 Singular Value Decomposition (SVD) method 25

2.1.3 Iterative Closest Point (ICP) algorithm 29

2.1.4 Stopping Criterion 30

2.1.5 Point Set Alignment algorithm 32

2.1.6 Coherence of correspondences 35

2.2 Technologies 37

2.2.1 Software 37

2.2.2 Hardware 43

2.3 Implementation 47

2.3.1 Geometry Alignment Dialog 47

2.3.2 Point Set Alignment 49

2.3.3 Communication IPS-Cobot 51

2.3.4 Tests 57

2.4 Results 59

2.4.1 Inspection phase 59

2.4.2 Alignment phase 60

2.5 Conclusions 62

3. Geometry Alignment using Point Cloud 64

3.1 Algorithms 66

3.1.1 Keypoint detection 67

3.1.2 Filtering 69

3.1.3 Point Cloud Alignment algorithm 72

3.2 Technologies 77

3.2.1 Point Cloud Library – PCL 77

3.2.2 GNU Octave 78

3.3 Implementation 78

3.3.1 Geometry Alignment Dialog 78

3.3.2 Point Cloud Alignment 80

3.3.1 Tests 82

3.4 Results 83

3.5 Conclusions 93

4. Closing Remarks 95

References 96

List of figures

Figure 1 – Industrial robots annual shipments by regions........13
Figure 2 – Industrial robots annual shipments by sectors........14
Figure 3 – Tunnel bracket assembly........15
Figure 4 – Welding cell optimization........16
Figure 5 - FCC logo........17
Figure 6 – FCC’s business model........18
Figure 7 – 12 degrees of freedom........20
Figure 8 – 3-2-1 Principle........21
Figure 9 – Non-rigid Registration........23
Figure 10 – Rigid Registration........23
Figure 11 – Comparison table........25
Figure 12 – Initial situation........27
Figure 13 – Optimal translation........27
Figure 14 – Optimal rotation........27
Figure 15 – Final solution T........27
Figure 16 – Stopping criterion........32
Figure 17 - Inspection-Object points correspondences........33
Figure 18 – Alignment Steps........34
Figure 19 – Inspection acquisition........35
Figure 20 – Inspection’s coherence........36
Figure 21 – IPS logo........37
Figure 22 – ROS logo........40
Figure 23 – ROS architecture........42
Figure 24 – ROS Topic protocol........42
Figure 25 – ROS Service protocol........43
Figure 26 – Cobots sales estimation........44
Figure 27 – Collaborative robotics positioning........45
Figure 28 – UR 10........46
Figure 29 – Iiwa 7........47
Figure 30 – Geometry Alignment dialog........48
Figure 31 – IPS to ROS communication scheme........54
Figure 32 – IPS to Iiwa communication scheme........55

Figure 33 – Aligned geometry......57
Figure 34 – Misaligned geometry......57
Figure 35 – Inspection result......59
Figure 36 – Iiwa inspection test......59
Figure 37 – 6 inspections geometry......60
Figure 38 – Tests results with δ = 100mm......61
Figure 39 – Test results with δ = 500mm......61
Figure 40 – 3D active scanner......65
Figure 41 – Active scanning tecnhique......65
Figure 42 – ABB IRB1600 point cloud......66
Figure 43 - Voxel grid......68
Figure 44 – Uniform sampling......69
Figure 45 – Cut-off filter......71
Figure 46 – Measure filter......72
Figure 47 – Weighting function......74
Figure 48 – Reduction function......75
Figure 49 – PCL logo......77
Figure 50 – Point cloud panel......79
Figure 51 – PCA Tests......82
Figure 52 – ABB IRB-1600 Model......83
Figure 53 – ABB IRB-1600 links......83
Figure 54 - IRB test results with δ = 200mm......84
Figure 55 – Link2 test results with δ = 50mm......85
Figure 56 – Link3 test results with δ = 500mm......86
Figure 57 – Link0 test results with δ = 1000mm......86
Figure 58 – Link1 test results with δ = 1000mm......87
Figure 59 – Link2 test results with δ = 500mm......88
Figure 60 – Link4 test results with δ = 200mm......89
Figure 61 – Link4 test results with δ = 500mm......89
Figure 62 – Link5 and Link6 detail......90
Figure 63 – Relationship δ-Time......92
Figure 64 – Number of points analysis......92
Figure 65 – Link 4 problem......94

1. Introduction

As stated by the International Federation of Robotics (FIR) [1] in the 2017 publication of its annual statistical study [2], since 2010, the shipments of industrial robots have greatly grown, thanks to the ongoing trend toward automation and to constant improvements and technical innovation in the area of industrial robots. As result of this growth, at the end of 2016, the total worldwide stock of industrial robots was about 1.8 million units. This scenario is well depicted by the graph below, displaying the thousands of industrial robot units sold per year, from 2007 to 2016.

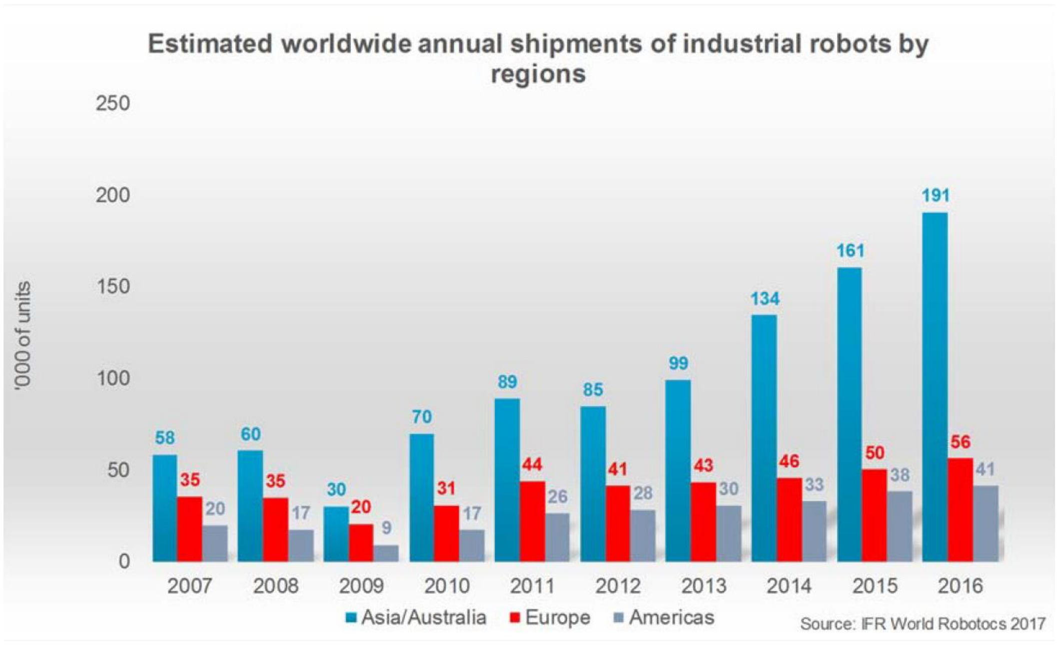


Figure 1 – Industrial robots annual shipments by regions

The exponential raise in the automation market that has been experienced in last years will probably continue in the next future. Indeed, FIR estimates that, from 2018 to 2020, global robot installations will increase by at least 15% on average per year. This means that, between 2018 and 2020, more than 1.7 million new industrial robots will be installed in factories around the world and in 2020 the total global sales will be around 520,900 units.

In *Figure* 2 is shown how sales are distributed between industrial sectors. In 2016, like in 2015, the driving industry of this expansion was the electrical/electronics one. Despite this, the major customer of industrial robots is still the automotive industry, with a market that grows moderately but constantly.

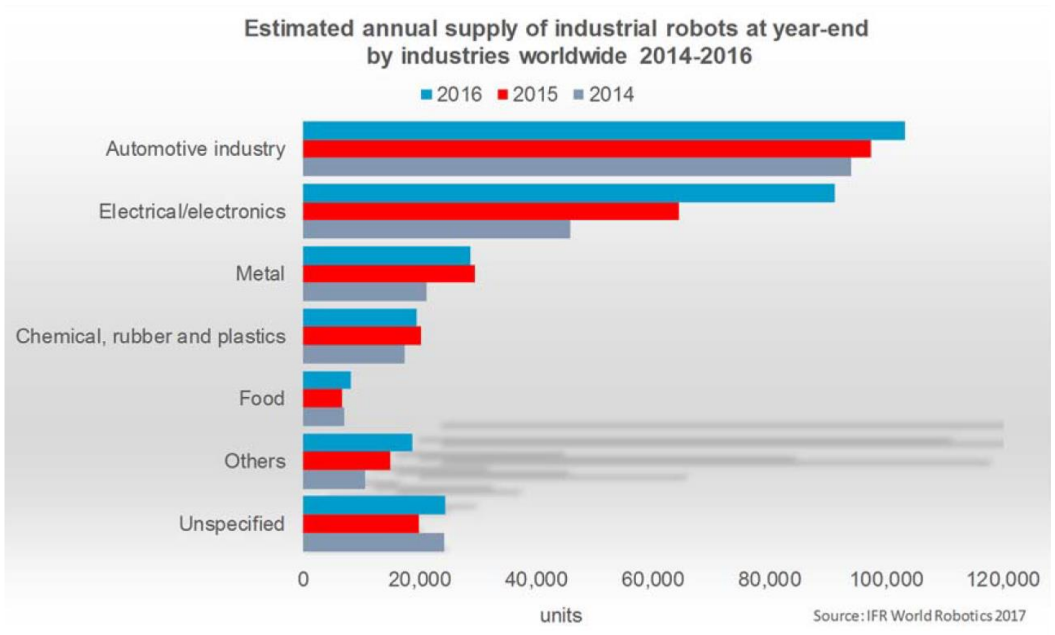


Figure 2 – Industrial robots annual shipments by sectors

The raise in shipments is reflected, of course, in the value of industrial robots' market. Indeed, in 2016, the sales value reached a new peak at US$13.1 billion. Notice that, this value doesn't include the cost of software, peripherals and systems engineering. Including those costs the worldwide market value for robot systems is obtained and, in 2015, this value was estimated to be US$40 billion.

From this brief analysis it appears clear that the level of automation and the importance of all processes and techniques concerning robotics are quickly increasing in modern industry and this trend will almost certainly continue in the next years.

One of the most important processes concerning industrial automation is the robots path planning and optimization, which addresses the problem of decomposing the movement that the robot should perform to complete a task into discrete motions respecting constraints, avoiding collisions and optimizing some specific parameters.

In general, as stated by Steven M. LaValle [3], the *path planning* or *motion planning* problem is to "convert high-level specifications of tasks from humans into low-level descriptions of how to move".

An example of this is the automotive assembly problem shown in the *Figure 3,* where a tunnel bracket has to be mounted avoiding collisions. This operation isn't not trivial and can be very time consuming for assembly engineers. For this reason, software tools capable of performing automatic paths computation and optimization are extremely helpful to engineers in this designing phase. Indeed, the solution to the problem depicted in 4 steps in *Figure 3* has been computed in less than 2 minutes by the simulation software IPS [4], developed by FCC [5], while an assembly engineer would have spent much more time, maybe days, to find a solution, as explained in [6].

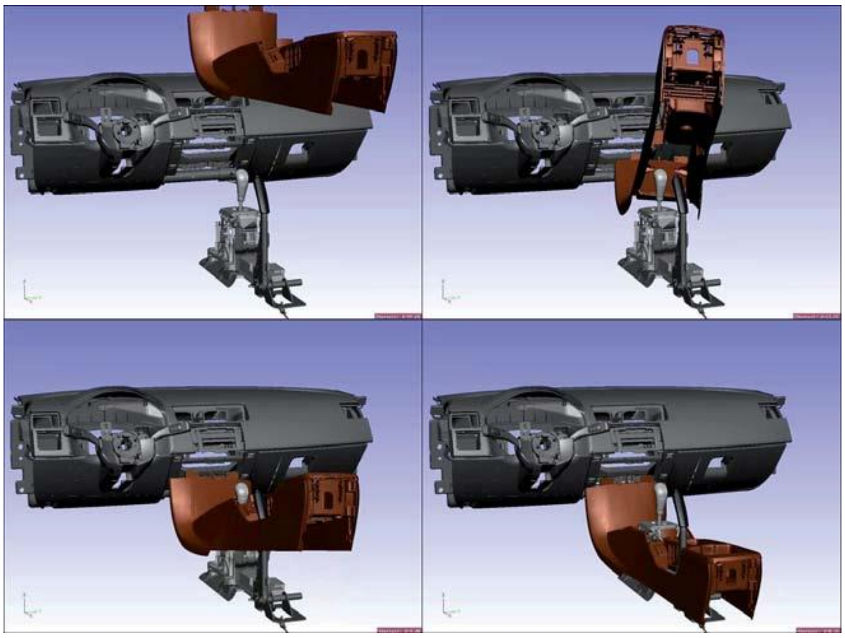


Figure 3 – Tunnel bracket assembly

Therefore, as stated in *Principle of Robot Motion* [7], the main goal in automatic path planning is to "be able to specify a task in a high-level language and have the robot automatically compile this specification into a set of low-level motion primitives, or feedback controllers, to accomplish the task".

Besides finding the solution to the path planning and assembly feasibility problems, software tools for automatic computation can offer many other powerful functionalities to engineers, allowing them to perform simulations and optimizations not feasible "manually". For example, with the IPS framework it's possible to integrate assembly design, path planning and sequence optimization, maximizing performances in multi-robot manufacturing cells. *Figure 4* shows an image from the simulation of welding operations performed in a 4-robots cell. This simulation allows to find collision-free alternatives to perform each welding operation, distribute the welding operations between the robots to minimize the cycle time and decide in which order and with what alternative the robot should weld. This process provides the plan for an optimized welding process, with faster commissioning and improved cycle time, preserving of course the coordination among robots to avoid collision.

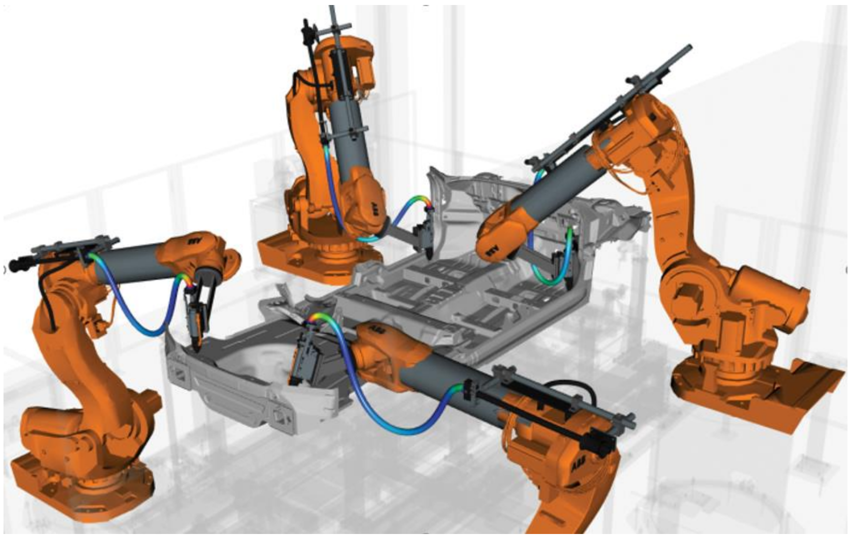


Figure 4 – Welding cell optimization

A common requirement for all software tools doing automatic motions planning computation is that the virtual environment in which the simulation is performed, usually derived from the CAD model, has to precisely reflect the real environment. Indeed, if that doesn't happen, the computation performed by the software could lead to collisions when applied, due to different positions of parts in the real environment. Therefore, this requirement asks for a precise calibration of the virtual geometry with respect to the real one which normally is a very time-consuming operation and requires efforts to achieve a precise result.

In this thesis is introduced a novel method for automatic geometry alignment, which aims to solve the alignment problem performing the registration of the virtual geometry with respect to the real counterpart. This method takes as input data a set of inspection points, which indicates the position of the real objects in space and updates the virtual geometry to make it match the inspections. Collection of inspections is performed with the aid of a collaborative robot. Indeed, by letting the robot touch the surface of the object and monitoring the joints state in that pose, it's possible to derive the exact position of the point in which robot and object are in contact and that is the inspection point.In this project is proposed also an alternative method for automatic alignment, which takes as input data a point cloud, captured with a 3D scanner, instead of a set of points collected through a collaborative robot. The point cloud constitutes a representation of the real environment and the algorithm presented here extracts from it the information useful to the alignment of the virtual geometry.

1.1 Background

This thesis project has been developed at Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC), in Gothenburg (Sweden) and it's part of the research project *Virtual Commissioning of Vehicle Maintenance Operations – UNIFICATION*, supported by VINNOVA [8], the Swedish governmental agency for innovation systems. The research partners are FCC, Chalmers, Volvo Group, SETEK, and ARHO.

1.1.1 Fraunhofer-Chalmers Centre (FCC)

Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC) [5] is a research center located in Chalmers Science Park. Its main focus is to bring solutions to advanced mathematical problems in industry.

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Figure 5 - FCC logo

In particular FCC is offering contract research, services, algorithms and software oriented to Modeling, Simulation and Optimizations (MSO) issues. MSO provides a significant leading edge in industrial innovation of products and production systems.
The Fraunhofer-Chalmers Research Centre was founded by Chalmers and the Fraunhofer-Gesellschaft in 2001. Fraunhofer is Europe's largest application-oriented research organization, while Chalmers is a highly progressive university situated in Gothenburg, Sweden.
The possibility of long term collaborations with Fraunhofer and Chalmers is a great advantage for FCC and it has contributed to the results achieved in projects with more than 160 clients, operating in different business areas: automotive, pharmaceutical, wood and paper, and electronics industries. These clients are located mainly in Sweden, but also in Germany, US, Finland, Denmark, Japan and Great Britain.

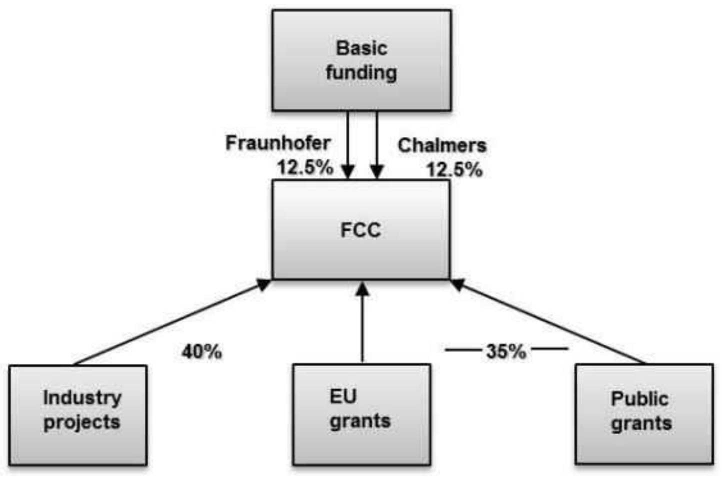


Figure 6 – FCC's business model

Fraunhofer-Chalmers Centre is divided in three departments: Geometry and Motion Planning, Computational Engineering and Design and System and Data Analysis.

The department Geometry and Motion Planning develops mathematical tools and algorithms for motion planning of robots and for visualization, simulation, optimization, and statistical analysis of geometrical variation and tolerances, thereby supporting the virtual production and manufacturing process of assembled products.

The department Computational Engineering and Design works on novel numerical methods, fast algorithms and engineering tools to support virtual product and process development. Applications include fluid dynamics, structural dynamics and electromagnetics.

The department Systems and Data Analysis offers competence in dynamics systems, prediction and control, image and video analysis, mathematical statistics, and quality engineering in both technical and biological/biomedical applications.

This thesis has been carried out in the contest of the Geometry and Motion Planning department and it's in tight relation with other research project performed by FCC dealing with automatic generation of optimized and collision free robot motions, which is an important industrial application for many FCC's clients. In particular, this work is oriented to the integration of alignment functionalities in the IPS simulation software framework [4], developed and sold by FCC.

1.1.2 UNIFICATION Project

As already mentioned, this work is part of a project called UNIFICATION, supported by Sweden's innovation agency VINNOVA, programme Production2030.
The main objective of UNIFICATION project is to provide and demonstrate a digitalized vehicle maintenance procedure where human operators and collaborative robots share the assembly or disassembly tasks, use common smart tools and interacting with both the vehicle and other machines.

1.2 Outline

This thesis is structured in two chapters. The first one addresses the solution to the alignment problem adopting a collaborative robot to gather information on objects' positions in real environment. In particular, it presents the methodologies and technologies used to perform the inspection and alignment phases , describing also the implementation details. The other chapter, instead, discusses an alternative solution to the problem, which doesn't involve directly collaborative robots, since information needed in the alignment phase are extracted from a point cloud representing the environment. Information extraction process and algorithms adopted in the solution are presented in this chapter, along with all the implementation details.
In both chapters are shown also the results obtained with the described solution and some final considerations on them are drawn.

2. Geometry Alignment using collaborative robot

In this chapter is addressed the problem of aligning the virtual geometry of an object with respect to its real counterpart. As already, said the method proposed here is based on the use of collaborative robots to gather information about the position of the object that has to be aligned. In practice, the collaborative robot, guided by a human user or programmed to complete the task in automatic mode, has to touch the surface of the object, then, by monitoring its joints value, it's possible to derive the position of the contact point and, in this way, a set of inspection points can be collected.

Inspection phase guidelines

The number of inspections and the way in which they're collected aren't random, but instead they're based on specific principles. First of all, it's important to notice that, in order to precisely identify a position of an object in space, at least 6 inspection points are required. Indeed, any free body has a total of 12 degrees of freedom, 6 translational (+X, -X, +Y, -Y, +Z, -Z) and 6 rotational (X-CW, X-CCW, Y-CW, Y-CCW, Z-CW, Z-CCW), which have to be fixed in order to locate it.

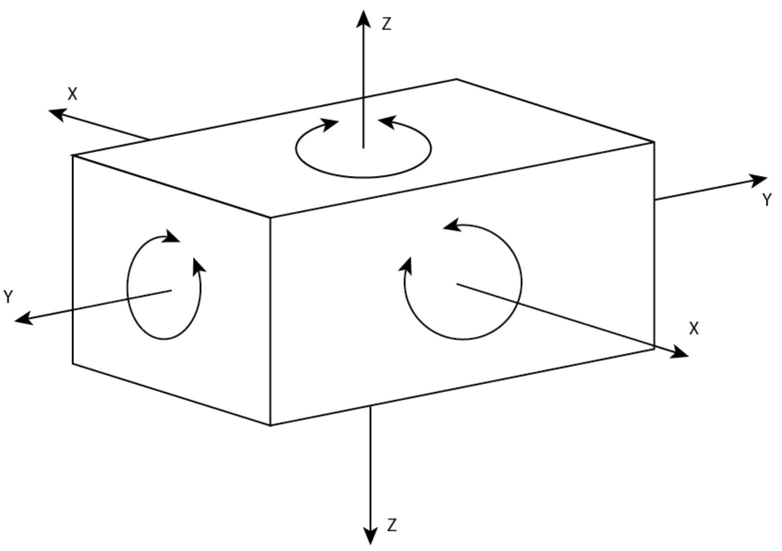


Figure 7 – 12 degrees of freedom

This can be done through the so called *3-2-1 Principle* or *Six Points Principle,* which works as follows:

1. Rest the object on three non-collinear points (A, B, C) of the bottom surface (XY), fixing in this way the +Z, CROT-X, ACROT-X, CROT-Y and ACROT-Y degrees of freedom.
2. Rest the object at two points (D, E) of side surface (XZ), fixing the +Y and ACROT-Z degrees of freedom.
3. Rest the work piece at one point (F) of the adjacent surface (YZ), fixing the +X and CROT-Z degrees of freedom.

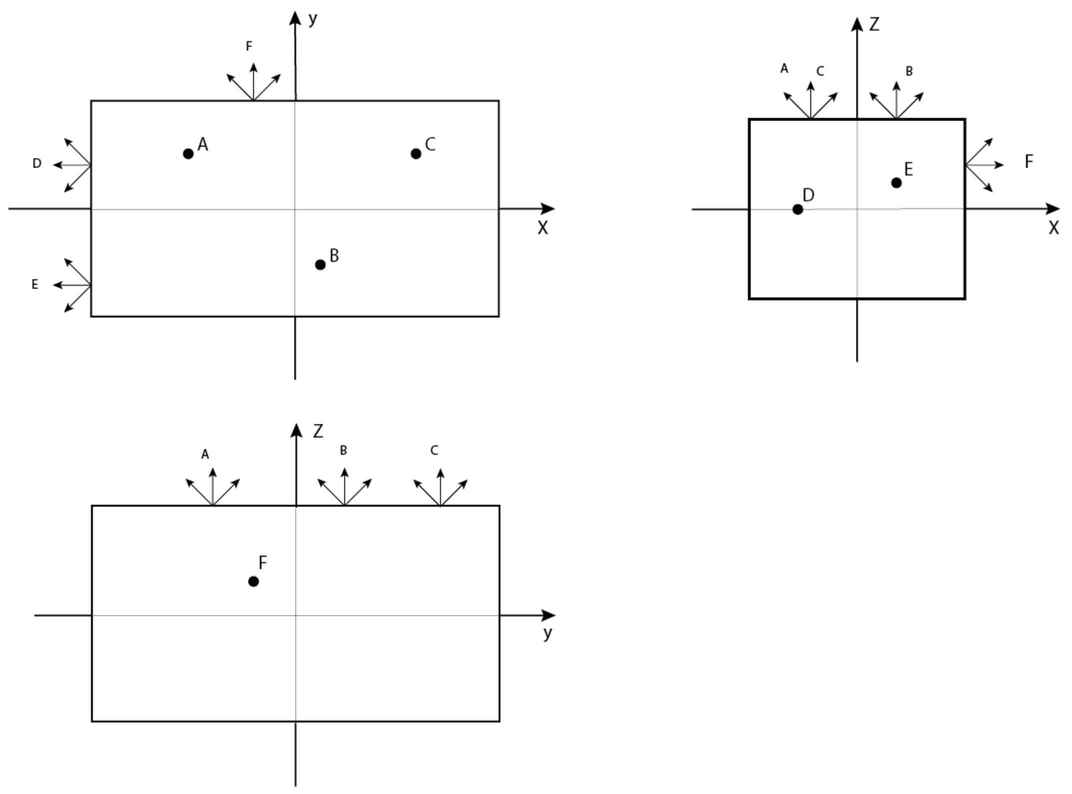


Figure 8 – 3-2-1 Principle

At the end of the inspection phase, the alignment process can be performed according to the gathered information. The idea behind the solution proposed in this work is to reduce the geometry alignment problem to a point matching one, which means, starting from the set of inspection points, derive a set of corresponding points on the object's surface and then match those sets, obtaining as a consequence the object's alignment. Unfortunately, finding those points on the object's surface which correspond exactly to

the inspection points is not a trivial operation and, therefore, a more complex algorithm is needed to put in practice the idea mentioned above.
In particular, the algorithm proposed in this thesis to solve the problem is based on the well-known *Iterative Closest Point (ICP)* algorithm [9], which is an iterative method frequently used when it comes to minimize the difference between two sets of points. The designed algorithm, indeed, is iterative like the ICP and it follows the same philosophy of finding, at each iteration, the couples of closest points in the two sets and align them. Basically, this algorithm computes at each iteration, for every point in the inspections set the correspondent closest point on the object's surface and then computes the geometrical transformation that minimize the difference between those two sets and looping until the object matches the inspections.
In this chapter is proposed a solution for the mentioned problem. First of all is presented an overview on the algorithms and methods used to solve the problem. Follows a description of software and hardware technologies adopted, then the implementation of the solution is exposed. Finally, a section devoted to the presentation of obtained results and one drawing conclusions and possible improvement are present.

2.1 Algorithms

This section offers an overview on methods involved in the solution of the geometry alignment problem proposed in this project.
As already said, the designed algorithm reduces the geometry alignment problem to a *Point Matching* problem, so, first of all, it's given here a detailed description of that problem and a list of literature sources which address it. In particular the *Singular Value Decomposition (SVD)* method [10] and the *Iterative Closest Point* algorithm [9] are explained and analysed step by step since they're used in the proposed solution.
After this preliminary introduction to all the inspiring sources and fundamental principles it's finally explained the *Point Set Alignment* algorithm, conceived to solve the geometry alignment problem.

2.1.1 Point Matching process

Considering two different point sets, the *Point matching* (or *Point-set Registration*) problem consists in finding a geometrical transformation able to align them. More precisely, given a "scene" point set and a "model" point set, the point matching process aims to find a transformation that applied to the model point set makes it match the scene one. As stated in a more formal way by Jian and Vemuri [11], let $\{M, S\}$ be the two finite size point sets to be registered, where M is the moving "model" set, S is the fixed "scene" set and both, M and S, are subsets of a finite-dimensional real vector space $\mathbb{R}^d$ and in general, they can be of different sizes. The registration process estimates the spatial transformation T, from $\mathbb{R}^d$ to $\mathbb{R}^d$, which yields the alignment between the transformed model and the target scene set.

$$S = T(M), \quad T: \mathbb{R}^d \rightarrow \mathbb{R}^d \qquad (1)$$

Notice that T can be either a rigid or a non-rigid transformation.
A rigid transformation is a transformation that does not change the distance between points, so it doesn't modify neither the size nor the shape of the object. Typically, is a composition of translation and rotation.
Non-rigid transformations instead, include affine transformations such as scaling and shear mapping. In the context of point set registration, non-rigid registration typically involves nonlinear transformation.
When a matching method yields to rigid transformations it's called *Rigid registration,* while if it produces non-rigid transformation it's called *Non-rigid registration.* In this work only rigid transformations will be used, since the nature of the problem doesn´t fit the non-rigid registration case.

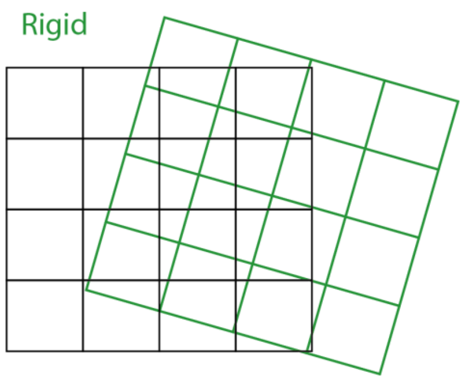


Figure 10 – Rigid Registration

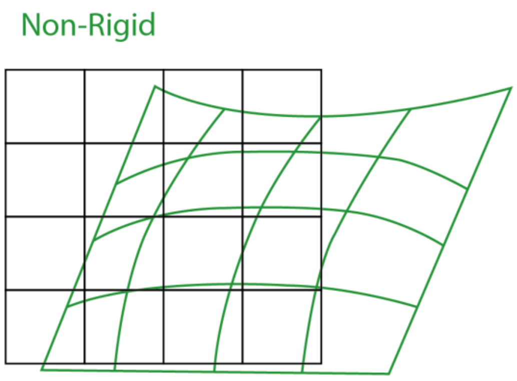
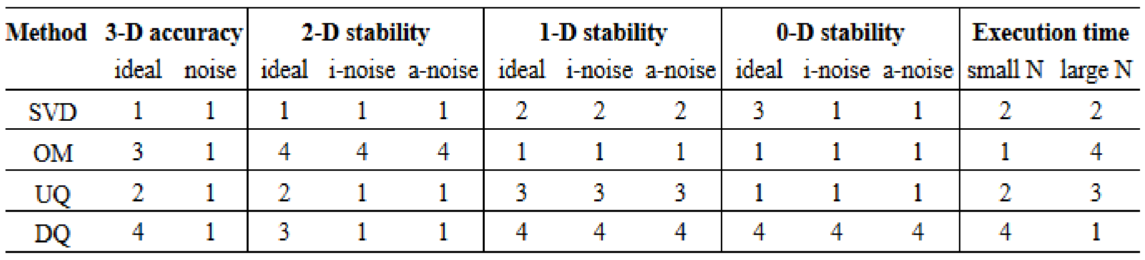


Figure 9 – Non-rigid Registration

From the point matching problem it's possible to derive a more specific sub-problem, called *Correspondent point matching.* Differently from point matching, in this problem the two sets M and S have the same size and corresponding pairs of points have already been determined.

In literature there are many papers addressing the corresponding point matching problem and, among them, there is a particularly interesting group in which are presented closed-form methods to solve the problem. For example, a solution, based on Unit Quaternion (UQ), was first proposed by Faugeras and Hebert in 1986 [12] and rivised by Horn, who presented an alternative formulation of the method [13]. Horn, in cooperation with Hilden and Negahdaripour, came up also with another algorithm [14], involving Orthonormal Matrices (OM), while Walker, Shao and Volz designed a method [15], oriented to the same issue and based on Dual Quaternions (DQ). In addition to these solution, there is also the already cited algorithm, based on Singular Value Decomposition (SVD) matrix and exposed in a paper by Arun, Huang and Blostein [10]. This last is the method adopted in this project and for this reason it's detailed in the following section.
The choice of designing a solution that embraces the SVD method has been done relying on Lorusso, Eggert and Fisher's work. Indeed, in the paper *Estimating 3-D rigid body transformations: a comparison of four major algorithms* [16], they made an analysis of the four main methods aimed to find the closed-form solution to the correspondent point matching problem. The article exposes also an exhaustive performances comparison between those four algorithms, which are the ones cited above: UQ (in the Horn's formulation), OM, DQ and SVD algorithms.
The table below, published in the mentioned paper, shows the qualitative comparison of performances between algorithms. The values written in the table provide a measure of the algorithm's performance in some specific experiments (1 = best, 4 = worst). Ratings are based on the overall responses to different noise levels (ideal, noise, i-noise, a-noise) and data set sizes (small N, large N). In particular, comparisons are given for accuracy/robustness using 3-D data sets, both with and without noise, stability of response on degenerate data sets corrupted with no noise (ideal), isotropic noise (i-noise) and anisotropic noise (a-noise) and the overall execution time for small and large data sets. In order to have a detailed view on how these experiments have been conducted, please refer to [16].



Method	3-D accuracy		2-D stability			1-D stability			0-D stability			Execution time	
	ideal	noise	ideal	i-noise	a-noise	ideal	i-noise	a-noise	ideal	i-noise	a-noise	small N	large N
SVD	1	1	1	1	1	2	2	2	3	1	1	2	2
OM	3	1	4	4	4	1	1	1	1	1	1	1	4
UQ	2	1	2	1	1	3	3	3	1	1	1	2	3
DQ	4	1	3	1	1	4	4	4	4	4	4	4	1

Figure 11 – Comparison table

From these results it appears that UQ and SVD algorithms are overall the most accurate and stable ones, even if, in any practical applications, there aren't discernible differences between methods. From the execution time point of view instead, OM and DQ are respectively the best on small and large data set sizes, but they're the worst in the opposite case, while the SVD method is the second best both for small and large sets.
In conclusion, analysing this comparison, the SVD algorithm has been chosen for this project since it presents the overall best behaviour in the experiments and, moreover, it's easy to manage in the implementation phase. The explanation of this method, with a detailed description of all steps composing it, is presented in the following section.

2.1.2 Singular Value Decomposition (SVD) method

As already said, the Singular Value Decomposition method was introduced in 1987 by Arun, Huang and Blostein, with the aim of providing a new and efficient solution for the corresponding point matching problem, based on the singular-value decomposition of a 3×3 matrix.
A singular-value decomposition is a factorization of an m x n matrix M in the form

$$M = U \Lambda V^T = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^T \qquad (2)$$

where $U, V \in \mathbb{R}^{3x3}$are unitary and $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}_0^+$ are the singular values of M.
What follows is a description of the SVD method, based on Arun, Huang and Blostein paper [10]. The notation used in this explanation is different from the one in the article, however concepts and equations are the same.

Let $I = \{i_k\}$ be the *scene point set* and $P = \{p_k\}$ the *model point set*, where $N_I = N_P = N$. This method aims to find the transformation *T*, composed by the rotation *R* and the translation *t* which minimize the following function

$$f(R,t) = \sum_{k=1}^{\mathrm{N}} ||i_k - Rp_k - \mathrm{t}||^2 \qquad (3)$$

In practice, the goal is to minimize the overall distance between points in *I* and points in *P* transformed by *T*. Follow the steps to obtain *T*.

Let be *I´* and *P´* two point-sets calculated by subtracting, to every point in *I* and *P*, the corresponding centre of mass of the point set, μ_I for *I* and μ_P for *P*.

$$I' = \{i_k - \mu_I\} = \{i'_k\} \qquad P' = \{p_k - \mu_P\} = \{p'_k\} \qquad (4)\,(5)$$

where μ_{I} and μ_{P} are computed like this

$$\mu_I = \frac{1}{N_I}\sum_{k=1}^{N_I} i_k \qquad \mu_P = \frac{1}{N_P}\sum_{k=1}^{N_P} p_k \qquad (6)\,(7)$$

In order to obtain the optimal rotation R, first the 3 × 3 matrix *H* is calculated

$$H \triangleq \sum_{k=1}^{N} p'_k i'^T_k \qquad (8)$$

and then the SVD of *H* is computed

$$H = U\Lambda V^T \qquad (9)$$

and finally, the *R* matrix is calculated

$$R = VU^T \qquad (10)$$

If $\det(R) = +1$, then *R* is the 3 × 3 optimal rotation matrix, otherwise, if $\det(R) = -1$ it means that *R* is a reflection rather than a rotation. This special case occurs when one of the singular values of *H* is zero, due to the presence of large amounts of noise or dealing with planar data sets. The desired rotation, in this case, is found by forming $R' = V'U^T$, where $V' = [v_1, v_2, -v_3]$ and v_3 is the third column of *V,* corresponding to the singular value of *H* equal to zero.

At this point, knowing the optimal rotation *R,* it's possible to derive the optimal translation *t* as follows

$$t = I - R(P) \qquad (11)$$

and therefore, the optimal transformation will be $T = [t, R]$.

Notice that, summarizing the explained SVD method is possible to distinguish three main steps which are also common to the other solutions mentioned before:

1. finding the centroids μ_I and μ_P
2. finding the optimal rotation R (in this case using the SVD matrix)
3. finding the optimal translation

These steps are depicted in the figures below.

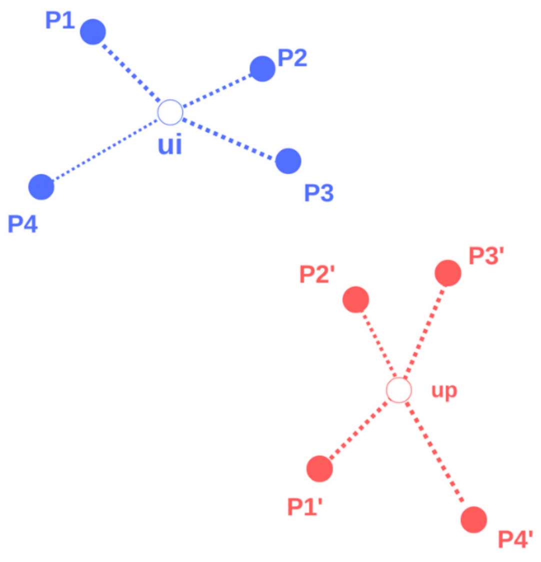


Figure 12 – Initial situation

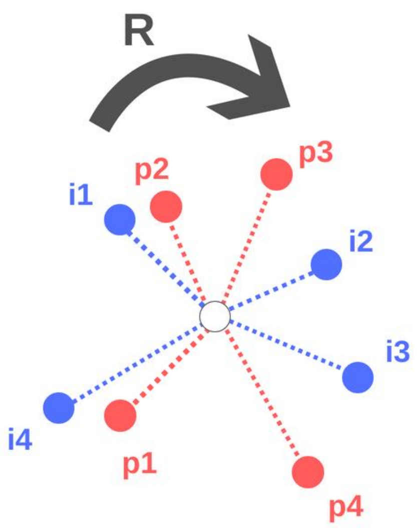


Figure 13 – Optimal translation

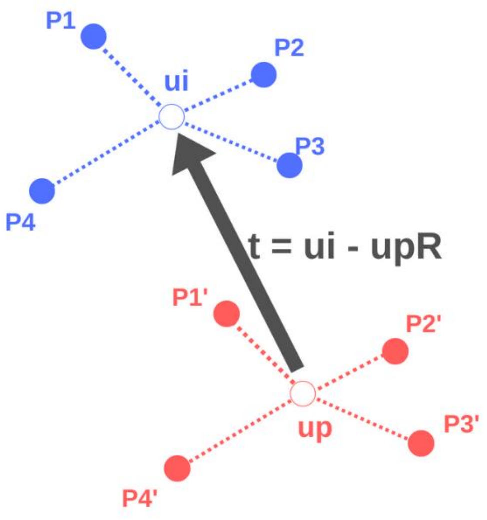


Figure 14 – Optimal rotation

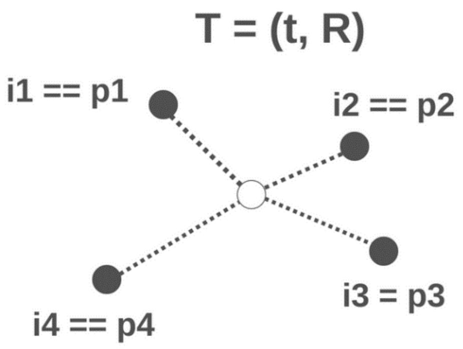


Figure 15 – Final solution T

In the SVD method presented above, all the pairs of correspondences have the same *weight* and it means that they contribute in the same way to the registration process. However, it exists also a weighted version of that algorithm, called here *Weighted SVD*, in which, as the name suggests, a specific weight is associated to each correspondence. This alternative version of the method is detailed below, following the Hornung and Rabinovich's work [17].

The main difference with the weighted version lies in the centroids computation, as shown in the equations below where w_k corresponds to the weight of the k-th correspondence.

$$\mu_I = \frac{\sum_{k=1}^{N} w_k i_k}{\sum_{k=1}^{N} w_k} \qquad\qquad \mu_P = \frac{\sum_{k=1}^{N} w_k p_k}{\sum_{k=1}^{N} w_k} \qquad (12)\,(13)$$

This new computation of the centres of mass leads to *weighted centroids* with a position that depends on the weights. In fact, correspondences with a greater weight have a stronger influence on the centroid's position, on the contrary if a correspondence has a small weight it will be less important. This property has been exploited in the design of the Point Set Algorithm which will be explained at a later stage.

As mentioned above, the SVD algorithm and all other closed-form methods presented here (UQ, DQ, OM) provide a solution to the corresponding point matching problem. Nevertheless, in the case of application addressed in this work, the correspondences between inspection points and points on the object's surface are not known in advance and for this reason the SVD algorithm is not enough. In other words, an algorithm able to solve the more general point matching problem is needed and the choice has fallen upon the *Iterative Closest Point (ICP)* algorithm.

The point set registration problem is a complex problem, with many facets, and it´s been analysed from many points of view and in all its different aspects. Therefore, of course, ICP is not the only algorithm developed to solve this problem. In literature are present many other methods that face the point-set registration problem, like the *Robust Point Matching (RPM)* [18]and the *Coherent Point Drift (CPD)* [19], and even within ICP there are lots of variations to the original approach [20].

In this work, however, only the ICP algorithm is addressed, since it well fits the requirements and, moreover, it has a simple approach to the problem that leads to an easier implementation.

2.1.3 Iterative Closest Point (ICP) algorithm

The *Iterative Closest Point (ICP),* introduced by Besl and McKay [9], is an iterative algorithm for registration of 3-D shapes through a rigid transformation. This algorithm indeed, aims to obtain the alignment by taking couples of closest points and then finding the least squares rigid transformation that minimize an error function on these couples. What follows is the description of the ICP algorithm and it's based on the original work of Besl and McKay [9]. First some formulas used in algorithm are explained, then the actual ICP algorithm is stated, in particular the steps executed on each iteration are listed.

Given two point-sets, the *model point set* $X = \{x_i\}$ and the *data point set* $P = \{p_i\}$, with N_X and N_P points respectively, for each $p \in P$ is defined a closest point in X

$$y = y(p, X) = \underset{x \in X}{argmin} \; \|x - p\| \qquad (14)$$

This operation has a cost $O(N_x)$ in the worst case, $\log(N_x)$ as expected cost.

The application of closest-point operation to all the points in *P* leads to obtain a set of closest points. Let *Y* denote this set of closest points and let *C* denotes the closest point operator, the operation can be defined as

$$Y = C(P, X) \qquad (15)$$

At this stage, having two sets, with $N_Y = N_P = N$, and knowing the correspondences between them, it's possible to proceed with the registration, adopting one of the closed-form methods explained above. As already said, in the solution designed for this project the SVD method is applied, however, since Besl and McKay used the UQ algorithm in their dissertation, for sake of coherence the same method is reported here in the explanation of ICP algorithm, intended that any of the other methods could be adopted obtaining a similar result.

Therefore, applying the Unit Quaternion method [13] on the point sets, the optimal transformation *T* which minimize the mean square point matching error d is obtained

$$(T, d) = UQ(P, Y) \qquad (16)$$

and then the data point-set can be updated applying *T*

$$P' = T(P) \qquad (17)$$

Now that all the equations involved have been introduced, the actual structure of the ICP algorithm may be explained. In practice, it consists of the iterative repetition of the process described above, till the error d end up below a certain threshold.
Let k be the iteration number and τ the desired tolerance. The following is the list of steps executed by the algorithm at each iteration:

1. Compute the closest points

$$Y_k = C(P_k, X) \tag{18}$$

2. Compute the registration

$$(T_k, d_k) = UQ(P_k, Y_k) \tag{19}$$

3. Apply the registration

$$P_{k+1} = T_k(P_k) \tag{20}$$

4. If $d_k - d_{k+1} < \tau$, which means that the change in mean square error is below τ, the algorithm stops, otherwise loops to the next iteration.

In Besl and McKay' work it's demonstrate that this algorithm converges monotonically to a local minimum from any given rotation and translation of the data point-set, nevertheless it may or may not converge to the desired global minimum. In [9] Besl and McKay explain how, using an appropriate set of initial state, it´s possible address the algorithm to the desired global minimum, but these technical arrangements are left out, since they´re not useful in this dissertation.
Notice that the algorithm terminates when the condition expressed at step 4 is respected, therefore, that condition can be considered as the ICP algorithm's *Stopping Criterion.*
The *Stopping Criteria* are an important and well-studied aspect of algorithms design and therefore, in literature, it's possible to find many different approaches to that issue which aim to provide an effective solution depending on the goal of the algorithm in question.
Before proceeding with the presentation of the Point Set Alignment algorithm, it's exposed here the stopping criterion adopted in this work.

2.1.4 Stopping Criterion

The technique used in this project to determine when to stop the algorithm's iteration aims to measure the solution's convergence and thus exit when a new stable position for the object has been found. In practice the algorithm terminates when, for *N* consecutive

iterations, the object's position doesn't change. When this happens, it means that the algorithm found a stable solution to the problem and it's very unlikely that it will change in the following iterations. Notice that *N* can vary based on the considered scenario.
The formalisation of the criterion is the following:

$$d_k = |p_i - p_{i-k}| \le \tau, \quad k = 1, \dots, N \tag{21}$$

where p_i is the object's position at i-th iteration, while the term p_{i-k}, with k varying from 1 to *N*, is the position at the end of the $(N-1)$ previous iterations. In practice, this criterion is respected (i.e. (21) it's true) when the difference $\mathrm{d_k}$ between the current object's position and the position assumed at (i − k)-th iteration is lower than the convergence threshold, with *k* going from 1 to 19.
In general, the difference between two positions (frames) is computed as follows

$$d(T, T^*) = \sqrt{|t - t^*|^2 + (r \cdot d\alpha)^2}, \tag{22}$$

where T and T^* are two transformation representing a specific position and orientation in space. The difference between those two frames is a combination of a *Translational displacement* $\left(\sqrt{|\mathrm{t} - \mathrm{t}^*|^2}\right)$ and a *Rotational Displacement* $\left(\sqrt{(r \cdot d\alpha)^2}\right)$.
The translational term can be seen as the Euclidean distance between the two positions

$$\sqrt{|t - t^*|^2} = \sqrt{dx^2 + dy^2 + dz^2}$$

while the rotational one is given by the angle $d\alpha$ between R and R^* multiplied by the object's radius r, which is useful to calibrate the rotational component with respect to the object's dimension.
It' also important to observe that this stopping criterion just asks for the convergence to a stable solution to exit from the iteration, which could also be a local minimum. Therefore, it may happen that the algorithm terminates without finding a position for the object ensuring the global minimum displacement error.

The following figure depicts the result obtained applying this stopping criterion.

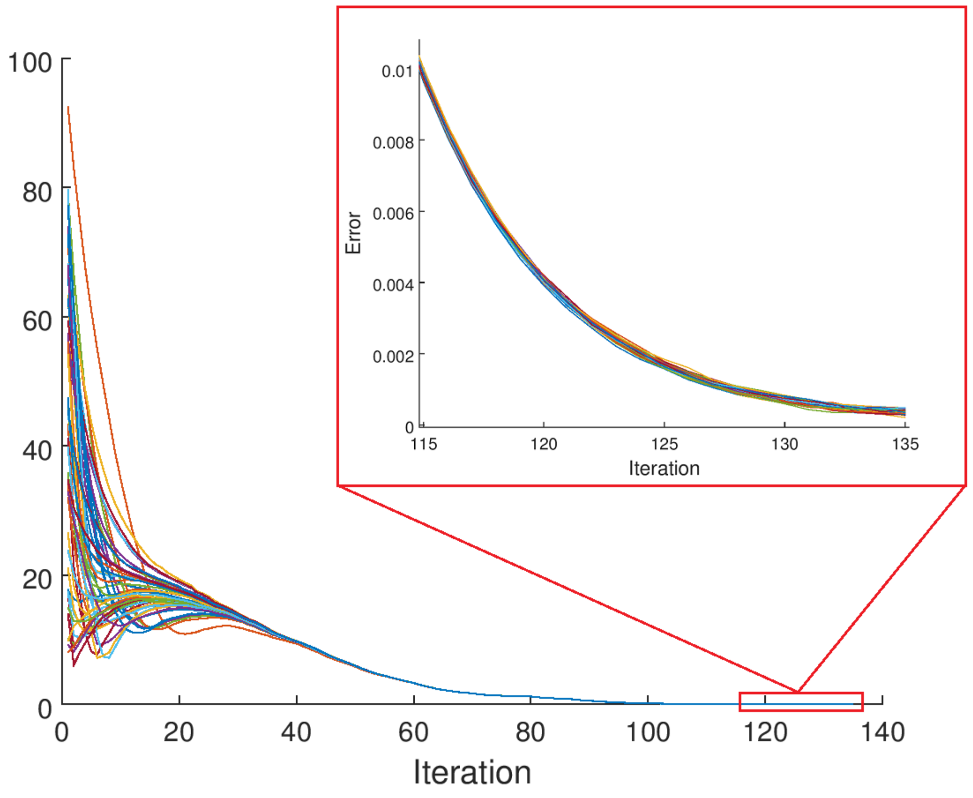


Figure 12 – Stopping criterion

2.1.5 Point Set Alignment algorithm

The *Point Set Alignment (PSA)* algorithm is the algorithm developed to solve the geometry alignment problem underlying this thesis. It provides a method to match a point-set to a rigid body and, in particular, it will be use in this work to match the scanned inspection points representing the physical object with the virtual model of that object.

The main issue in performing this type of alignment is that, not only correspondents are not known, but even the set of model's points which have to match the inspections are not defined. Therefore, the main idea underlying this iterative algorithm is to find, at each iteration and for each point in the inspections set, the closest point belonging to the virtual geometry. As result of this operation, another point-set is obtained, that can be matched to the inspections set using one of the closed form method described above.

The PSA algorithm is basically an adaptation of the ICP to the issue addressed in this thesis. Indeed, the iterative structure and the stages composing the cycles are basically the same, however some significative changes have been made. First of all, in the ICP algorithm the closest point is found through a point-to-point distance computation, while in the PSA algorithm it's derived through a distance computation involving the geometry of an object, which is a much more complex operation. The following image shows the result of this operation, where each inspection point (red) has a correspondent closest point on the object's surface.

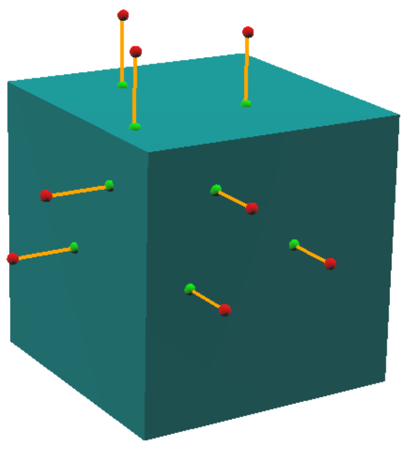


Figure 13 - Inspection-Object points correspondences

The other main difference with the ICP version described above is that the PSA algorithm doesn't use the UQ method but the SVD one.

Algorithm

Having explained all the aspects characterizing the PSA algorithm, now the stages composing the method can be introduced.

Let $I = \{i_k\}$ be the inspections set and S the object to align. Notice that i has to be intended as iteration number and τ as desired tolerance.

These are the steps of the algorithm:

1. Find for each point $i_k \in I$ the closest point on belonging to the object S_i, obtaining in this way the set of point P_i

$$P_i = \cup_{k=1}^{N} p_k, \text{ where } p_k = y(i_k, S_i) \qquad (23)$$

   Notice that y is the same function defined in y=$y(p, X) = \underset{x \in X}{argmin} \|x - p\|$ (14) but here it involves a point-to-object distance computation, instead of a point-to-point one as in the ICP algorithm.

2. Compute the registration applying weighted SVD and obtaining the optimal transformation for the i-th iteration T_i

$$T_i = SVD(I, P_i, W_i) \qquad (24)$$

3. Update the position of the object S_i, applying the computed transformation T_i and moving the object to the new position S_{i+1}

$$S_{i+1} = T_i(S_i) \qquad (25)$$

4. If the stopping criterion explained in *2.1.4* is triggered then exit from the algorithm, otherwise iterate to the next loop.

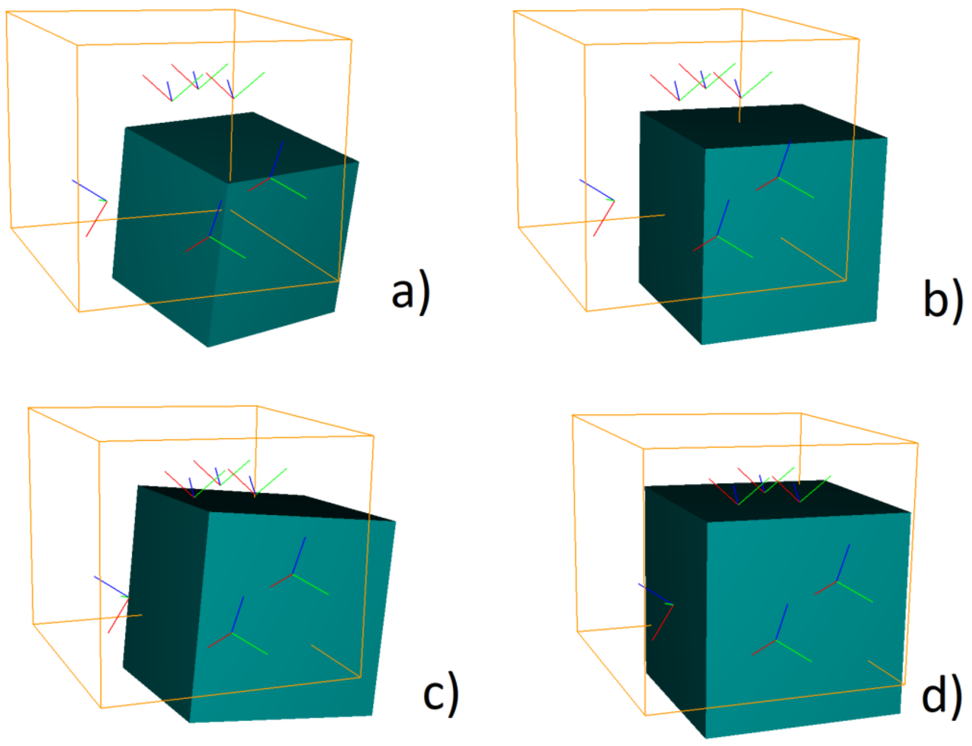


Figure 14 – Alignment Steps

In *Figure 18* is shown the aligning process resulting from the application of the algorithm. In particular, starting from initial situation *(a)* and going on with iterations the object is progressively aligned with inspections, until it precisely matches them as depicted in *(d).*

As with ICP algorithm also with PSA the error converges to a minimum, but it might not be the global one. A possible improvement to steer the algorithm to the global minimum would be to exploit inspections' directional information to determine the coherence of correspondences, as explained in the following section.

2.1.6 Coherence of correspondences

As already mentioned, during the inspection phase a collaborative robot it's used to gather measure points by touching the object's surface. In practice, when an inspection is collected, a reference frame is stuck in the IPS's environment, in correspondence of the contact point between the robot's tool and the object, as shown in *Figure 19*.

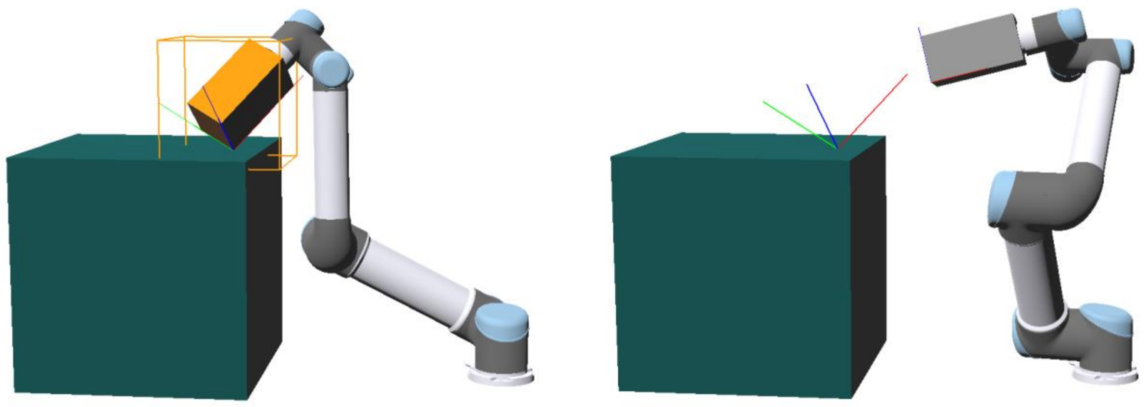


Figure 15 – Inspection acquisition

The produced inspection frame provides not only positional information, but also directional ones, indeed the frame's orientation indicates the pose assumed by tool at the inspection time.

The orientation of the inspection frame can be used to determine if the found correspondence between inspection point and object point is coherent or not.

The idea is the following: assuming that, for each point belonging to the object's surface is known the outward pointing normal N, it's possible to define a correspondence as *coherent* when N points to the first octave of the cartesian coordinate system defined by the inspection frame, otherwise the correspondence can be considered *non-coherent*.

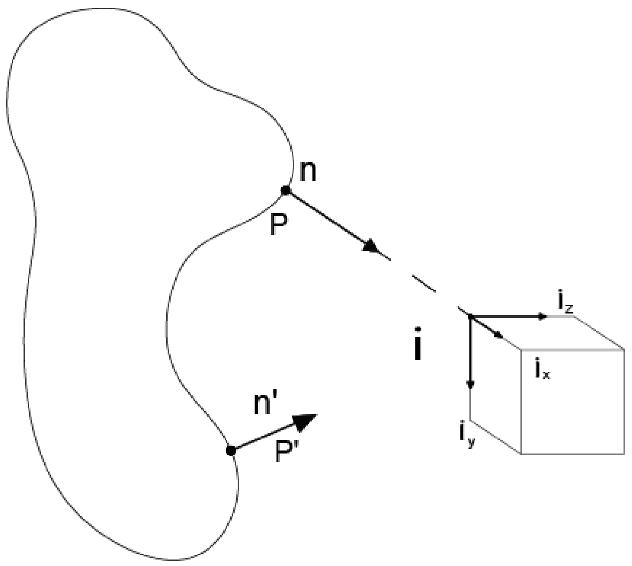


Figure 16 – Inspection's coherence

In the figure above the correspondence between P and i is coherent because the direction of the surface's normal in P, called n, points to the first octave of the coordinate system i. The inspection P', instead, is *non-coherent*, indeed the projection of the normal n' is outside the first octave.

The formalisation of concepts explained above can be exposed as follows.

Let be N the surface normal in the point P and i_x, i_y and i_z the three versors of the inspection frame i, then if

$$\begin{cases} N \cdot i_x > 0 \\ N \cdot i_y > 0 \\ N \cdot i_z > 0 \end{cases} \tag{26}$$

the correspondence is coherent, otherwise is non-coherent.

In practice this additional information can be exploited during the closest point search, excluding from the process those object's points which are not coherent with the orientation of the considered inspection.

Notice that, this improvement adopting directional information has been only theorized, but not implemented and tested in the solution presented here.

2.2 Technologies

In this chapter are described all the technologies used in this thesis work. In particular, first are presented all the software technologies, with a particular focus on IPS software and ROS framework. Then, regarding hardware technologies, UR10 and Iiwa collaborative robots are introduced, which are the ones used in this project.

2.2.1 Software

In the implementation of the solution proposed in this thesis are involved many software technologies in different aspects of the work. First of all, as already said, the final goal is to integrate the alignment functionality into IPS software framework, which is primarily developed using Microsoft Visual Studio IDE and C++ programming language. Therefore, also in the context of this thesis has been produced C++ code, using Visual Studio 2017. Moreover, in order to communicate with robots, and in particular with UR10, the Robot Operating System (ROS) technology has been used to manage the communication on the robot's side. More precisely, the code oriented to ROS has been developed in a Linux environment, always using C++ programming language.

Another important aspect of the communication is the management of the TCP/IP connection between IPS and ROS, which has been implemented using the ZeroMQ socket library [21]. Finally, in order to standardize the messages exchanges the JSON [22] [23] technology has been used.

The following sub-sections go into slightly more detail about IPS and ROS, which are the main software technologies adopted in this project.

IPS – Industrial Path Solutions

Industrial Path Solutions (IPS) [4] is a math-based software tool for automatic verification of assembly feasibility, design of flexible components, motion planning and optimization of multi-robot stations, and simulation of key surface treatment processes.

Figure 17 – IPS logo

This software platform is in-house developed by FCC (Fraunhofer-Chalmers Centre) and it's composed by many different modules which are extensively used in industry by hundreds of engineers.

Follows the list and description of all IPS's modules, explaining their contribution to industrial processes innovation and optimization.

IPS rigid path planner

It's a tool which provides to simulation engineers an easy and time saving way to calculate collision free assembly paths.

Indeed this tool lets simulation engineers import a scene geometry from any CAD system, as, for example, a VRML or JT file. Any object in the scene can be set as a so called planning object, which IPS will find an efficient path for, provided that the object can be freely assembled along a path.

The calculations done by IPS save the engineer a substantial amount of time, which otherwise would have to be put into manual planning of a collision free assembly path.

IPS inspection path planner

It's a tool for automatic programming and optimization of Coordinate Measurement Machines (CMM), which can reduce both commissioning and the stations cycle time.

In particular it provides:

- collision free and optimized inspection task
- automatic programming of station
- visualization of results

IPS robot optimization

It's a tool for automatic task planning, sequencing and line balancing, which enables the user to program robot stations. In particular it provides:

- automatic balancing of tasks between stations and robots
- sequencing, coordination and generation of collision free motions

Proven results: 75% faster commissioning, 25% improved cycle time.

IPS cable simulation

It's a tool for virtual assembly design as well as verification and visualization of flexible parts. Its main capability is the real time calculation of the deformations of cables, hoses and wires of various material types and a variety of cross-section profiles. Forces and moments can be analyzed, the cable length can be optimized, clips can be attached, and motions can be evaluated.

IPS IMMA
This tool implements fast and efficient algorithms for easy evaluation of assembly ergonomics that considers human diversity using a realistic biomechanical model.
In other words it ensures collision free assembly motions for both human and object to be assembled, minimizing biomechanical load and considering human diversity. It's able to manage 82 bone segments connected with joints, in total 162 degrees of freedom.

IPS Virtual Paint – Spray
Implements a realistic simulation of spray painting including all relevant physics. It offers a revolution in spray painting simulation due to its capability of performing accurate simulations in only a few hours on a desktop computer.
This tool it's used by automotive industries in the product preparation phase to fine-tune the robot paths and process parameters, to optimize the process to be more environmentally friendly, more energy and cost efficient, and give a better product quality.

IPS Virtual Paint – Sealing
Detailed simulation of the sealing deposition process taking the complex material rheology and surface flow into account. In combination with the IPS Robot Optimization module the software can be used to optimize a sealing robot cell to minimize the cycle time, improve bead quality and reduce material consumption.

IPS Project Simulation
IPS Projection Simulation is used for virtual product preparation of key surface treatment processes such as spray painting, high-pressure washing and thermal spray. Our customers use the tool to fine-tune robot paths and process parameters, reduce cycle time and improve product quality. In particular this module offers:
- simulation of spray painting and thermal spraying applications for predicting resulting film thickness
- simulation of high-pressure washing processes for improving purity and reducing cycle time
- extremely fast algorithms and powerful analysis tools

IPS IBOFlow

The immersed boundary octree flow solver IBOFlow requires a minimum of preprocessing and the very efficient implementation offers unique possibilities to simulate complex industrial multiphase and multiphysics applications. In particular this tool provides:

- automatic adaptive octree mesh
- unique immersed boundary methods
- novel Volume of Fluids module for multiphase flow
- turbulence models
- particle and sprays module
- fluid-structure interaction simulation
- fluid-heat transfer and fluid-electromagnetics coupling

ROS – Robot Operating System

Robot Operating System (ROS) [24] is a collection of software frameworks which aims to simplify the task of developing robot software.

Figure 18 – ROS logo

Despite the name ROS is not an operating system in the traditional sense of process management and scheduling. Nevertheless, it provides a set of services, typical of an operating system, designed for heterogeneous computer cluster such as: hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management.

ROS has been designed to be language-neutral. Indeed, it currently supports different languages, such as C++, Python and LISP.

ROS is distributed under the terms of the BSD license, which allows the development of both non-commercial and commercial projects.

Currently ROS runs only on Unix-based platform and the software for ROS is primarily tested on Ubuntu and Mac OS systems. The ROS community, however, has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms.

Follows a briefly explanation of the ROS environment, with a description of its elements, architecture and communication protocol. For a better understating of ROS mechanisms refer to the full documentation published on the website [24].

ROS Elements

- **Nodes →** A ROS Node is a process that performs computation and it communicates with other nodes in ROS environment using different communication protocols. This modular structure entails some important benefits like *fault tolerance* (crashes are isolated to individual nodes) and *reduced code complexity.*
- **Master →** The ROS Master provides naming and registration services for all nodes in ROS system. In other words, it acts as a name server for the system.
- **Messages →** A message is a simple data structure, comprising typed fields, that nodes used to communicate each other through *Topics* or *Services.*
- **Topics →** A topic in a named bus that nodes use to exchange messages. Topics are based on a publish/subscribe semantics, which decuples reading and writing of information. In practice a node can send out a message by *publishing* it on a specific topic, while another node, interested on that type of information, can *subscribe* that topic and read the messages.
- **Services →** The publish/subscribe paradigm is very flexible but it's not appropriate for request/reply interactions. For the reason, besides topics, in ROS has been introduced another element, the *Service.* A service is defined by a pair of message, a *request* and a *reply*. The node providing the service offers it with a name and the client uses the service sending a request and waiting for the reply.

ROS Architecture

As already said the ROS Master acts as a name server in the ROS system. Indeed, nodes communicate with the Master to report their registration to topics and services and to receive information about other registered nodes.
Nodes connect to other nodes directly, so the Master only provides lookup information, acting as nameserver.
The most common protocol used in a ROS environment is called TCPROS, which uses standard TCP/IP sockets.

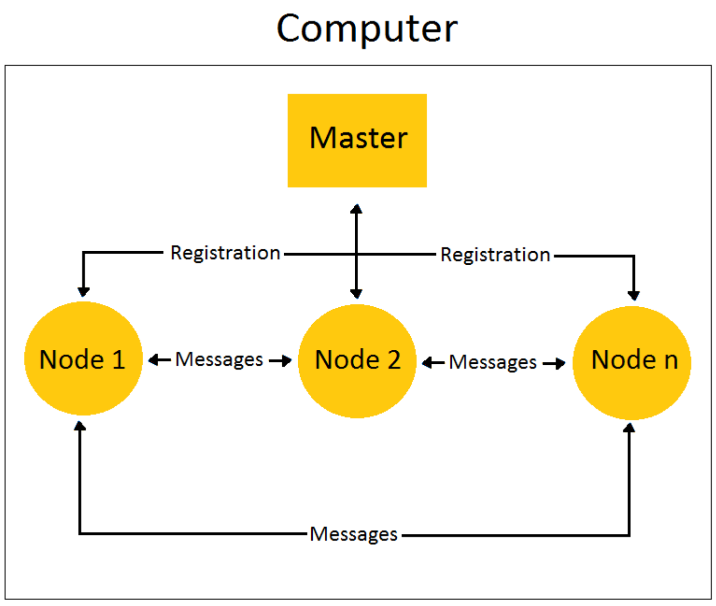


Figure 19 – ROS architecture

ROS Communication Protocols

As previously mentioned there are two main communication protocols in the ROS environment, the *publish/subscribe*, based on topics and the *request/reply,* which uses services.

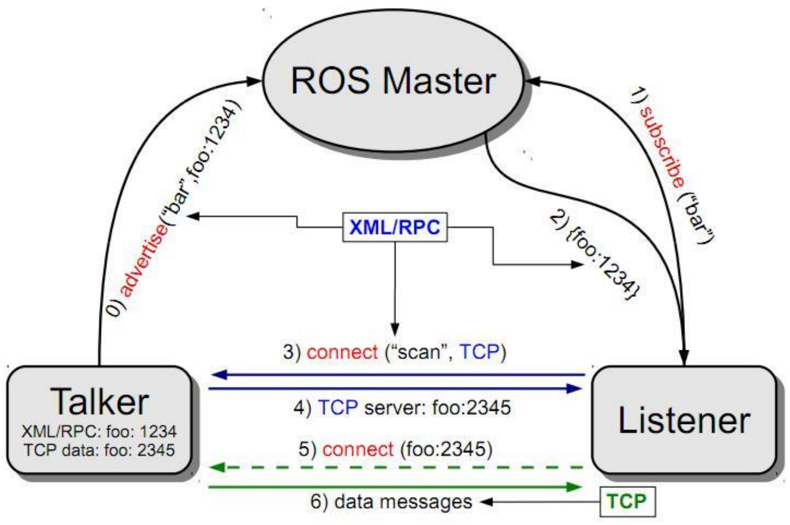


Figure 20 – ROS Topic protocol

Publish and subscribe paradigm is used for continuous data flows. Data can be published and subscribed on a topic at any time independent of any senders/receivers, so basically, it's a many to many, one-way connection. For example, a node publishing the current robot's joints state, once it receives data from the robot it sends them directly to joints

state's topic and the same time nodes subscribing that topic can continuously read that information.
Request and reply method instead, is adopted for remote procedure calls that terminate quickly, for example querying for the joints state at a specific time. In that case the client node should send a request to the node providing the service, which will reply with the current joints state.

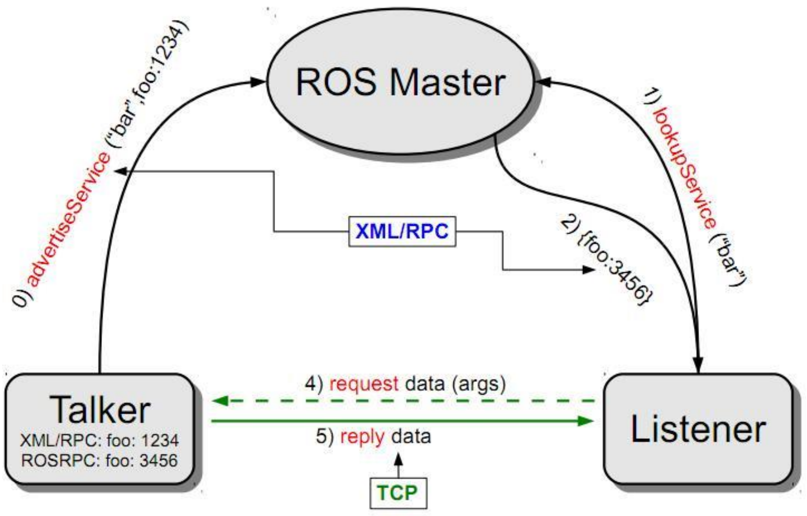
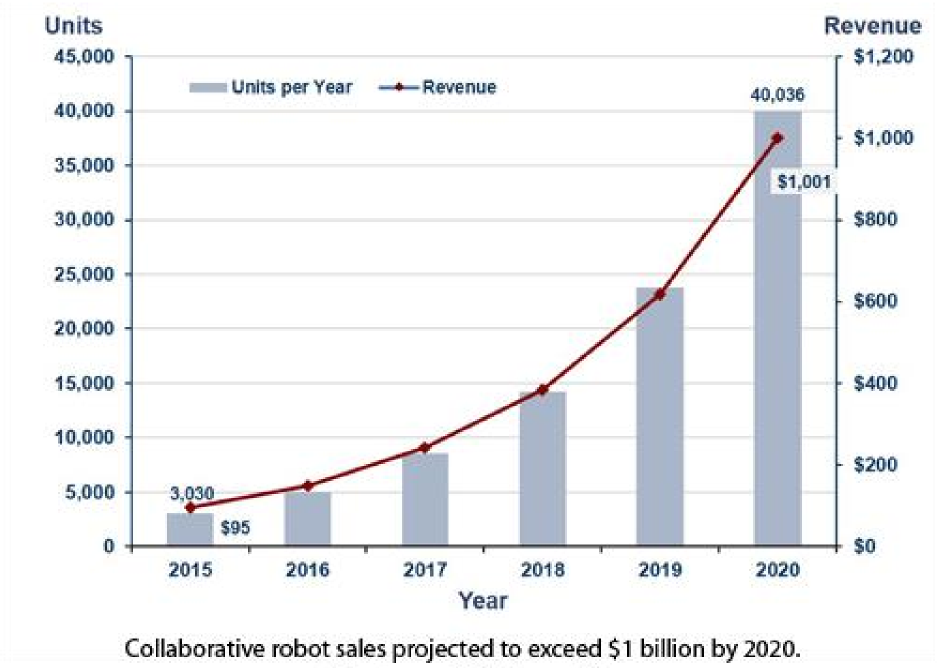


Figure 21 – ROS Service protocol

2.2.2 Hardware

Besides the software technologies described above, in this project is involved a fairly recent technology, increasingly important in the manufactory, the *collaborative robot (cobot)*. In fact, this work addresses the geometry alignment issue with an emphasis on the context of collaborative robots.
The graph in *Figure 26* indicates the estimated raise in collaborative robot sales. These data have been collected by ABI Research [25] and published in 2016 by Tanya M. Anandan, contributing editor of Robotic Industries Association (RIA) [26].



Collaborative robot sales projected to exceed $1 billion by 2020.
(Courtesy of ABI Research)

Figure 22 – Cobots sales estimation

In this sub-section is explained the concept of collaborative robot and the benefits it brings in industry compared with classical industrial robots. Then the two collaborative robots used in this thesis work, the UR10 and Iiwa, are presented.

Collaborative robots (cobots)

Collaborative robots, also named *cobot*, are robots meant and designed to collaborate and share the same workspace with human workers. Indeed, the sensor technology which cobots are equiped with allows them to sense the presence of a human colleague the proximity and safely assist him. This characteristic, besides avoiding that the robots hurt the worker, has also an economic benefit because it makes it that there's no need for safety guarding, saving in this way space and money.

Collaborative robots are also smaller, cheaper and easier to program compared to the classical industrial robots. These qualities make them more flexible then traditional robots, as they can be fairly easily moved and reporgrammed. On the the other hand, cobots have limited reach, payload, speed and accuracy, and for this reason they're not suited for large scale production, where traditional industrial robots are still preferable.

These differences are outlined in the figure below, where it's shown how cobots are the middle level between manual work and fully automated production. They aim to fill that gap, carrying out that range of works not addressable with classical automation and dangerous or arduous for human workers.

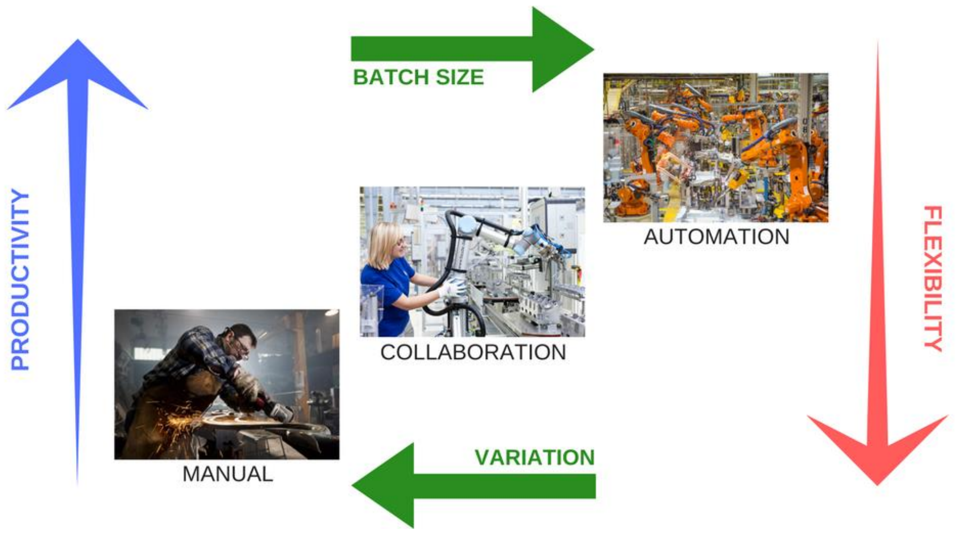


Figure 23 – Collaborative robotics positioning

Some of the most common applications for cobots are:

- **Packaging** and **Pick&Place** tasks, which are repetitive and humdrum works for humans and this can lead to oversights or strain due to the repeated movements.
- **Material handling** operations, for which the cobot could help the worker in lifting weights
- **Machine tending**, which is the activity of assisting an automatic machine during its work, for example changing tools or providing the raw material. This task is monotonous and tiring for humans, since it requires to stand for hours in front of a machine.
- **Quality inspection** operations, which means inspect and catch measures of parts. Using cobots for this purpose can result in higer quality and fairly fast inspections. Notice that the usage of cobots made in this project falls in this category of application.

Universal Robots - UR10

Universal Robots [27] is a company, based in Odense (Denmark), producing small and flexible collaborative robotic arms. It was founded in 2005, with the goal of making robot technology accessible to small and medium-sized enterprises.
Universal Robots is currently producing three different models of collaborative robots: UR3, UR5 and UR10.
All these three robots have six articulation points and a wide scope of flexibility, designed to mimic the range of motion of a human arm. The main differences between the three models are size and payload, as shown in the table below.

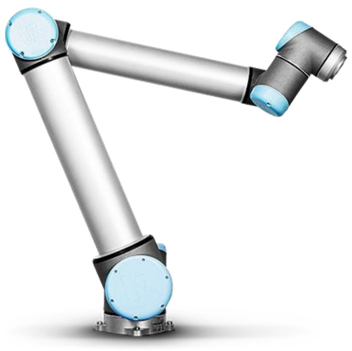


Figure 24 – UR 10

Model	DOF	Radius	Payload	Weight
UR3	6	500 mm	3 kg	11 kg
UR5	6	850 mm	5 kg	18.4 kg
UR10	6	1300 mm	10 kg	28.9 kg

KUKA - LBR iiwa

KUKA is a German corporation producing industrial robots and solutions for factory automation. KUKA was founded in 1898 by Johann Joseph Keller and Jakob Knappich in Augsburg, with the aim of producing affordable illumination for houses and streets, but soon the company was expanded to other products (e.g. welding technology).
In 1973 KUKA developed FAMULUS, its first industrial robot, orienting in this way the company to the robot manufacturing field. Today KUKA is one of the world's leading suppliers of robot technology and plant and systems engineering and, in 2013, released its first collaborative robot, the *LBR iiwa*.
Currently KUKA is producing two versions of LBR iiwa: LBR iiwa 7 and LBR iiwa 14. They both have 7 controlled axis and similar maximum reachable distance. The main difference between the two, as shown in the table below, is in the payload.

Figure 25 – Iiwa 7

Model	DOF	Radius	Payload	Weight
Iiwa 7	7	800 mm	7 kg	22 kg
Iiwa 14	7	820 mm	14 kg	30 kg

2.3 Implementation

In this section are detailed all the implementation aspects of the solution presented here. In particular, is first presented the dialog provided to the user to manage the inspection phase, then the implementation details of the PSA algorithm are exposed and commented. Finally, is presented an exhaustive description of the communication between IPS and the collaborative robots, for both Iiwa and UR10.

2.3.1 Geometry Alignment Dialog

The user dialog for the Geometry Alignment tool (*Figure 30*) has been developed using *FOX* [28], which is the C++ based Toolkit used in the IPS environment for developing Graphical User Interfaces.

This dialog is divided in four sections: *Input*, *Connection*, *Communication* and *Data*.

The *Input* section allows the user to select in IPS the robot and the tool he wants to use for the alignment process and the product he wants to align.

In the *Connection* frame the user has to insert the IP address of the robot and specify also two port numbers, one for instructions communication channel (*Port*) and the other one for hardware state publishing and reading channel (*HS port*).

The *Communication* frame is where the user receives human readable messages from the application. These messages can indicate the process status, signal the errors or suggest actions to the user, guiding him to the correct workflow.

Finally, in the *Data* section all the inspection points collected are shown and it’s possible for the user to selectively remove one or more inspection points, selecting them from the list and clicking on the red icon.

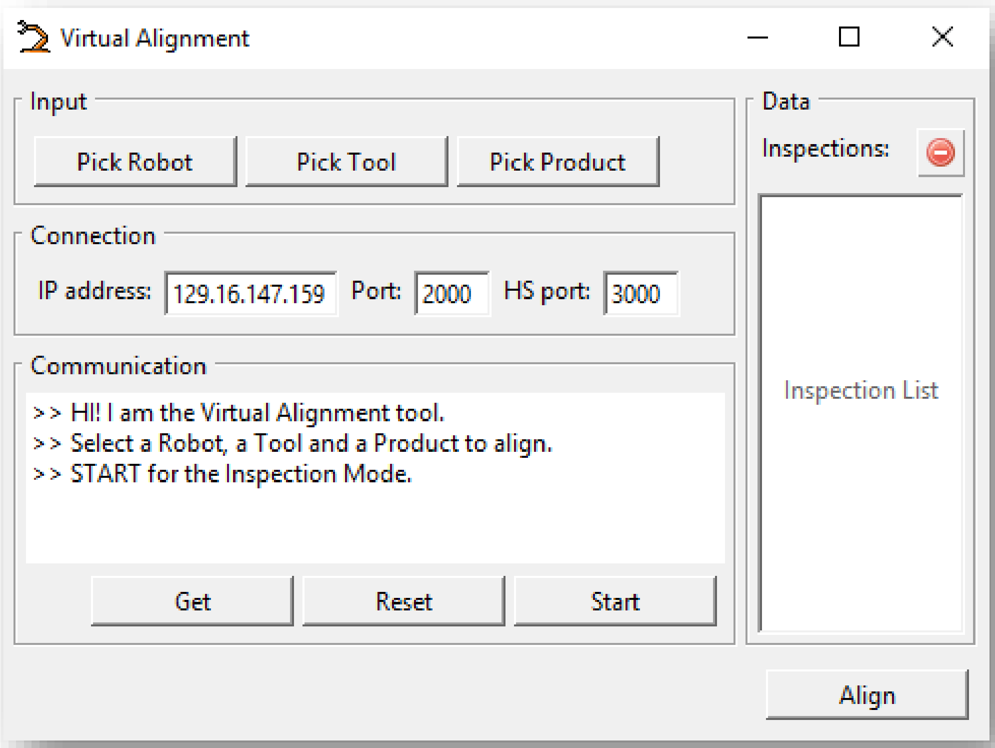
The GUI has also four input buttons: *Start, Reset, Get* and *Align.*

The *Start* button allows the user to start the inspection process through which he can collect the inspection points. The start phase will be successful only if the user has correctly selected the robot, the tool and the product.

The *Reset* button can be used to stop the ongoing inspection process and reset all the values. So, basically, it allows the user to re-start the inspection phase, discarding all the inspection points taken till that moment.

The *Get* button allows the user to get an inspection point. When the robot is in the desired position the user can press *Get* to stick a reference frame, representing the inspection point, in that position.

Finally, when the user has collected a consistent set of inspection points, he can press the *Align* button to launch the alignment process.



IP address: 129.16.147.159 Port: 2000 HS port: 3000

>> HI! I am the Virtual Alignment tool.
>> Select a Robot, a Tool and a Product to align.
>> START for the Inspection Mode.

Figure 26 – Geometry Alignment dialog

2.3.2 Point Set Alignment

The *PointSetAlignment* class implements the PSA algorithm described in section 2.1.6. In particular its constructor takes as input the set of inspections gathered by the collaborative robot and the object to align, selected through the dialog, and stores those data in member variables.

The core of the alignment phase is implemented in the *align* method, which exploits the data members saved by the constructor as input values. Its structure is described by the following lines of pseudo-code.

```
while (!termination(newPos) > threshold) {
   // find closest points on object
   findPoints(inspectionPoints, object, objPoints);
   // SVD registration
   newPos = SVDRegistration(inspectionPoints, objPoints, weights);
   // loop
   update();
}
```

This function is based on a *while* loop, which confers to the method the iterative form underlying the PSA algorithm. In particular, the *align* function loops until the termination criterion is triggered.

Analysing the content of the loop, the first step of each iteration is the *findPoints* function. This method finds the set of points, belonging to the object, which are closest to the inspections. In particular, for each inspection the function computes the distance from the object and stores in *objPoints* the correspondent closest point.

```
for each iPoint in inspectionPoints {
   // compute the distance from the object
   measure = createMeasure(iPoint, object);
   measure->getDistance(distance, from, to);
   // save the closest point on object
   objPoints.push_back(to);
}
```

The following step is the actual registration. In particular, the following pseudo-code depicts the *SVDRegistration*'s structure, which is the function implementing the SVD method analysed in section 2.1.2.

```
SVDRegistration(inspectionPoints, objPoints, weights) {
   // compute centers of mass
   for (int i = 0; i < nPoints; i++) {
      ui += inspectionPoints[i];
      up += objPoints[i];
   }
   ui /= nPoints;
   up /= nPoints;
   // compute H matrix
   for (int i = 0; i < nPoints; i++) {
      I(i) = inspectionPoints[i] - ui;
      P(i) = productPoints[i] - up;
   }
   H = P*I.transpose();
   // SVD
   Eigen::SVD(H, &U, &V);
   // compute optimal rotation and translation
   R = V*U.transpose();
   t = ui - upW.t;
   // apply complete transformation
   newPos = object->applyTransformation(R, t);
   return newPos;
}
```

As first step of this function, the centres of mass are computed, then the *H* matrix is derived and decomposed using the *SVD* method implemented in the Eigen library (Eigen::SVD). Finally, *U* and *V* matrices are used to derive the optimal transformation, which is applied to object. This function returns the new object's position, resulting from the application of the transformation.

Notice that, the position returned from *SVDRegistration* is stored in the *newPos* variable and it's then passed as input to the *termination* function, which implements the stopping criterion explained in *2.1.4*.

```
bool termination(newPos) {
   if (terminationArray.Size() < nPrevious) {
      // first iteration
      terminationArray.PushBack(newPos);
      return false;
   }
   for each position in terminationArray {
      if (computeError(newPos, position) > convergence) {
         // no convergence --> no termination
         terminationArray.Delete(0);
         terminationArray.push_back(newPos);
         return false;
      }
   }
   // convergence --> termination
   return true; }
```

The lines above describe the *termination* function, which triggers the end of the algorithm. This method conserves an array of *nPrevious* positions and then, at each iteration, it receives the current object's position. This input value is directly pushed in the array if this last hasn't still reached the desired size, otherwise the method proceeds to the criterion check. In practice, it scans the array and for each previous position it computes the displacement error with respect to the current position, if all the resulting values are lower the *convergence* this function returns true, which means that the termination will be triggered.

The final step of each iteration is the execution of the *update* function, which increments the iterations counter and clears the data structures, preparing the algorithm to the next cycle.

2.3.3 Communication IPS-Cobot

Besides the aligning phase, another important aspect of the implemented solution is the communication between IPS software and the collaborative robot. This functionality is essential for the inspection phase, indeed, IPS has to know the exact position of the robot's tool in order to stick a reference frame in its virtual environment corresponding to the inspection acquired. The designing choice adopted here is to have a continuous TPC/IP communication between the software framework and the cobot, which allows IPS to known at every time the exact pose of the collaborative robot. This functionality is called *Digital Twin* because provides a twin of the real collaborative robot in IPS's virtual environment, which mimics exactly the movements of the cobot.

In particular, once the system is up and running, the robot waits for requests coming from IPS, which at some point, when the inspection phase starts, sends a message to it, asking to be updated on its position. In response to this request, the cobot starts continually sending messages containing its joint values, from which IPS can derive the pose of the robot. Similarly, at the end of the inspection, IPS will send another message asking to stop the flow of information and the cobot will stop sending messages, returning in the initial waiting state.

Notice that the TCP/IP communication is managed through ZeroMQ [21] software library, which provides API oriented to simplify sockets programming issues.

On the IPS side this communication is managed through a template class called *DTHardwareMirror*. The aim of this class is to provide a common interface handling the communication with collaborative robots in IPS, which is in a way independent from

the specific cobot model involved. The two main methods implemented in this template class are *activateMirror* and *deactivateMirror*, which manage the procedure of starting and stopping the flow of joint values from the robot to IPS. These functions are then inherited by the classes which refers to a specific cobot, for example in this project have implemented the *DTRosHardareMirror* and the *DTIiwaHardwareMirror* classes. The main advantaged of this design is that it's easily extendable to support the communication with other robots.
On the collaborative robot side, instead, the scenarios can be completely different based on the model involved. In general, environment and tools provided by cobots manufacturers are very different and therefore the communication design has to be customized for the specific case.

Communication between UR10 and IPS

The communication between UR10 robot and IPS is based ROS. It runs on a Linux machine and acts as intermediary, managing the messages exchange on the UR10's side. On ROS has been developed a ROS node called *ROS to IPS link*, which can be execute through the bash command: `rosrun ips_link ROStoIPSlink.`
When executed this node instantiates a *ROS Instruction Interface*, which is actually a ROS service listening (on port 2000) for requests coming from IPS. At some point, when a request of mirroring comes, this service triggers the *Hardware State Publisher* which start sending the joint values coming from the robot to IPS. On the contrary, when a stop request comes the interface blocks the publishing of joint states.
The *Hardware State Publisher* is called like this because it acts as a publisher for IPS. In particular, it subscribes the ROS topic called *joint_states* and reads the values published by the UR10 robot on that topic. Then, it serializes those values in a message and sends it to IPS (on port 3000).
The following lines are the pseudo-code of the *ROS Instruction Interface*, which summarizes the explained concepts.

```
while(!stop){
   // wait for request
   request = mInstructionInterface->receive();
   // process request
   if(request.type == INSTRUCTION){
      if(request.command == START){
         // start publishing the hardware state
         HwStatePublisher->startMirroring();
      } else if(request.command == STOP){
         // stop publishing hardware state
         HwStatePublisher->stopMirroring();
      } else{
         // reply ERROR
         mInstructionInterface->send(error_msg);
         continue;
      }
   } else{
      // reply ERROR
      mInstructionInterface->send(error_msg);
      continue;
   }
   // reply OK
   mInstructionInterface->send(ok_msg);
}
```

The schema depicted in *Figure 31* summarizes all concepts explained about the communication between IPS and the UR10 robot. Notice that, the *ROS Messages Module* is devoted to the serialization and deserialization of messages, while the *ROS ZeroMQ Interface* is the module that manages the TCP/IP sockets, using the ZeroMQ library.

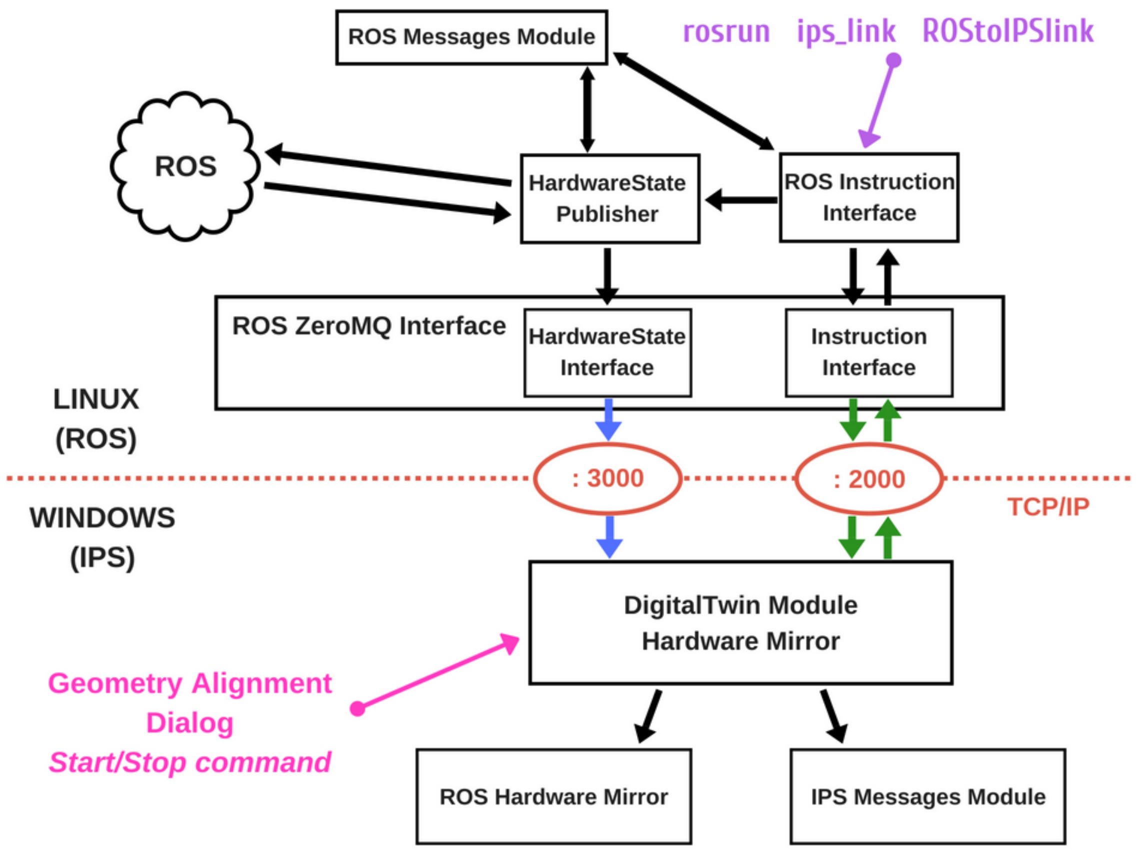


Figure 27 – IPS to ROS communication scheme

Communication between Iiwa and IPS

Differently from the communication with UR10, with Iiwa robot ROS is not involved. In fact, in this case, the communication is performed directly between IPS and the Iiwa's virtual controller, which has been developed by Christian Larsen in his thesis work [29]. In *Figure 32* shows the communication scheme presented in [29], which has been exploited in this project. The structure is similar to the one adopted with the UR10, indeed there is a request/reply module, called *Instruction Distributor Thread,* which manages requests coming from external applications (e.g. IPS) and a publish/subscribe thread *(FRI Thread)*, used to exchange hardware states information (e.g. joint values). The *iiwa State Change Thread,* instead, manages errors or unexpected behaviours on the Iiwa's side.

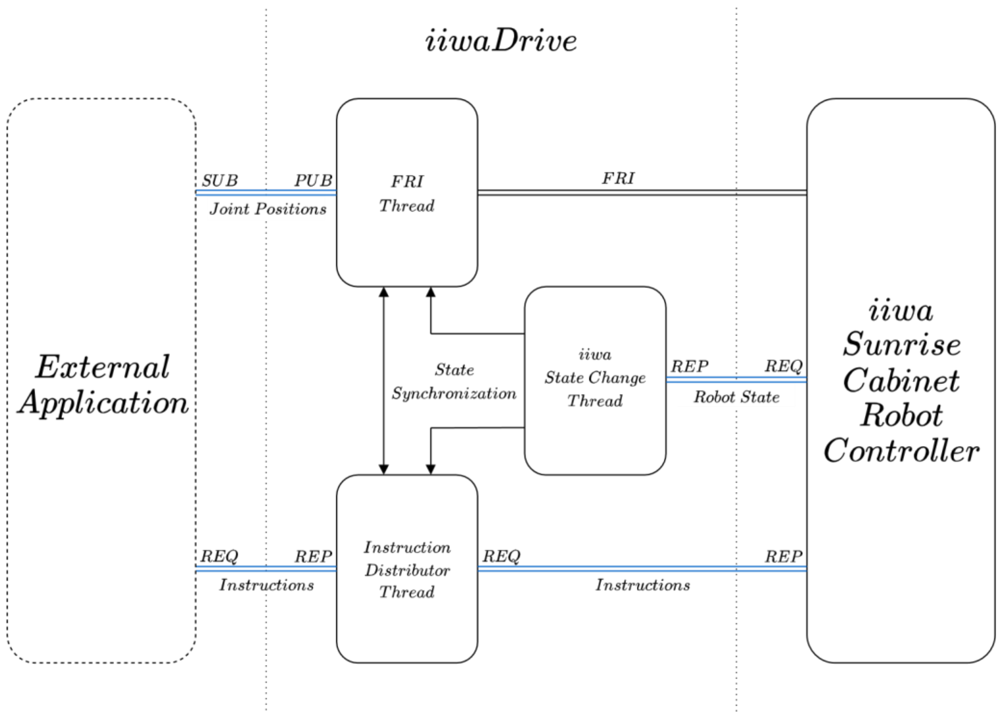


Figure 28 – IPS to Iiwa communication scheme

JSON Messages

All the messages transferred between the two systems are encoded using the JSON format, which provides portability and it’s easy to manage. More precisely, a series of standard messages have been defined, which are independent from the cobot model IPS is talking to. These standard JSON messages can be divided in *Instruction messages* and *Hardware State messages*. The instruction messages are used for requests and replies. In particular, there are *START* and *STOP* messages that are sent from IPS to the cobot in order to start or stop the flow of joint values information.

START message

```
{"header", {
   {"from", "ips"}
   {"to", "ros"}
   {"type", "instruction"}
}},
{"body", {
   {"command", "start"}
   {"content", "hwState"}
}}
```

STOP message

```
{"header", {
   {"from", "ips"}
   {"to", "ros"}
   {"type", "instruction"}
}},
{"body", {
   {"command", "stop"}
   {"content", "hwState"}
}}
```

The *OK* and *ERROR* messages instead, are sent from the robot to IPS as a reply to its request. In fact, they are used to notify the smooth running of the operation or a problem during execution.

OK message

```
{"header", {
   {"from", "ros"}
   {"to", "ips"}
   {"type", "instruction"}
}},
{"body", {
   {"command", "ack"}
   {"content", "ok"}
}}
```

ERROR message

```
{"header", {
   {"from", "ros"}
   {"to", "ips"}
   {"type", "instruction"}
}},
{"body", {
   {"command", "ack"}
   {"content", "error"}
}}
```

The other type of JSON message involved is the Hardware State (HS) one. This message is sent by the robot to IPS in order to deliver information about the current joint state. More precisely, when IPS send the request and the robot starts a continuous flow of HS messages at high frequency, each of them containing the current joint values.

Hardware State message

```
{"header", {
   {"from", "ros"}
   {"to", "ips"}
   {"type", "hsmsg"}
}},
{"body", {
   {"jointValues", {js.position[0], js.position[1],
                    js.position[2], js.position[3],
                    js.position[4], js.position[5]}
   }
}}
```

Notice that, in the message's structures reported here, the cobot in the fields *from* or *to* is identified with *ros*. That's because these are the messages exchanged between IPS and the UR10 robot, which relies on ROS to manage the communication and therefore ROS is the recipient of those messages. However, nothing really changes in the messages exchanged with the Iiwa robot, the only difference is that, instead of *ros*, the content of those fields will be *IiwaDrive*.

2.3.4 Tests

In order to test, in a systematic way, the implemented algorithm and collect data on its behaviour, an *AlignmentTests* class has been developed.

The idea behind this class is to implement a testing method which, starting from the ideal situation where the object is perfectly aligned with inspections (*Figure 33*), applies a random transformation that distorts the scenario, misaligning the geometry, and then it runs the algorithm to see if the object can return to the correct position.

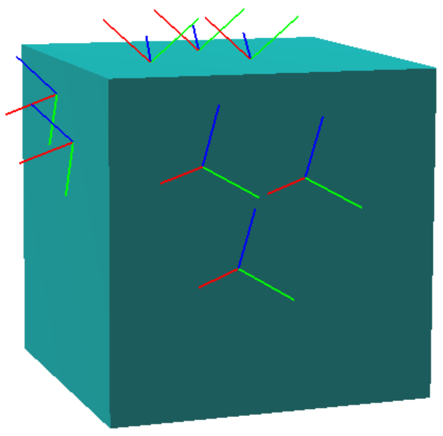


Figure 29 – Aligned geometry

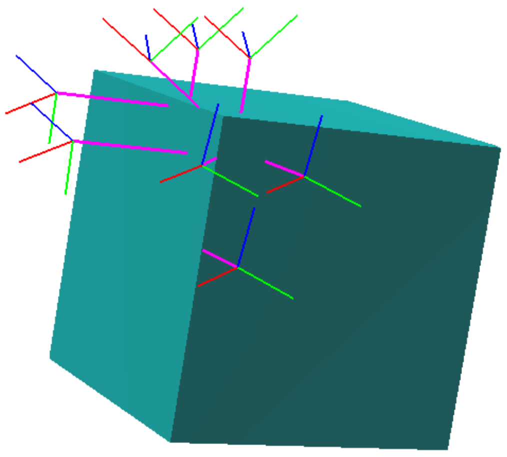


Figure 30 – Misaligned geometry

The structure of this method is summarized in the following lines of pseudo-code.

```
for (int i = 0; i < testNumber; i++) {
   // generate random transformation
   RandT.t = tRandGeneration(delta);
   RandT.R = RRandGeneration(delta);
   // apply transformation
   object->applyT(RandT);
   // alignment test
   errorsArray = object->align(inspectionList, product);
   // restore initial position
   object->setPosition(originalPos);
}
```

Observe that, the random transformation is computed in two steps, first the translational part, through *tRandGeneration*, and then the rotational component, using *RRandGeneration.* In both cases, the functions take as input parameter *delta* (δ), which represents the upper-bound for the displacement and it's manually entered by the tester.

Analysing more formally the generation of those random components, it's possible to define the translation as follows

$$t = \beta \cdot \mu \tag{27}$$

where μ is a random unit vector, which confers an arbitrary direction to the translation, while β represents the module of the translational vector and its value is a random variable, uniformly distributed between 0 and δ.

$$\beta \sim U[0, \delta] \tag{28}$$

The random rotation instead, can be expressed in the exponential representation with the following notation

$$R = e^{\alpha \hat{\mu}} \tag{29}$$

Notice that $\hat{\mu}$ is the 3×3 matrix composed in this way

$$\hat{\mu} = \begin{pmatrix} 0 & -\mu_3 & \mu_2 \\ \mu_3 & 0 & -\mu_1 \\ -\mu_2 & \mu_1 & 0 \end{pmatrix} \tag{30}$$

where μ_1, μ_2 and μ_3 are components of the random unit vector $\mu = [\mu_1, \mu_2, \mu_3]$, while α is a random variable uniformly distributed between 0 and $\delta/_r$, being r the object's radius.

$$\alpha \sim U\left[0, \delta/_r\right] \tag{31}$$

Another important aspect to remark of that testing method is that, the output of the alignment process is *errorsArray*, which is a vector containing the displacement errors for all the algorithm's iterations and it represents a useful measure to monitor the alignment performance.

In particular, this displacement error is the difference between the current object's position and the correct one and it's computed through $d(T, T^*) = \sqrt{|t - t^*|^2 + (r \cdot d\alpha)^2}$, (22).

2.4 Results

In order to analyse the results obtained with the solution presented, it's necessary to focus on two different aspects of the process, the inspection phase and alignment phase.

2.4.1 Inspection phase

The evaluation of the inspection phase has been done through tests on real collaborative robots. In particular have been performed trials with the Iiwa and UR10 cobots oriented to check the implemented communication with IPS and verify the efficiency of data collection process. *Figure 36* documents the tests performed using the Iiwa cobot on a real geometry. This cell is used in the SWEDEMO project, financed by VINNOVA, and it has been kindly made available for these tests.

The result of this inspection phase is the geometry shown in *Figure 36*, which depicts the IPS scene representing the object and the inspection taken using the Iiwa.

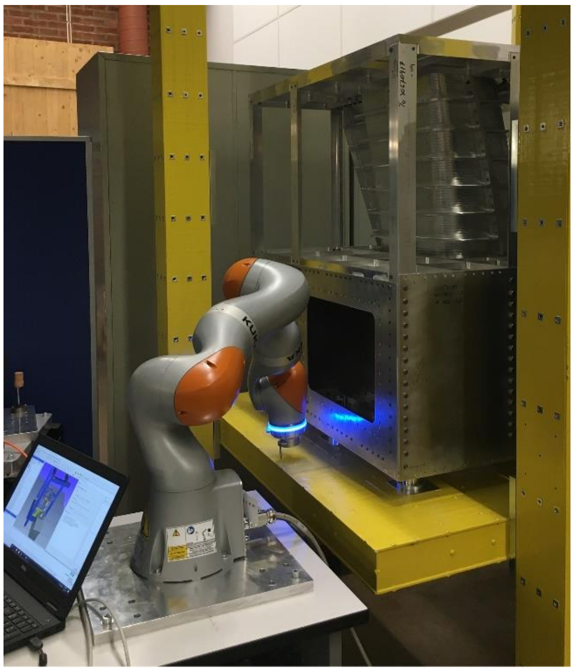


Figure 32 – Iiwa inspection test

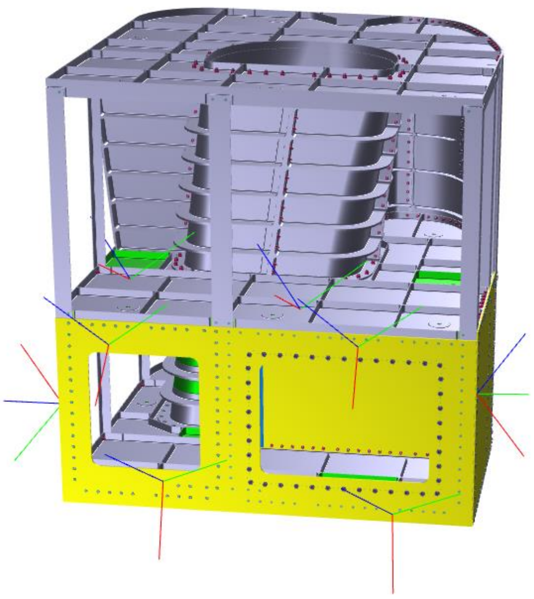


Figure 31 – Inspection result

2.4.2 Alignment phase

In order to verify the performances of the alignment algorithm, the implemented *AlignmentTests* class has been used, applying the testing method on the geometry showed below and with different δ values.

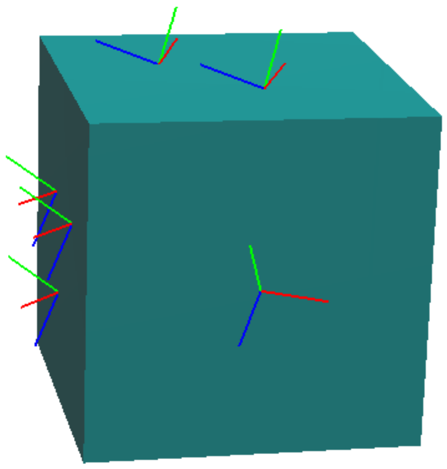


Figure 33 – 6 inspections geometry

Notice that, in the figure are present six inspection frames, which is the minimum number of inspections necessary to identify the position of an object in space. In particular, are discussed here the results obtained with δ varying from 50mm to 1000mm.

When δ is sufficiently small (50mm–200mm) the algorithm's behaviour can be considered ideal, indeed all the tests converge monotonically to the optimal solution. This scenario is showed in *Figure 38*, where the results obtained in 50 tests with δ*=100mm* are displayed.

On the contrary, when δ assumes greater values the algorithm gets stuck in global minimum and it can't find the global optimum. The graph depicted in *Figure 39* underlies these difficulties, indeed 23 tests over 50 don't converge to the optimal solution.

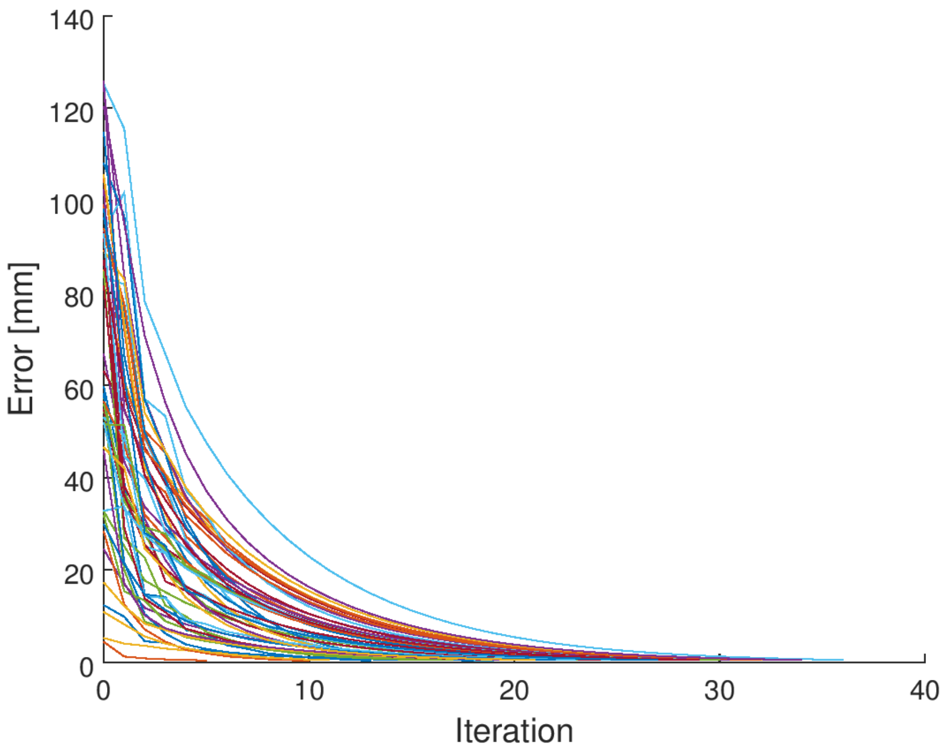


Figure 34 – Tests results with δ = 100mm

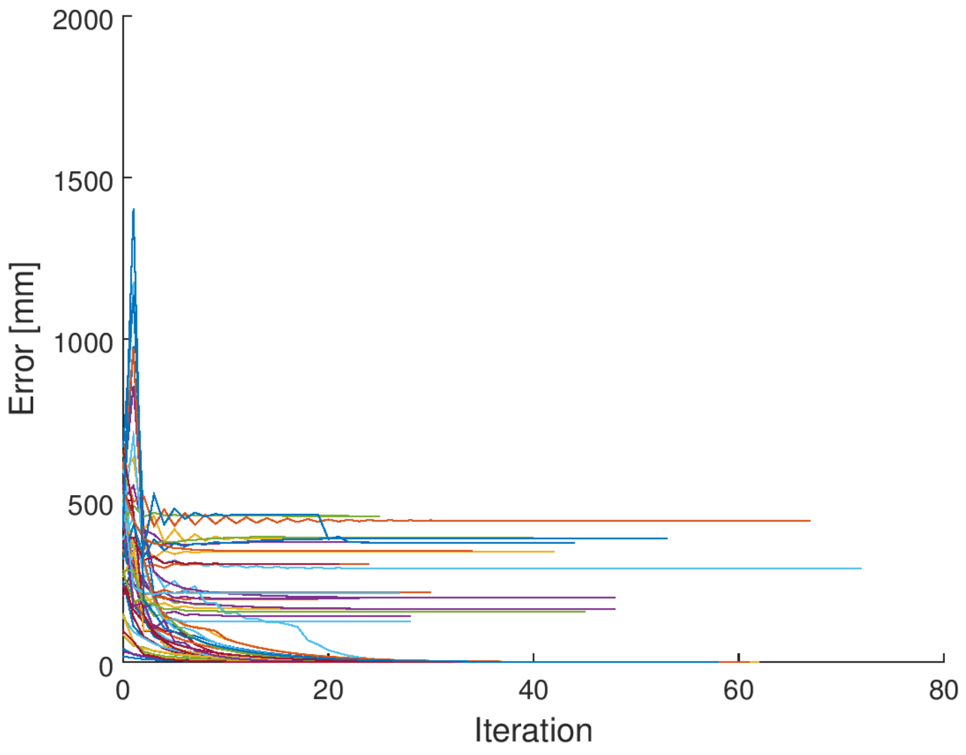


Figure 35 – Test results with δ = 500mm

The table below summarizes the results obtained in the tests.

Delta (mm)	Succeed	Fail	Avg. Iterations	Avg. Error (mm)	Avg. Time (s)
50	50	0	36	$5{,}34 \cdot 10^{-2}$	$2 \cdot 10^{-3}$
100	50	0	41	$5{,}34 \cdot 10^{-2}$	$2 \cdot 10^{-3}$
200	45	5	50	17,64	$2{,}9 \cdot 10^{-3}$
500	27	23	74	142,8	$4{,}5 \cdot 10^{-3}$
1000	13	37	88	222,3	$5{,}6 \cdot 10^{-3}$

Therefore, as these results demonstrate, the developed solution works well when only the calibration of the object's position is needed (i.e. when the object's position is close enough to the real one), while it has some problems in dealing with bigger displacements.

An immediate solution to improve the algorithm performance is to enforce the input data set by gathering more inspections. For example, starting from the geometry in *Figure 37* and taking three more inspections, one for each cube face not already considered, the test ran with *δ=500mm* and *δ=1000mm* gave better results, even if they're still not the ideal ones.

The table below summarizes the results obtained in this case (9 inspections).

Delta (mm)	Succeed	Fail	Avg. Iterations	Avg. Error (mm)	Avg. Time (s)
50	50	0	23	$5{,}18 \cdot 10^{-2}$	$1{,}7 \cdot 10^{-3}$
100	50	0	27	$5{,}24 \cdot 10^{-2}$	$2 \cdot 10^{-3}$
200	50	0	29	$5{,}11 \cdot 10^{-2}$	$2{,}3 \cdot 10^{-3}$
500	42	8	45	34,65	$3{,}5 \cdot 10^{-3}$
1000	40	10	51	45,1	$4{,}2 \cdot 10^{-3}$

2.5 Conclusions

In the view of the obtained results, the solution presented here appears to be promising it comes to calibrate the virtual geometry. The fact that results degenerate for greater displacements shall not constitute an big obstacle to the application of this method, indeed, in real cases, it's quite unlikely that the virtual model differs very much from the real environment and therefore an effective calibration method could be the answer in the majority of cases.

The real restriction to the practical application of this solution as it has been presented here, is that its behaviour gets drastically worse when it comes to calibrate the position of complex geometry, as the one shown in *Figure 35.* In order to overcome these problems a possible improvements would be to exploit directional information of inspections as explained in *2.1.6*. Another possible improvement, which could help to solve problems in dealing with complex geometry, would be to introduce correspondences in the inspection phase. In practice, it could be useful to decide by convention a set of key features belonging to the object's geometry (i.e. corners, edges, holes) and then gather inspections in correspondence of that features. In this way the algorithm could know in advance the correspondences between the inspections and the virtual geometry, simplifying considerably the problem.

Besides these upgrades, oriented to enhance the alignment phase, some revisions could also be done to streamline the inspection phase. For example, instead of having the user who manually guides the cobot to touch the object's surface, an *automatic inspection mode* could be implemented. Adopting this mechanism the collaborative robot would be programmed to automatically collect a predefined set of inspections without the direct involvement of the human worker, who would just monitor the operation. Of course this improvement requires to mount a force torque sensor on the object's tool which allows to detect the contact with the object.

3. Geometry Alignment using Point Cloud

This part debates about the same alignment problem explained in the first part of the project but starting from a different type of input data. Indeed, instead of dealing with a set of inspection points, collected using a collaborative robot, this alignment method takes as input data a point cloud modelling the physical object.
Follows a brief introduction on point cloud data structure. In order to have a deeper understanding of the argument it's suggested to refer to the mentioned sources.

Point cloud introduction

In relation to the Weinmann's work [30], a point cloud can be defined as a set of data points in space. More precisely a point cloud is a data structure used to represent a collection of multidimensional points. This data structure is usually adopted as a representation of three-dimensional objects in many and different fields including manufacturing, medical science and various virtual reality applications.
In the context of this thesis the term point cloud is used to indicate a collection of 3D points, characterized by spatial XYZ-coordinates representing the surface of the physical object to align. More in general it's possible to assign additional attributes to each point in a point cloud, such as intensity, colour, thermal information and other specific properties related to orientation and scale.
Point clouds are generally produced by 3D scanning machines, which gather point measurements from real-world objects. There is a wide variety of 3D scanners and acquisition systems based on different principles but, as asserted in [30], the optical methods have proven to be particularly advantageous since they offer an efficient and touchless acquisition of 3D structures.
Moreover, it's possible to distinguish two main categories of optical 3D acquisition mechanisms, the *active* and the *passive* techniques.
The idea behind active techniques is to emit electromagnetic radiations, either in the visible spectrum or in the form of infrared laser light, and record information about reflection or radiation passing through objects in the scene. These collected data describe the space surrounding the scanner and allow to virtually reconstruct the environment.
Figure 41 depicts the principle of operation of active techniques, while in *Figure 40* is shown an arm-laser scanner, which is an example of active scanner.

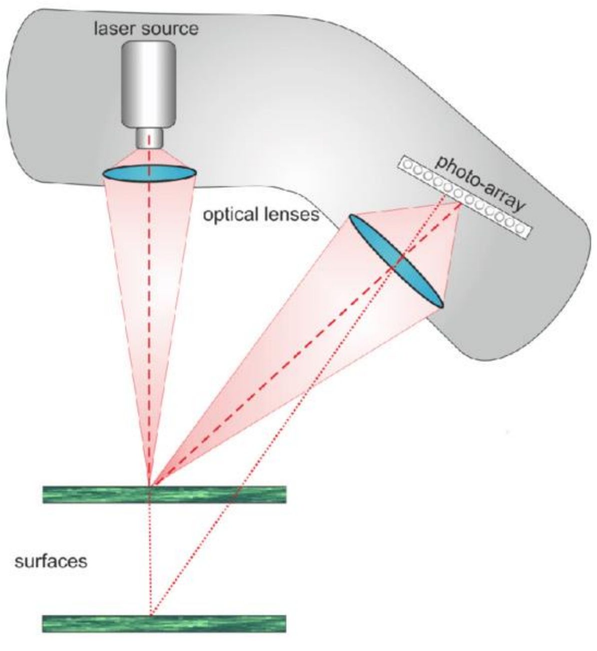


Figure 37 – Active scanning tecnhique

Figure 36 – 3D active scanner

Passive techniques instead, rely just on radiometric information. They don't emit any kind of radiation because they only detect the reflected ambient radiation and for this reason this type of acquisition methods are usually cheaper, since they just need simple digital cameras. However, these techniques require a sufficient ambient lighting condition in order to collect information.

Of course, depending on the involved acquisition technique and the used device, the acquired 3D point cloud data may be corrupted with more or less noise. In fact, the design of the acquisition system may have a significant impact on the quality of the measurements. However, that is not the only factor affecting the measure accuracy, indeed, also the atmospheric and environmental conditions, the shape of the scanned geometry and its characteristic, such as reflectivity and surface roughness can have a great influence on the final result.

In *Figure 42* is shown the point cloud representing the ABB robot IRB 1600, which has been used to test the solution presented in this chapter. This point cloud has been gathered using active scanning technology provided by ATS [31].

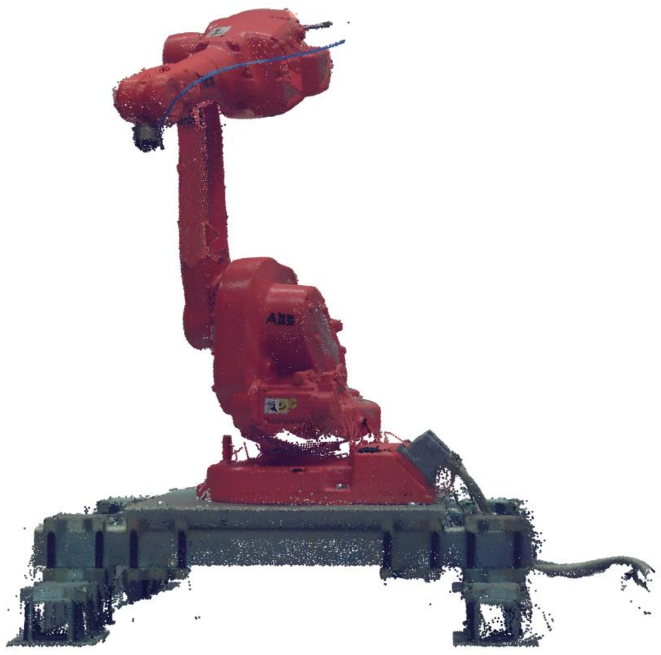


Figure 38 – ABB IRB1600 point cloud

As already said, in this chapter is described an alternative solution to the alignment problem using point cloud, therefore, instead of having an inspection phase, information on object's position will be extracted directly from the point cloud.

In particular, first of all is presented here an overview on the algorithms and methods used to solve the problem. Follows a description of the software technologies used in this part and finally the solution's implementation is exposed. The last sections of this chapter are devoted to the results presentation and discussion and to draw the conclusions.

3.1 Algorithms

This chapter explains all the algorithms and methodologies used to solve the alignment problem using point clouds, and it presents some of the solutions, existing in literature, that somehow inspired this work or that constitute an alternative to the methods adopted here.

First of all, it's important to notice that working with point clouds entails a further obstacle which was not present in the previous case: selecting those points, inside the point cloud, that are actually useful for the alignment.

Indeed, while the inspection points taken through the collaborative robot, assuming an inspection phase correctly performed by user, correspond to real point on the object's surface, with point cloud not all points are significative or strictly related with the object to align. Considering, for example, a point cloud which represents a manufacturing cell and a robot inside that cell, to align the robot's virtual geometry it's necessary, first of all, to identify the point-cloud's area representing the robot. Moreover, even considering a simpler case, outliers due to scanning errors and noise have to be ignored in the aligning phase.

Therefore, an algorithm that aims to solve the alignment problem using a point cloud has to implement a very effective keypoint detecting and filtering phase before operating the alignment. The idea, that underlies the solution proposed in this work, is to derive from the point cloud a set of inspection points and then perform the alignment with the same algorithm already used in the first part, with just some minor modifications. In other words, instead of designing a completely different algorithm, the choice made is to readapt the already developed algorithm to fit this new problem.

In the following sections are first presented the methodologies used to implement the information extraction phase, in particular the keypoint detector and the filters adopted are described, and then the iterative algorithm designed to solve the alignment problem is explained.

3.1.1 Keypoint detection

In an image or, like in this case, in a point cloud it's possible to define keypoints. A keypoint is a point which neighbourhood is different from neighbourhoods of other points and thus, it's particularly important in describing the point cloud's geometry.

Efficient keypoints detection phase inside a point cloud is one of the major issues in computer vision and image processing, and therefore it's a well-studied problem in literature.

This interest in keypoints detection led to the design of many methods to solve the problem, which differ in complexity and performances. Harris and Stephens, for example, proposed, in 1988, one of the earliest corner and edge detector method [32]. SUSAN [33] and FAST [34] methods were later designed with the same purpose but adopting different approach. FAST, in particular, is real-time applications oriented, in fact it aims to reduce the computational weight with respect to Harris and SUSAN methods, which are not suitable for real-time applications.

SURF [35] and SIFT [36] instead, are two of the most used methods for features detection and feature description. This means that they, besides finding interesting points inside a point cloud, describe also the behaviour of the geometry around that point.

All these methods play an important role in computer vision field, in particular they've been used in tasks as object recognition, image registration, classification and 3D reconstruction. However, for the case of application treated in this project these methods are ill-suited because, although they could be very precise in finding significative points, they require computational overhead that, in the end, doesn't worth to spend.

For this reason, the decision made in this work is to adopt another approach to the problem: instead searching for few but very significative points, the strategy embraced is to select more points using a simpler method, saving in this way computation time.

According to this philosophy, the elected method, detailed below, is the *Uniform sampling*, which more than selecting keypoints operates a uniform down-sampling of the point cloud, reducing in this way the number of points taken into account.

Uniform sampling

The *Uniform sampling*, as already said, is a method to down-sample a point cloud. In particular it samples a point cloud in a uniform way, respecting the repeatability principle, which means that, sampling the same point cloud many times, using the same parameters, the obtained result will be the same.

The uniform sampling method used in this project is the one implemented in the *Point Cloud Library (PCL)* [37]. As explained in the PCL documentation [38] the uniform sampling method creates a *3D voxel grid*, which can be seen as a set of small 3D boxes in space, over the point cloud. Then, all the points present in each voxel grid (i.e. for each 3D box), are approximated with their centroid.

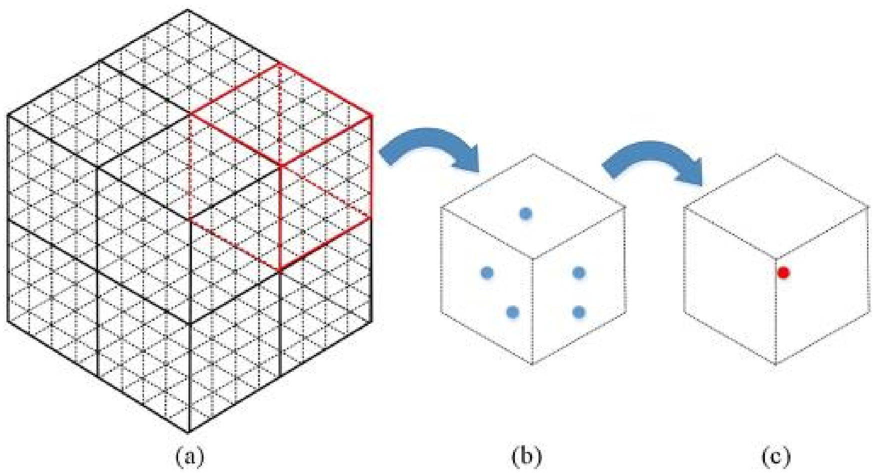


Figure 39 - Voxel grid

Notice that, by varying the length of voxel boxes' radius it's possible to choose the sampling rate. In fact, smaller is the radius and higher is the sampling rate. The picture below shows the result of a down-sampling procedure, done with uniform sampling, over a point cloud representing the ABB robot – IRB 1600.

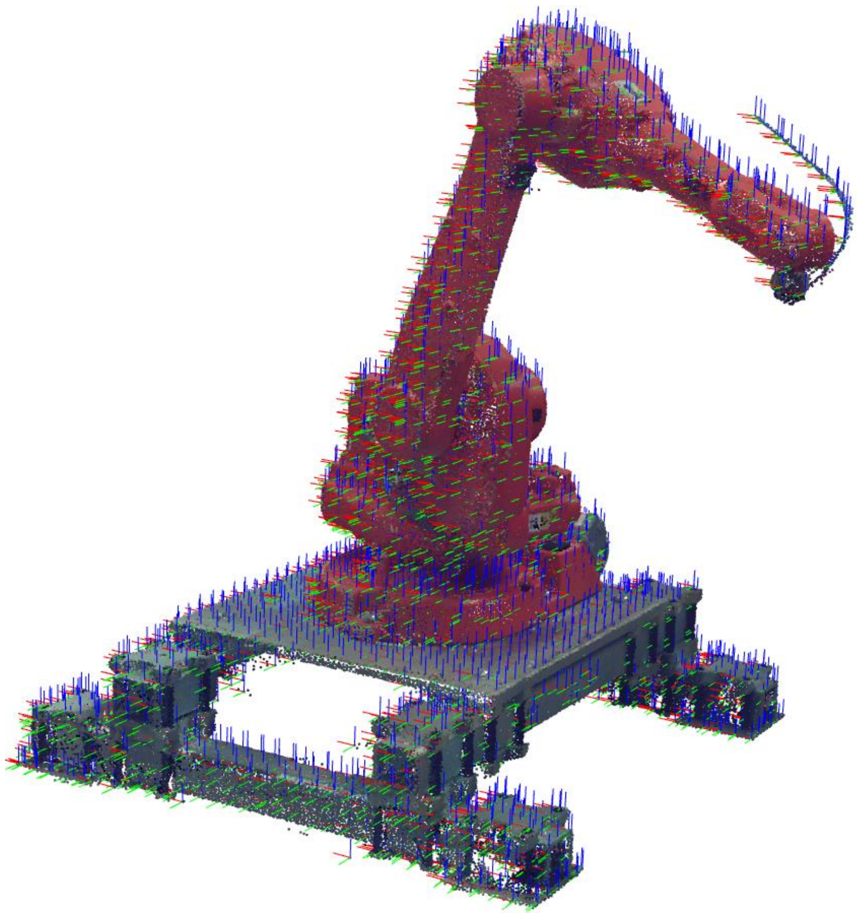


Figure 40 – Uniform sampling

3.1.2 Filtering

The application of the *Uniform sampling* on the point cloud, as shown in *Figure 44*, could led to an extremely large set of selected points, not manageable in the alignment process. Moreover, many of those points could be not significant or even counterproductive during the alignment. For example, aligning a specific link of IRB 1600 *(Figure 52)*, it's necessary to exclude from the aligning process all those points that don't belong to that link, otherwise they would have a bad influence on the process, leading to a wrong alignment. More precisely, in that case, would be essential to avoid the contribution from points belonging to other links or from outliers due to disturbance in the scanning phase. For these reasons the filtering phase plays a key role in the designed solution, indeed, it allows to reduce the set of points detected by the uniform sampling to a smaller and more coherent set, somehow similar to the inspection points set used for the collaborative robot's case.

Nevertheless, it's important to notice a big obstacle faced in this phase. In this scenario, in fact, the algorithm hasn't any hints on which section of the point cloud is representing the object to align, so, in other words, it hasn't any parameters useful to calibrate the filtering phase in the right way. To overcame this problem has been introduced an input parameter, called *delta (δ),* which is a sort of threshold measure that guides the algorithm during the filtering phase. More precisely δ represents the upper bound for the object's displacement, as in 2.3.4, except that, here, this value is entered by the user as a worst case approximation of difference between the current object's position and the correct one. In practice, if the user inserts $\delta = \tau$, it means that the object is approximatively τ millimeters away from the correct position.

The filters designed and implemented for this are two, the *Cut-off filter* and the *Measure filter*, which are both detailed below. The cut-off filter operates a coarse selection, taking decisions based just on the position of points, while the measure filter refines the point set in a more precise way, taking into account also the distance measure. Notice that both filters receive as input parameter the same δ value.

Cut-off filter

The *Cut-off filter,* as said above, operates a first, coarse, filter on points coming from the uniform sampling. It gets the *bounds* of the virtual object in the virtual environment, it extends those bounds by δ, the parameter introduced by the user, and then remove from the keypoint set all those points that are outside these extended bounds. This filter allows to detected the area of interest inside a point cloud and it can be particularly useful when it comes to align an object having as input data a very large point cloud (e.g. a point cloud representing a whole production cell).

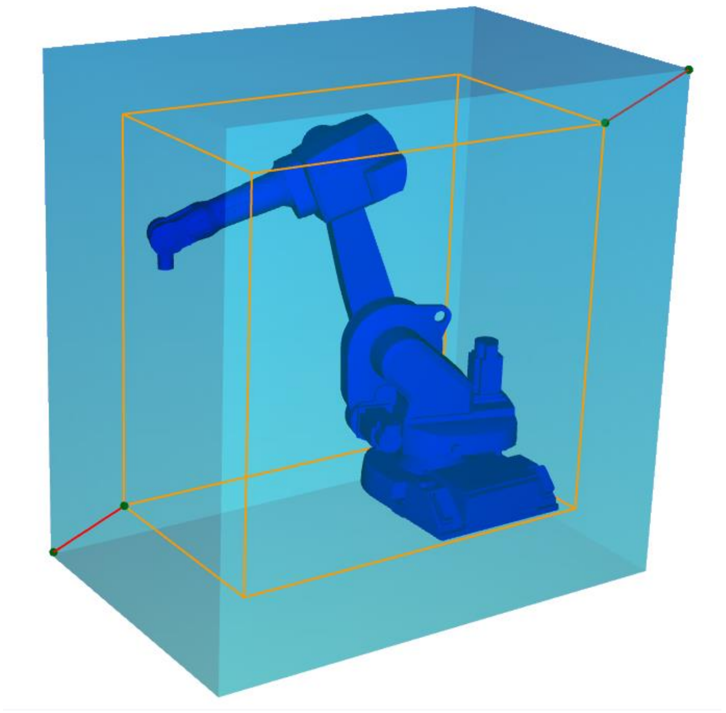


Figure 41 – Cut-off filter

Measure filter

Since the δ introduced by the user can be big and, applying the cut-off filter, the bounds are expanded in all directions by δ, the set of keypoint selected can be very large and it may include non-significant points. Therefore, in order to refine this set of keypoint it's necessary to apply a second, more precise, filter, called *Measure filter.*

As its name suggests, the Measure filter is applied on the distance measure between the keypoint and the virtual object. More precisely, for each point coming from the cut-off filter it's computed the distance from the virtual object, then, if this distance is greater than δ the point is removed from the selection, otherwise it's kept in the keypoint set.

Noticed that, theoretically, it would be possible to apply directly this filter on the keypoints coming from the *Uniform sampling*, skipping the cut-off filter, and the result would be the same. However, in practice, this cannot be done because, being the number of points resulting from the down-sample very large, it would be computationally too heavy to calculate the distance for each of them. That's why it's necessary to apply, in first instance, the cut-off filter.

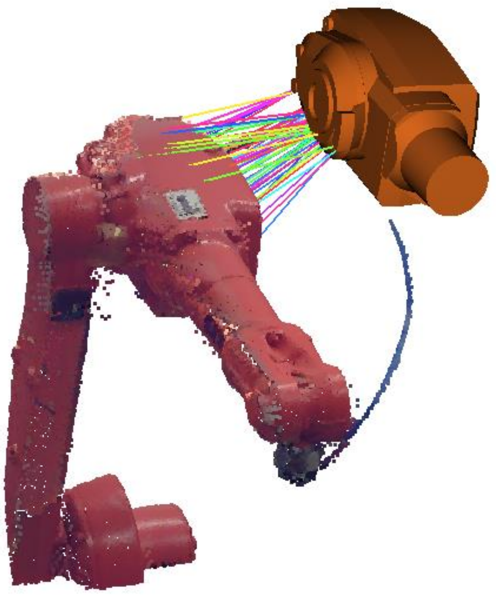


Figure 42 – Measure filter

3.1.3 Point Cloud Alignment algorithm

This section presents the algorithm proposed in this project to perform the alignment using point clouds. As already said, this *Point Cloud Alignment* algorithm can be considered as a readjustment of the *Point Set Alignment* algorithm (see 2.3.1).

Having as input the point cloud representing the physical object $\mathcal{P}$, the virtual object to align $\mathcal{S}$ and the upper bound of the initial displacement δ, the basic idea is to perform, in first instance, the down-sampling of the point cloud, with the method explained in 3.1.1. Then, for each iteration of the algorithm, the filters described in the previous section are applied, using δ as threshold measure. At the end of this point cloud's refinement the remaining points are used for the alignment, in the same way as the inspection points coming from the cobot were used in the first solution.

The actual alignment phase, core of the algorithm, is similar to the *Point Set Alignment* algorithm. In fact, also for the point cloud case it's used an adapted version of the ICP algorithm. However, differently from method described in section 2.3.1, here is used the weighted version of the SVD algorithm, applying a *weighting function* to the correspondences.

Finally, at the end of each iteration, is performed the reduction of the parameter δ, which corresponds to a contraction of the algorithm's search space. For the purpose of applying this iterative reduction has been designed a specific function, called *reduction function*. Below are shown and explained in detail the weighting function and the reduction function used in the *Point Cloud Alignment* algorithm.

Weighting function

In the context of the alignment with point cloud the weighting of correspondences is useful to avoid the contribution of outliers to the alignment process. This procedure consists in assigning a certain weight to each pair of closest inspection-object points and then apply the weighted SVD method (see *2.1.2*). Weights are assigned to the correspondences based on the distance (d) between the two points.
The function designed to weight the correspondences is the following:

$$w(d) = \cos\left(\frac{d}{\delta} \cdot \frac{\pi}{2}\right) \qquad (32)$$

Notice that, when the two points coincide $d = 0$ and therefore $w = 1$. Instead, when $d = \delta$ the cosine's argument becomes equal to $\pi/_2$ and so $w = 0$. Basically, this weighting function, starting from 1, decreases smoothly while the distance increases and reaches the 0 in δ.
It's also important to observe that the function has meaning only when the distance is lower or equal to δ, indeed distances greater than δ are excluded by the measure filter.
In *Figure 47* is shown in blue the weighting function, while in red is drawn the evolution of the weight with respect to the distance when is used a non-weighted version of the algorithm or, in other words, when a uniform, equal to 1 weighting function is applied. As already said, the purpose of this weighting function is to avoid that outliers affect the alignment. Supposing, for example, that the object is moved away from its correct position by a minimal distance ε, some of the outliers previously excluded by the measure filter because δ + ε away from the object could come into play. In that case the described function would, in any case, attribute a very low weight to the outlier, preventing it from influencing the algorithm and ensuring in this way the robustness of the provided solution. At the opposite, applying the uniform weighting function those outliers would enter in the process with an important weight (equal to 1), causing instability in the algorithm.

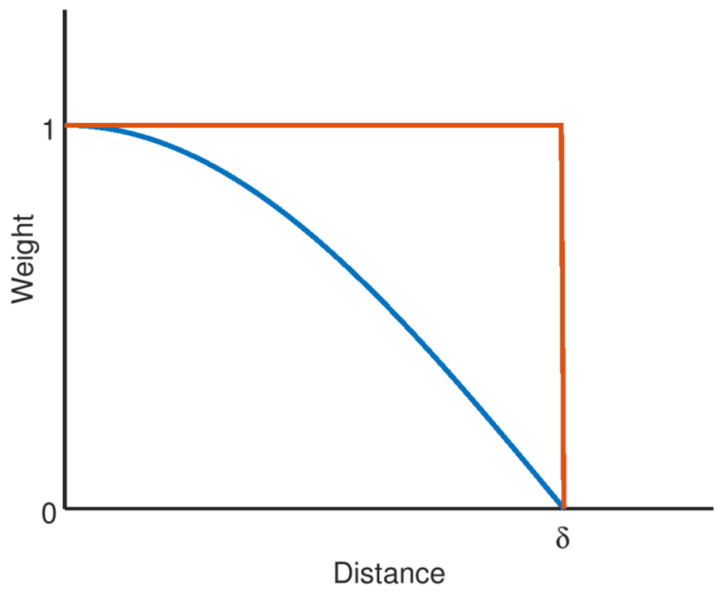


Figure 43 – Weighting function

Reduction function

The reason for reducing the δ, as stated before, is to reduce the search space of the algorithm. Indeed, consider for example the case in which the object is moved away from its correct position by a significant distance and, therefore, the user enters a quite big initial upper bound δ_0 for the displacement. In that case, in first instance, the filters applied on the point cloud would be coarse-grained, since it must be recalled that they're based on the δ value. Therefore, the set of points resulting from the filtering phase would be quite large and, of course, it would contain lots of outliers. In that scenario, without a reduction of the parameter δ, the algorithm would never converge to the solution. Instead, applying the reduction function, with continued iterations the object starts to align and meanwhile the δ reduces, causing the filters to become more fine-grained and to produce more precise sets of points. As result of this progressive alignment and reduction of the search space the algorithm converges to the solution.
The reduction function designed for this purpose is:

$$\delta_i = (\delta_0 - \delta_{min})\, cos\left(\frac{i}{100} \cdot \frac{\pi}{2}\right) + \delta_{min} \qquad (33)$$

Starting from δ_0 the δ value decreases smoothly at each iteration, till it becomes equal to δ_{min}. Observe that this reduction takes place in 100 iterations to allow the alignment of the object and prevent the loss of significative inspection points that may be thrown away by the filters if the δ is reduced too quickly.

The reason why the reduction stops when δ reaches δ_{min} is that, due to inaccuracies of various kinds, the virtual model can't exactly coincide with the reality and therefore, the reduction of the δ below a certain threshold could lead to unexpected misbehaviours in the alignment. Instead, maintaining this thin layer of uncertainty the algorithm converges more regularly. However, it doesn't mean that the algorithm can't align objects with a precision greater than δ_{min}, indeed, even if this uncertainty layer causes the contribution of outliers, being it very thin the influence of coherent points will always prevail.

Notice that in this work has been used a value for δ_{min} equal to 10mm, which demonstrates to be a good choice.

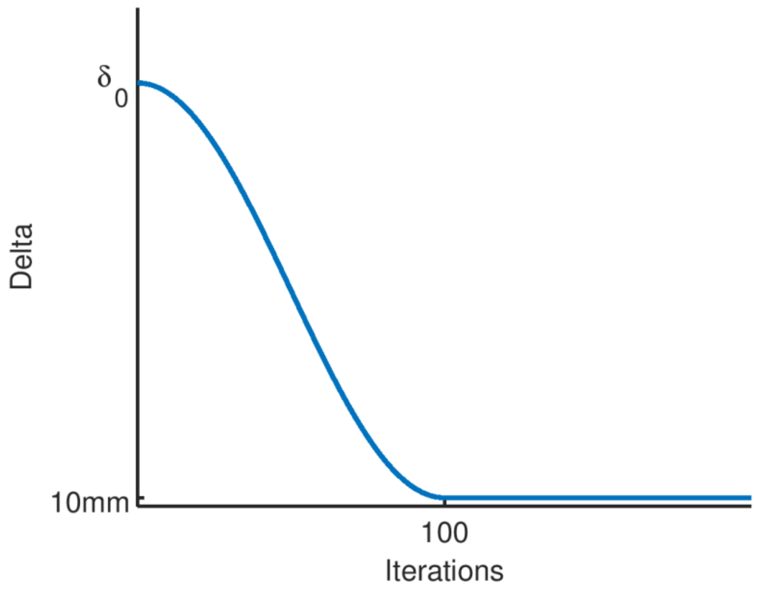


Figure 44 – Reduction function

Algorithm

At this point all the elements and aspects characterizing the algorithm have been discussed and, therefore, can be presented here a detailed description of the stages composing the method.

The *Point cloud alignment* algorithm can be summarized in the following steps:

1. Down-sample, through the *Uniform sampling,* the point cloud $\mathcal{P}$, having n_p points, obtaining the set of keypoint K_1, having n_k points, where $n_k < n_p$.

$$K_1 = UniformSampling(\mathcal{P},\ radius) \qquad (34)$$

2. Apply the cut-off filter on K_1, using δ and getting as result the keypoints set K_2

$$K_2 = CutOffFilter(K_1, \delta_i), \qquad (35)$$

where δ_i is the value of the parameter δ at i-th iteration.

3. Apply the measure filter on K_2, using δ and obtaining in this way $I = \{i_k\}$, the set of *N* inspection points involved in the alignment.

$$I = MeasureFilter(K_2, \delta_i), \qquad (36)$$

where δ_i is again the value of the parameter δ at i-th iteration.

4. Find, for each point in *I*, the corresponding closest point on the object's surface *S*, getting as result $P = \{p_k\}$.

$$p_k = C(i_k, S_i), \quad k = 1, \dots, N \qquad (37)$$

where S_i is the object's surface at the i-th iteration. Obviously, the surface's shape doesn't change during the alignment, but its position can be different at each iteration.

5. Weight the correspondences found at step 4 based on the weighting function presented above, obtaining the set of weights $W = \{w_k\}$.

$$w_k = cos\left(\frac{d_k}{\delta_i} \cdot \frac{\pi}{2}\right), \qquad (38)$$

where $d_k = \|i_k - p_k\|$ is the distance between the two points and δ_i, as usual, is the value of the parameter δ at i-th iteration.

6. Compute the registration using the SVD method described in section 2.1.1, finding the optimal transformation.

$$T_i = SVD(I, P, W) \tag{39}$$

7. Apply the transformation to the object, obtaining a new surface S_{i+1}, with the same shape but different position.

$$S_{i+1} = T_i(S_i) \tag{40}$$

8. Check if the stopping criterion (see *2.1.4*) is triggered or if the maximum number of iterations has been reached. If one of the two statements it's true then terminate the algorithm, otherwise compute the δ reduction, based on the reduction function explained above and then loop to 2.

3.2 Technologies

Besides all the software technologies adopted in the implementation of the first solution (see *2.2.1*), here have been used also the PCL software library and the GNU Octave software, which are presented in the following sections.

3.2.1 Point Cloud Library – PCL

The *Point Cloud Library (PCL)* [38] is a standalone, large scale, open project for 2D/3D image and point cloud processing. In particular, the PCL framework provides numerous algorithms operating on point clouds, which allows to perform filtering, keypoint detection, feature estimation, surface reconstruction, registration, model fitting and segmentation on a point cloud.

Figure 45 – PCL logo

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

PCL library is open source software, released under the terms of the 3-clause BSD license and is open source software, so it's completely free both for commercial and research use.

3.2.2 GNU Octave

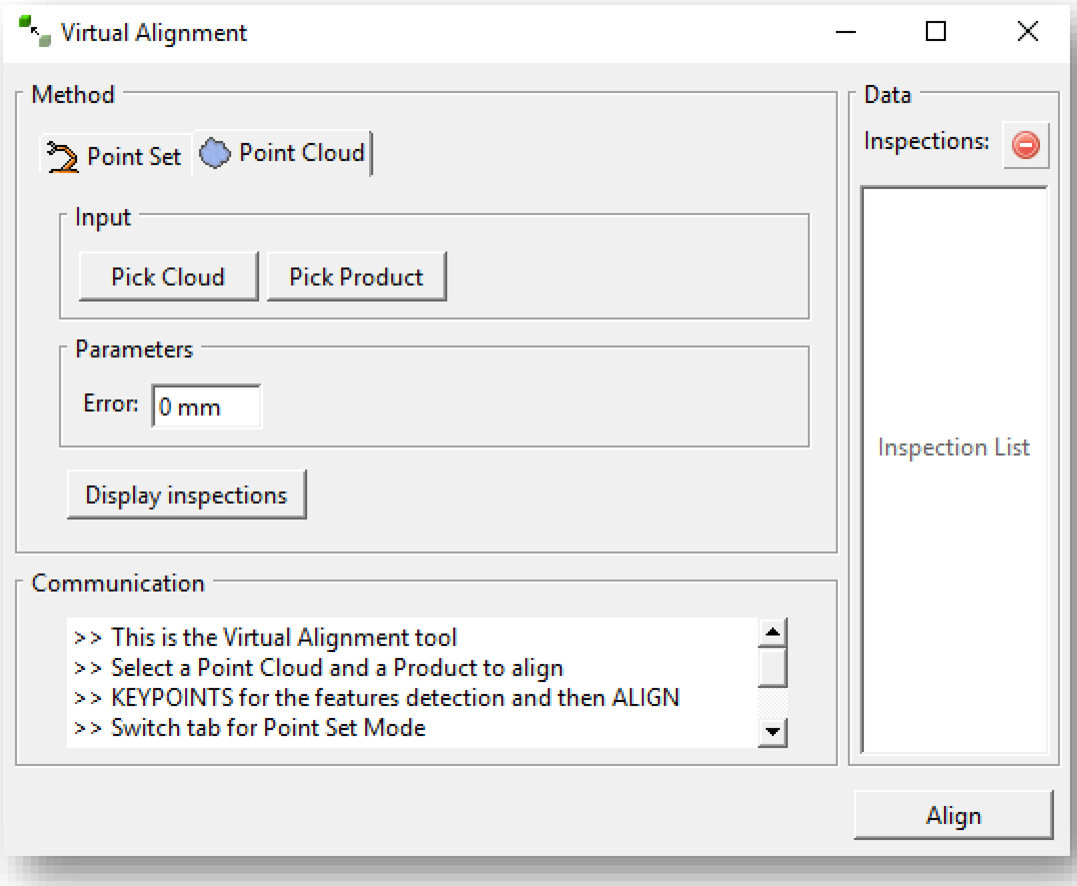
GNU Octave [39] is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments.
This software offers functionalities very similar to the ones provided by Matlab [40], the well-known proprietary software, and it uses a language that is mostly compatible with Matlab's one.
GNU Octave is freely redistributable software, under the terms of the GNU General Public License (GPL).

3.3 Implementation

In this section are detailed all the implementation aspects of the solution involving point clouds. In particular, are first described the modifications made to the user dialog in order to handle this new alignment method, then the implementation details of the PCA algorithm are exposed and commented.

3.3.1 Geometry Alignment Dialog

In order to handle the alignment through point cloud and allow the user to manage this new alignment process, the dialog developed in the first part (see 3.1) has been modified. Same as for the previous dialog, also in this case all the modifications have been made using the FOX Toolkit [28].
The main change done is the division of the dialog in two panels, the first one called "Point Set" and the other one "Point Cloud". The point set panel allows the user to manage the alignment process done through collaborative robot and presents the same functionalities implemented in the previous dialog. The point cloud panel instead, is the one introduced to handle the new alignment process.
In *Figure 50* is shown the new dialog and, in particular, it shows the point cloud panel.



```
>> This is the Virtual Alignment tool
>> Select a Point Cloud and a Product to align
>> KEYPOINTS for the features detection and then ALIGN
>> Switch tab for Point Set Mode
```

Figure 46 – Point cloud panel

The user, depending on how he wants to align the object, has to select the proper panel and then follow the alignment procedure.

The point cloud panel has two main sections: *Input* and *Parameters*.

The *Input* section allows the user to choose on IPS the product he wants to align and the point cloud he intends to use for the alignment.

In the *Parameters* frame the user has to insert an estimation, as accurate as possible, of the object's initial displacement, which represent the δ value. To be exact, this parameter represents an upper bound for the initial displacement and this means that the initial distance between the real position and the virtual one can't be greater the *Error*.

The *Communication* and the *Data* frames fulfil the same role they had in the previous dialog and the same is true also for the *Align* button, that here again is devoted to the alignment process´s start.

The button *Display Inspections* instead, has been introduced in this new point cloud panel to give the possibility to the user of displaying the inspection points used for the alignment. In other words, clicking this button the user can see those points, among all the point cloud's points, that the algorithm selected for the alignment through keypoint detection and filtering phases.

3.3.2 Point Cloud Alignment

The *PointCloudAlignment* class implements the algorithm explained in section *3.1.3*. In particular, the constructor of this class takes as input and stores as data members the point cloud and the object to align, selected through the dialog, and the value δ entered by the user, representing the upper bounds of the initial displacement. Then it performs the down-sampling of the cloud, in accordance with the method explained in *3.1.1* and saves also the resulting set of points in a member variable. The following lines are pseudo-code of this constructor.

```
PointCloudAlignment(iCloud, iObject, iDelta)
: object(iObject), cloud(iCloud), delta(iDelta) {
     // down-sample the cloud
     keypoints = pcl::UniformSampling(cloud, voxel_r);
}
```

Notice that the down-sampling operation is done exploiting the point cloud library's uniform sampling functionality (`pcl::`), passing as parameters the point cloud and the voxel grid's radius.

The most important member function of the *PointCloudAlignment* class is the *align* method. In fact, using the data members saved by the constructor, it implements the alignment´s core. The following pseudo-code define the *align* method's skeleton:

```
while (!termination(newPos)) {
   // cut off
   inspectionPoints = cutoffFilter(keypoints, object, delta);
   // find closest points on product, apply measure filter and
   // weight the correspondences
   findWeightPoints(inspectionPoints, objPoints, weights, delta);
   // SVD registration
   newPos = SVDRegistration(inspectionPoints, objPoints, weights);
   // loop
   update();
}
```

The iterative nature of the algorithm is implemented through a *while* loop, which iterates until the stopping criterion is triggered. In each loop's cycle, first the cut-off filter is applied on the down-sampled cloud *keypoints*, getting as result the set of points useful for the alignment of the object (*inspectionPoints)*. Then, the function *findWeightPoints* fulfils a triple function. It computes the distance between each element of *inspectionPoints* and the object, finding in this way the set of closest points. Moreover, the function exploits the computed distance to apply the measure filter and, finally, it weights the resulting correspondences. The pseudo-code below gives a more detailed representation on the *findWeightPoints* function.

```
for each inspection in inspectionPoints {
   // compute distance
   measure = createMeasure(inspection, object);
   measure->getDistance(distance, from, to);
   // measure filter
   if (distance <= delta) {
      // valid inspection point --> save inspection and object point
      newInspectionPoints.PushBack(inspection);
      objectPoints.push_back(to);
      // weight correspondence --> w = cos^2(dist/delta*pi/2))
      weight = 2*pow(std::cos(dist/(delta*2)*PI/2),2) - 1;
      weights.PushBack(weight);
   }
}
inspectionPoints = newInspectionPoints;
```

At this point the algorithm proceed with the registration phase, implemented by the function *SVDRegistration*, which calculates the optimal transformation and applies it to the object. Differently from the implementation explained in *2.3.2*, here is used the weighted version of the SVD method (see *2.1.2*), which is described by the following section of pseudo-code.

```
SVDRegistration(inspectionPoints, objPoints, weights) {
   // compute centers of mass
   for (int i = 0; i < nPoints; i++) {
      ui += inspectionPoints[i]*weights[i];
      up += objPoints[i]*weights[i]; }
   ui /= weightsSum;
   up /= weightsSum;
   // compute H matrix
   for (int i = 0; i < nPoints; i++) {
      I(i) = inspectionPoints[i] - ui;
      P(i) = productPoints[i] - up; }
   H = P*I.transpose();
   // SVD
   Eigen::SVD(H, &U, &V);
   // compute optimal rotation and translation
   R = V*U.transpose();
   t = ui - upW.t;
   // apply complete transformation
   newPos = object->applyTransformation(R, t);
   return newPos; }
```

Notice that in the computation of the centres of mass is taken into account the weight value of each correspondence.

Finally, at the end of each iteration, the *update* function prepares the algorithm to the next cycle. In particular this method increments the iterations number and calculates the new value for the δ, applying the reduction function defined at *3.1.3*. The pseudo-code of this function is the following, where *its* stands for iterations and *d0* is the initial value of the δ (the one entered by the user).

```
update() {
   ...
   // iterations increment
   its++;
   // delta reduction
   if (delta > threshold) {
      // reduction function
      delta = (d0-0.01)*pow(std::cos(its/(double)100*PI/2),2)+0.01;
   }
   return;
}
```

Notice that the *termination* method used to trigger the exit from the loop is the same already used and explained in *2.3.2*.

3.3.1 Tests

The testing method used to check the PCA algorithm's performances is the same already described in *2.3.4*. Therefore, also in this case, starting from the ideal situation where the object is perfectly aligned with point cloud, a random transformation is applied to the object, misaligning the geometry, and then the algorithm is run, checking if it can realign the scenario.

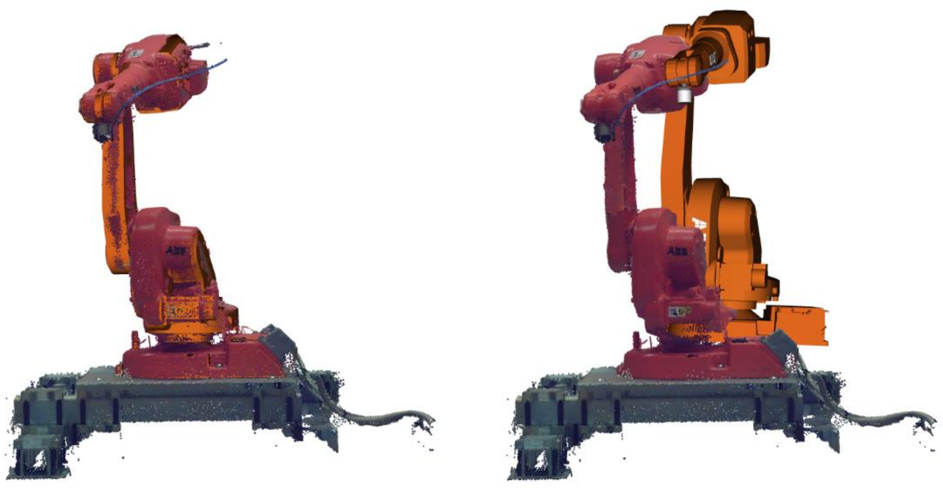


Figure 47 – PCA Tests

3.4 Results

The *Point Cloud Alignment* algorithm has been tested on a model representing the robot IRB-1600 produced by ABB *(Figure 52*), using as input data the point cloud shown in *Figure 42*.

The testing method adopted is the same used for the *Point Set Algorithm*, which is described in *2.3.4* and *3.3.1*. In particular, trials have been performed, first on the whole model and then, separately on each of the links belonging to the robot, evaluating the algorithm's performances in face of displacements of different magnitude.

Figure 53 instead, depicts the IRB's links involved in the test, which are: Link0 (yellow), Link1 (green), Link2 (red), Link3 (blue), Link4 (orange), Link5 (violet) and Link6 (grey).

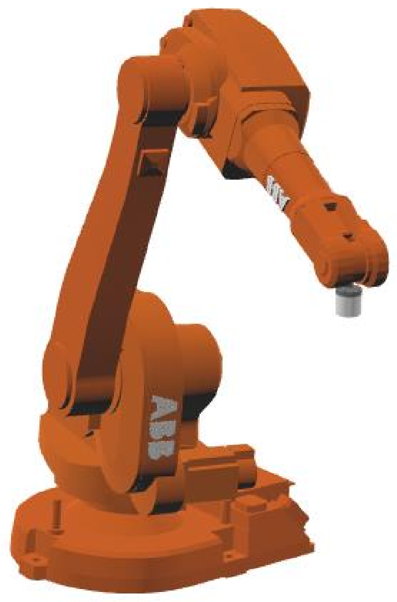


Figure 48 – ABB IRB-1600 Model

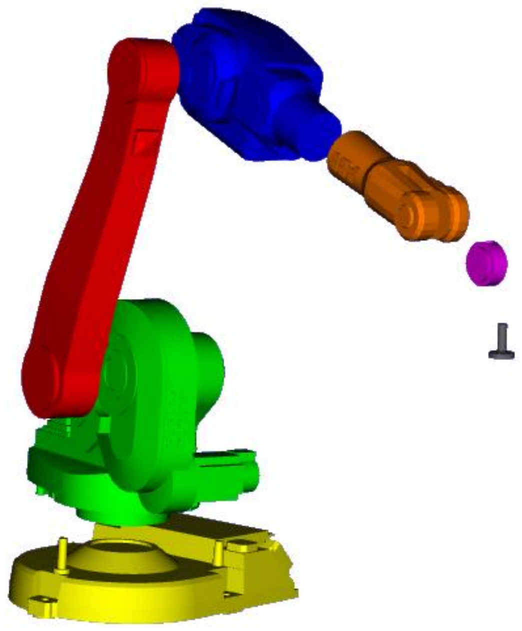


Figure 49 – ABB IRB-1600 links

Alignment performance analysis

The following graph shows the evolution of the displacement error on 50 different tests ran on the whole IRB model, setting a maximum initial displacement δ = 200mm.

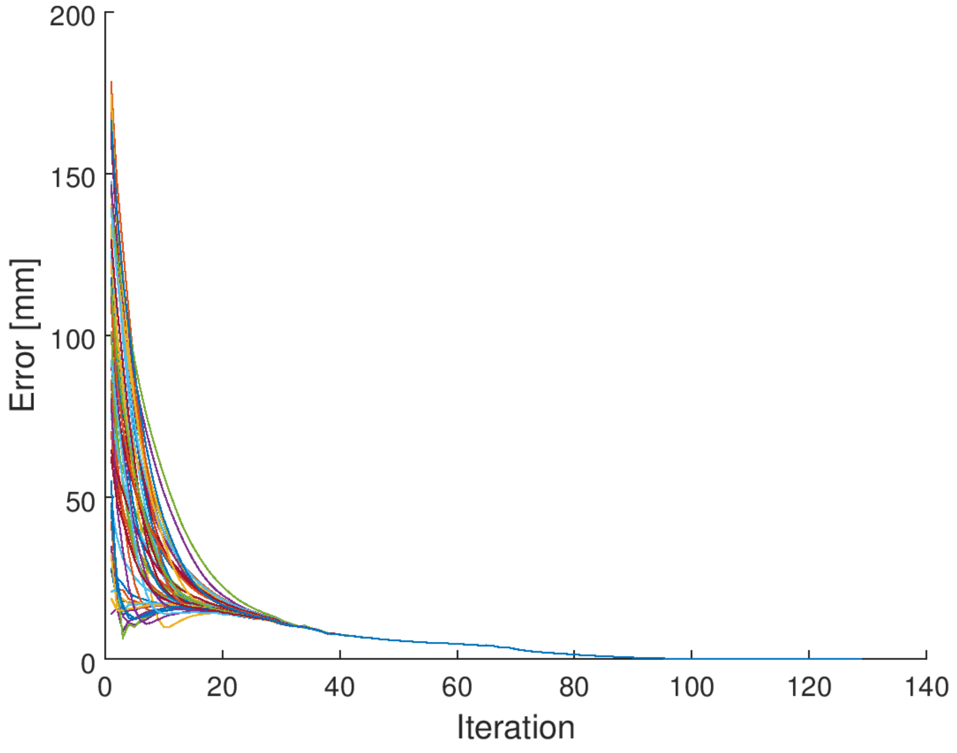


Figure 50 - IRB test results with δ = 200mm

Observe that, all the tests, even if they start from different, wrong, positions, tend to converge to same optimal solution. This behaviour proves the stability of the algorithm and its efficiency in finding the correct object's position. Indeed, for the specific case depicted in the graph the average final error is $1{,}4 \cdot 10^{-4}$ mm. This ideal behaviour is valid also for other δ values and even for big displacements such as 1000 mm.

It's important to notice that the alignment of the whole model is easier to solve by the algorithm, compared to the alignment of a single, specific link. In fact, the alignment of a link requires to precisely distinct which points belong to that link and which not, while this problem is not present in the model case, where basically all the points belong to the IRB's geometry.

For this reason, the tests ran singularly on each of the IRB's link gave particularly contrasting results. In particular, the algorithm's behaviour for Link0, Link1, Link2, Link3 and Link4 is very similar to the one observed with the whole model, with just some minor exceptions, while for Link5 and Link6 the depicted scenario is completely different, due to some critical issues related with the nature of the geometry and the point cloud's resolution. The following analysis refers just to the first five links, while the issues related to other links will be discussed afterword.

In *Figure 55* are shown the results obtained in Link2's alignment with a maximum initial displacement δ = 50mm.

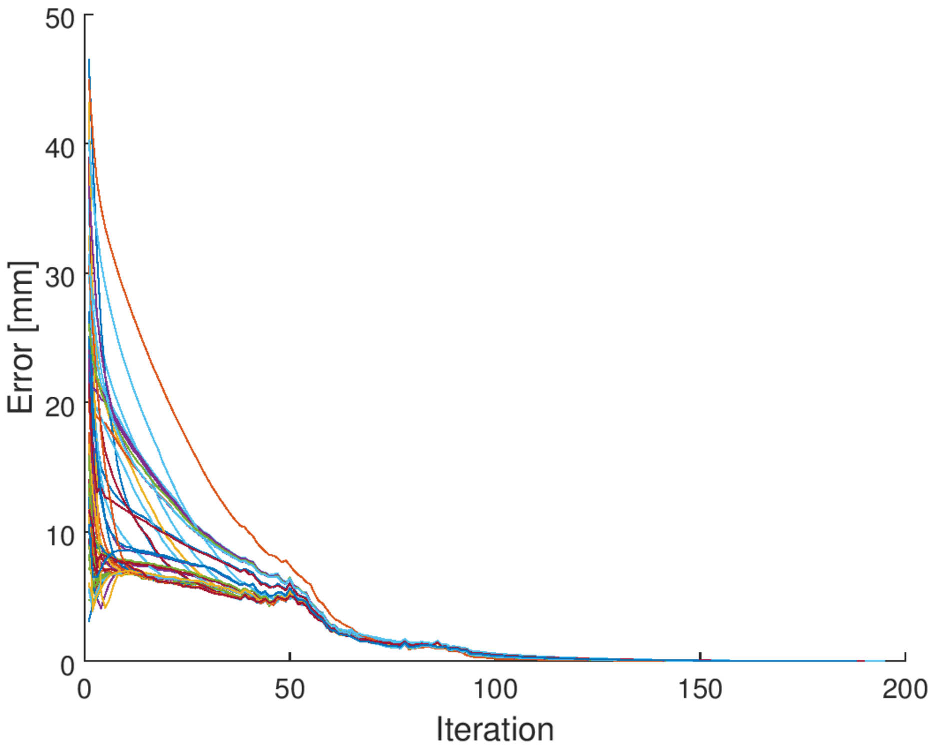


Figure 51 – Link2 test results with δ = 50mm

Observe that, those results reflect the behaviour already described above and it has been encountered also for the other four links, even with greater δ, such as 100mm and 200mm.

In particular, for Link0 and Link3, the algorithm appears to be stable and effective also when the initial displacement is very big as shown in *Figure 56* and *Figure 57*, which display, respectively, the results for Link3 with δ = 500mm and for Link1 with 1000mm of maximum displacement.

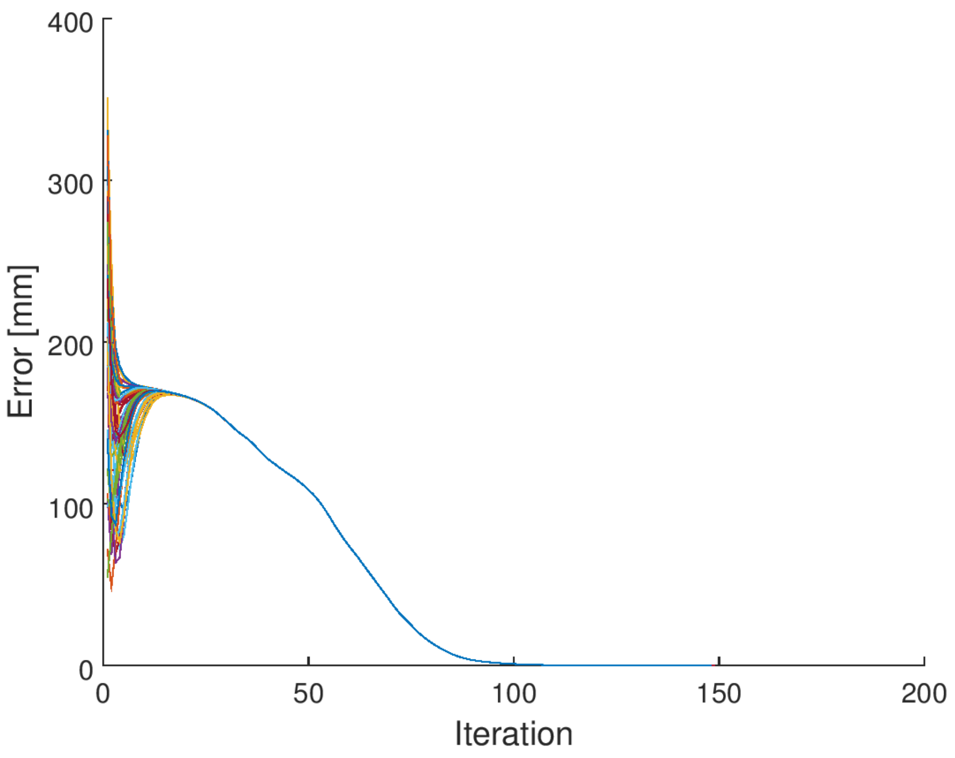


Figure 52 – Link3 test results with δ = 500mm

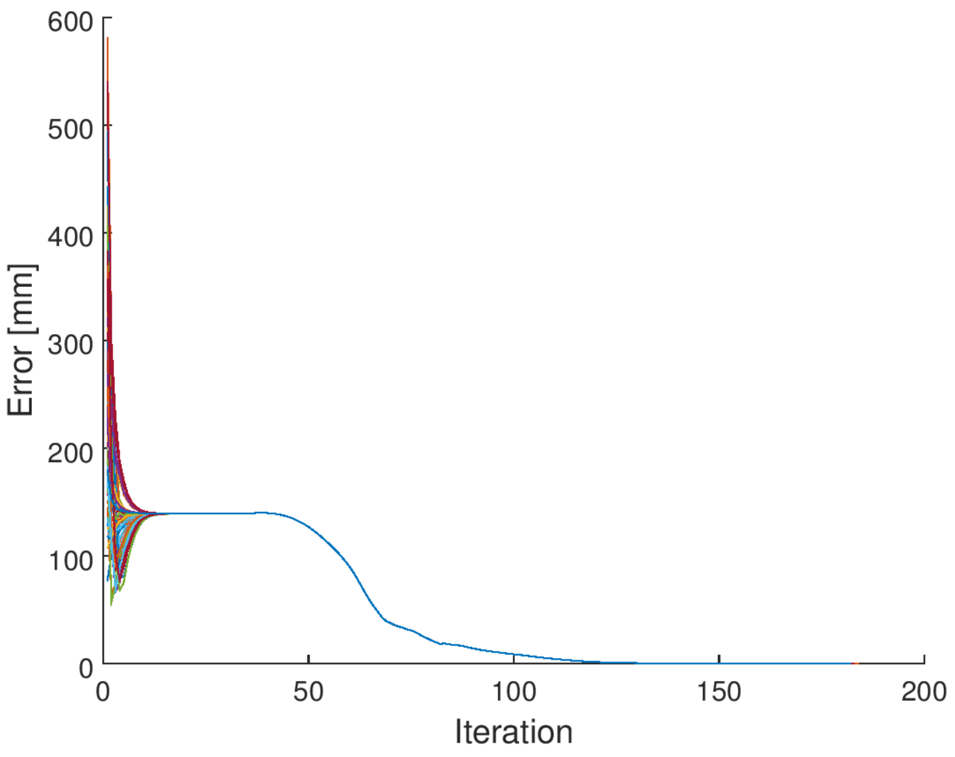


Figure 53 – Link0 test results with δ = 1000mm

With Link1, Link2 and Link4, instead, the algorithm's performances tend to degenerate when the δ increases. More precisely, Link1's case has still very positive results, indeed just with δ = 1000 mm, which can cause quite big displacements, the algorithm doesn't converge to the global minimum, as shown in *Figure 58*. Instead, the solutions found on Link 2 don't correspond to the best one already with δ = 500mm (*Figure 59*). However, it's important to notice that in those critical situations, even if the algorithm doesn't converge to the global optimal solution, it still maintains a quite stable behaviour, indeed all the tests continue to converge to a single solution.

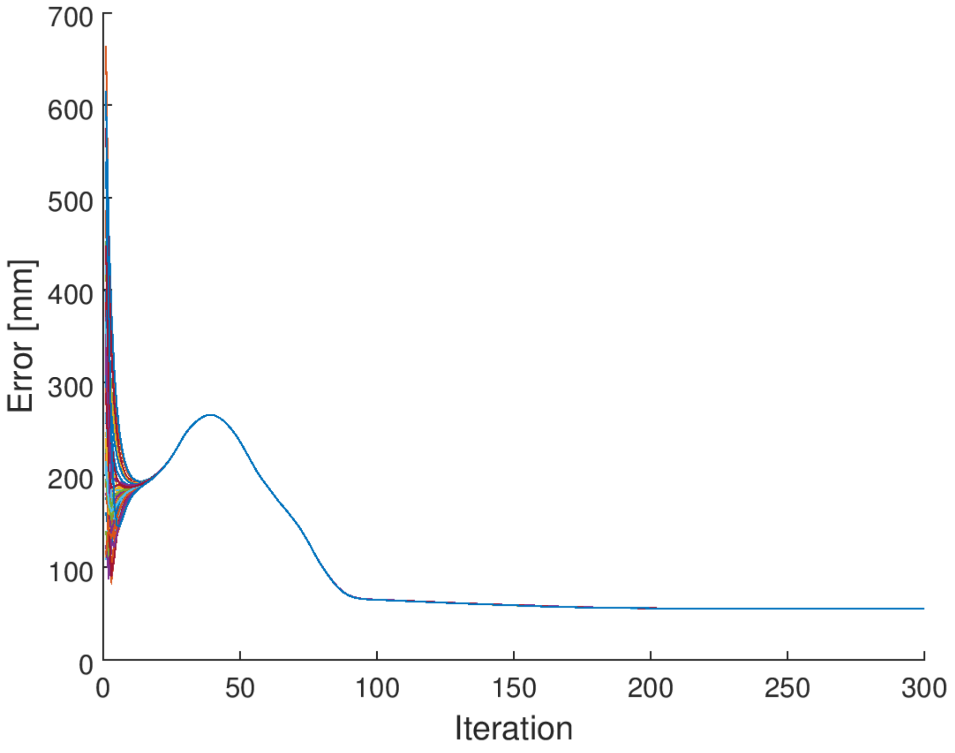


Figure 54 – Link1 test results with δ = 1000mm

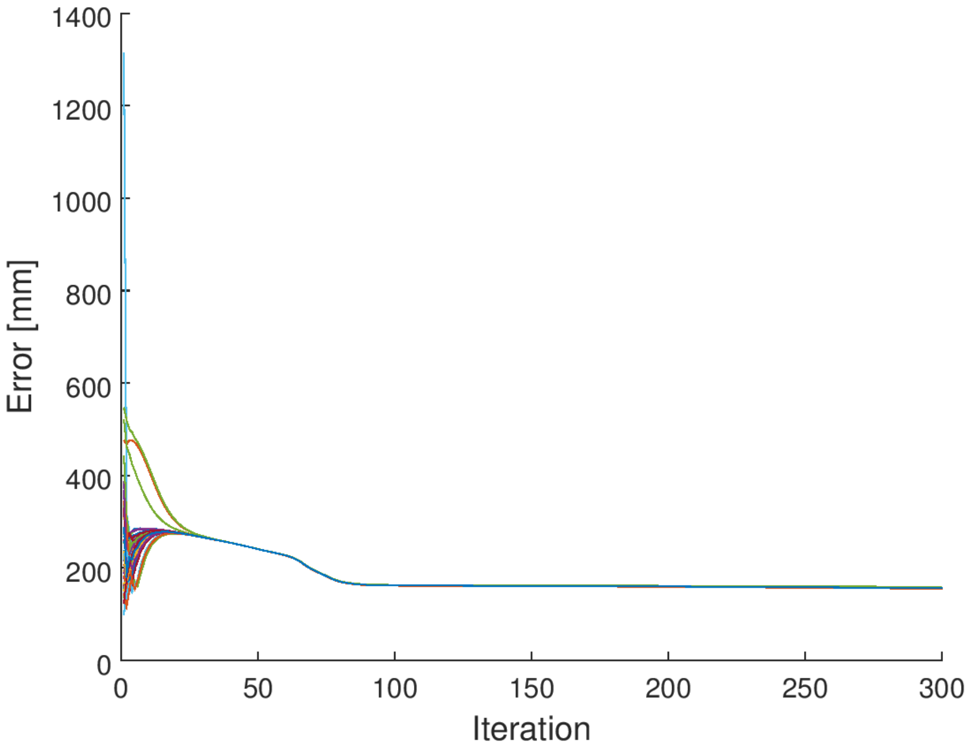


Figure 55 – Link2 test results with δ = 500mm

Notice that in the last two graph the number of iterations computed by the algorithm is equal to 300. This value is not determined by the termination criterion but, is the maximum number of iterations set on the tests to avoid a never-ending loop and, therefore, it can be inferred that in those cases the method can't find a definitive and stable solution, but it's still improving its position.

In the alignment of Link4 instead, problems come already for δ = 200mm, as shown in *Figure 60*, where the found solutions are stable since the number of iterations is lower than 300, but they're not the optimal one: Moreover, the algorithm appears to be not stable, indeed, starting from different initial positions, it converges to different solutions. With δ = 500mm (*Figure 61*) instead, the algorithm has as stable behaviour but it can't find the optimal solution.

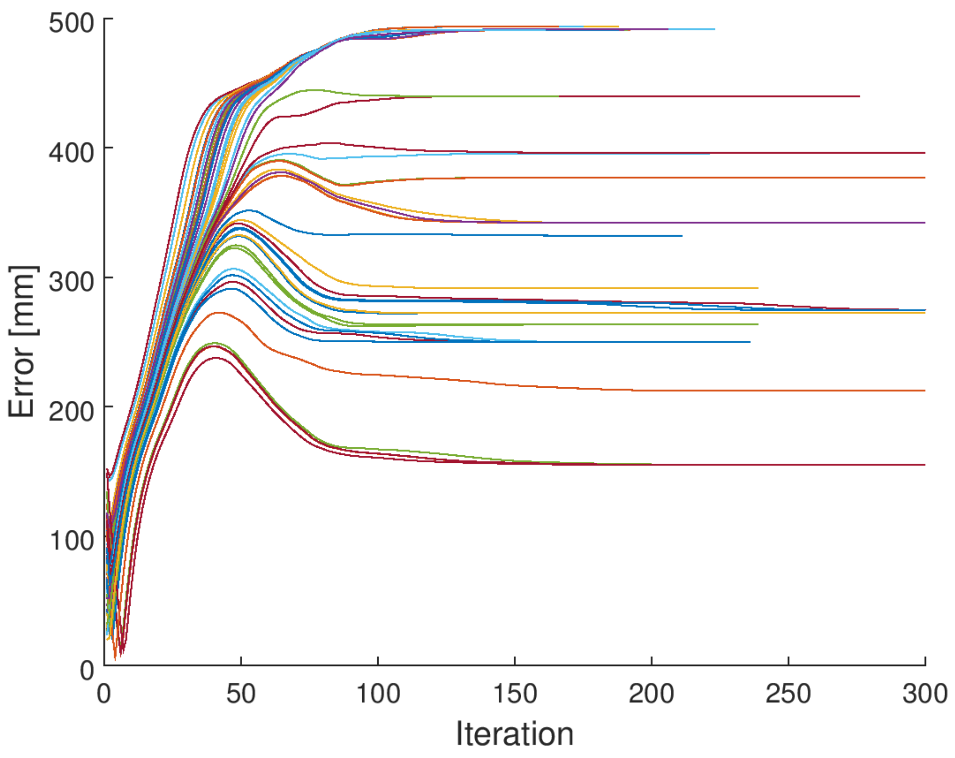


Figure 56 – Link4 test results with δ = 200mm

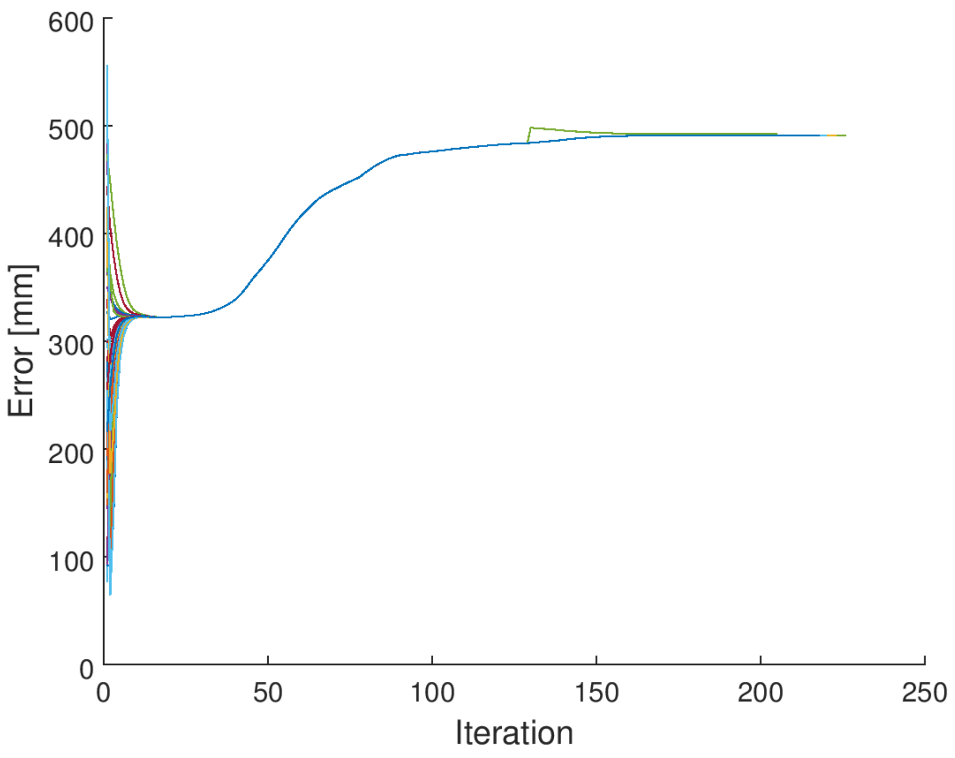


Figure 57 – Link4 test results with δ = 500mm

For what concerns Link5 and Link6, as already mentioned, the results obtained are not coherent with the ones described above and they underlie the difficulties encountered by the algorithm in aligning those links. In those cases, the nature of the problem has to be found in the resolution of the adopted point cloud. Indeed, as shown in the figure below, Link5 and Link6 are little compared to dimension of the whole model and, for this reason, the point cloud's resolution is not sufficient to accurately derive the position of those links.

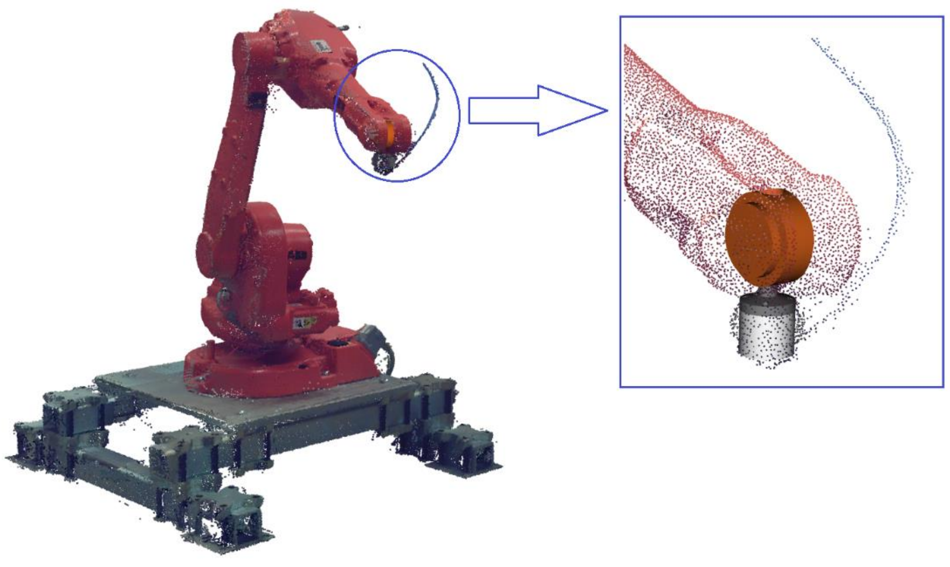


Figure 58 – Link5 and Link6 detail

The table below summarizes the obtained results from the point of view of the alignment performances.

Object	Delta (mm)	Succeed	Fail	Avg. Error (mm)
IRB	50	50	0	$\mathbf{1,29 \cdot 10^{-4}}$
	100	50	0	$1{,}11 \cdot 10^{-4}$
	200	50	0	$1{,}42 \cdot 10^{-4}$
	500	50	0	$1{,}45 \cdot 10^{-4}$
	1000	50	0	$1{,}4 \cdot 10^{-4}$
LINK 0	50	50	0	$2{,}86 \cdot 10^{-4}$
	100	48	2	3,75
	200	50	0	$2{,}75 \cdot 10^{-4}$
	500	50	0	$2{,}98 \cdot 10^{-4}$
	1000	50	0	$2{,}26 \cdot 10^{-4}$

LINK 1	50	50	0	$6{,}35 \cdot 10^{-4}$
	100	50	0	$7{,}1 \cdot 10^{-4}$
	200	50	0	$5{,}92 \cdot 10^{-4}$
	500	50	0	$6{,}13 \cdot 10^{-4}$
	1000	0	50	55,1
LINK 2	50	50	50	$3{,}55 \cdot 10^{-3}$
	100	50	50	$3{,}74 \cdot 10^{-2}$
	200	50	50	$7{,}05 \cdot 10^{-2}$
	500	0	50	156,7
	1000	0	50	212,4
LINK 3	50	50	0	$3{,}46 \cdot 10^{-2}$
	100	50	0	$3{,}44 \cdot 10^{-4}$
	200	50	0	$3{,}49 \cdot 10^{-4}$
	500	50	0	$3{,}37 \cdot 10^{-4}$
	1000	50	0	$3{,}27 \cdot 10^{-4}$
LINK 4	50	50	0	$3{,}92 \cdot 10^{-4}$
	100	50	0	$3{,}77 \cdot 10^{-4}$
	200	0	50	379,2
	500	0	50	457,9
	1000	0	50	490,8

Computational Time Analysis

Besides the analysis on the algorithm's alignment performance, it's important to evaluate also the computational time required to find a solution. In general, the time necessary to align the object is directly proportional to the number of points the algorithm has to manage and to the initial displacement error.

The graph in *Figure 63* shows the relationship between alignment time and initial displacement δ. The red markers in the graphs indicate the failed tests, for which 300 iterations have been performed, therefore they are not significative in this context. Notice that, the time consumed by the algorithm grows along with δ but, in some cases, this increment is more pronounced than in others.

The reason of that is the different number of points managed by the algorithm. Indeed, in the IRB and Link1 cases, due to the object's dimensions, the points selected and used to compute the solution are many more than in the Link3 and Link4 cases. The graph in *Figure 64* underlies this difference, showing for each case the average number of points considered, for different δ values.

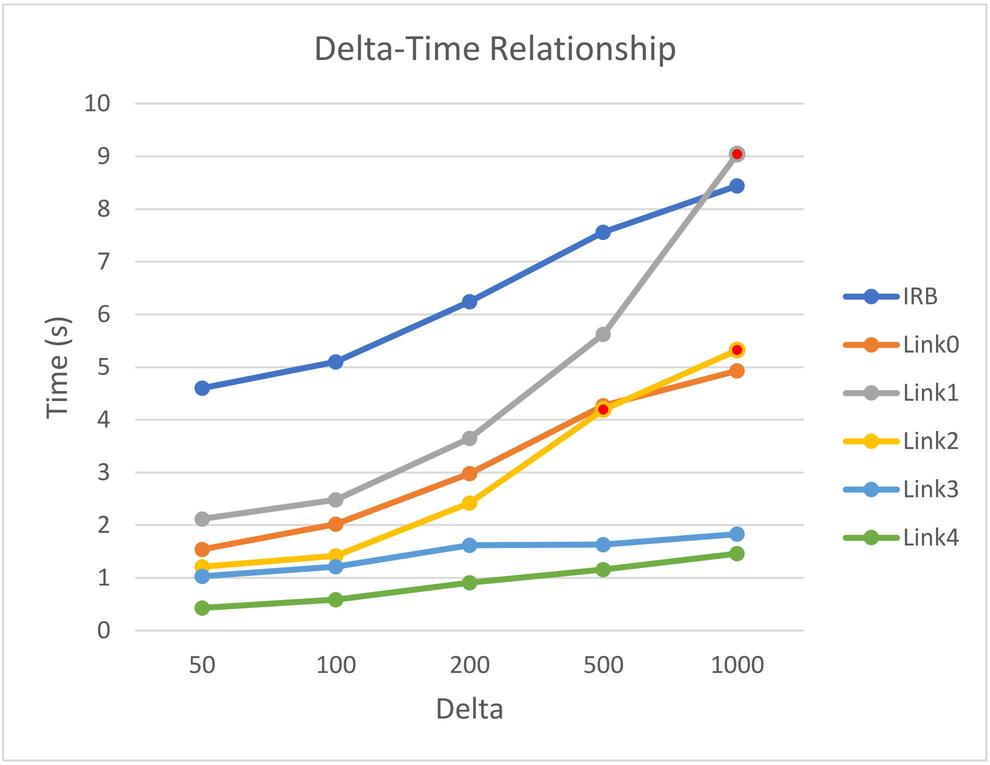


Figure 59 – Relationship δ-Time

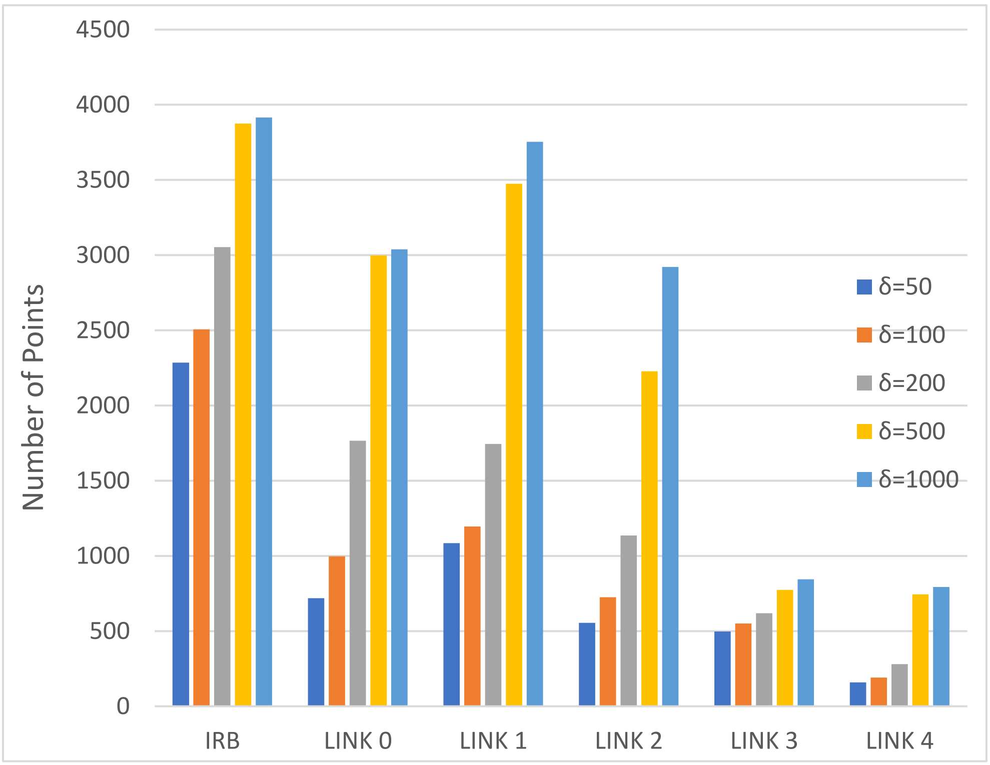


Figure 60 – Number of points analysis

The following table summarizes the tests results from the point of view of the computational time.

Object	Delta (mm)	N. of points	Iterations	Avg. Time (s)
IRB	50	2285	122	4,6
	100	2506	124	5,1
	200	3053	127	6,24
	500	3875	137	7,56
	1000	3915	142	8,44
LINK 0	50	718	158	1,54
	100	997	167	2,02
	200	1765	170	2,98
	500	2998	181	4,27
	1000	3038	183	4,93
LINK 1	50	1084	123	2,12
	100	1196	125	2,48
	200	1745	131	3,65
	500	3474	145	5,62
	1000	3753	300	9,04
LINK 2	50	554	193	1,21
	100	725	166	1,42
	200	1135	210	2,42
	500	2228	300	4,19
	1000	2921	300	5,32
LINK 3	50	497	132	1,03
	100	551	136	1,21
	200	618	141	1,62
	500	774	158	1,63
	1000	844	149	1,83
LINK 4	50	159	202	0,43
	100	190	278	0,59
	200	280	225	0,91
	500	744	215	1,16
	1000	794	285	1,46

3.5 Conclusions

The presented results testify the efficiency and stability of the implemented solution, in particular in presence of small displacements, which means that this method can provide a good answer to calibration problems.

However, as already said, difficulties have been encountered in some particularly tricky cases, as with Link 5 and Link 6 and, on a smaller scale, also with Link 4. A possible solution to this kind of issues, which are due to the geometry's characteristics, would be to introduce the *hierarchical alignment.*

Hierarchical alignment

In many practical cases objects present in the environment are not independent from each other, indeed there are physical constraints that tie the position of an object to the one assumed by another element of the environment. Considering, for example, the situation analysed in the tests, it appears clear how the position of a robot's link cannot be completely independent from other links' positions. Indeed, the fact that they have to be connected each other introduces a series of constraints in the pose they can assume. In other words, it means that it's possible to determine a *hierarchy of objects*, where the position of a *child* object depends on its *father*'s location (e.g. Link4 has to be physically connected to Link3).

The possible improvement would be to introduce a *hierarchical alignment*, with which a child object is aligned accordingly to the position of its father and exploiting int his way the physical constraints to simplify the alignment process. For example, by adopting this new alignment technique, the scenario depicted in *Figure 65,* where is shown the error in Link4's alignment, could be avoided. Indeed, taking into account the physical constraint, which states that Link4 should be connected to its father (Link3), it would not be possible to fall into this alignment error.

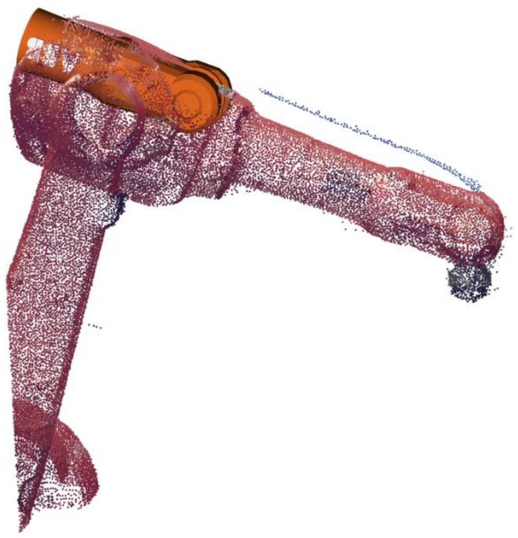


Figure 61 – Link 4 problem

Moreover, the hierarchical alignment could also be useful in Link5 and Link6 cases, indeed, by performing a first rough alignment of those links based on the physical constraints that tie them to Link4 and then utilizing the PCA algorithm just for small adjustments it may be possible to overcome the problems of low point cloud's resolution and small object's dimension.

4. Closing Remarks

In this work has been presented two promising methods to align virtual geometries. Both approaches have pros and cons and can be suitable for different practical cases of application. In particular, the solution using collaborative robots doesn't require the deployment of additional equipment and it can be applied in all the working environments. Indeed. the inspection phase is performed by using directly the cobot and collected measurements are not effected by disturbances coming from the surrounding environment. On the other hand, this measurement mode provides a small set of inspections, meaning that the information available in the alignment phase are limited.
The method involving point clouds instead, provides a very large set of information which can be used by the alignment algorithm, but, on the other hand, it requires the deployment of the 3D sensors technology needed to acquire the point cloud. Moreover, in order to obtained a consistent and precise point cloud congenial environmental conditions are needed, which means that this solution can't fit all the application cases.

References

[1] International Federation of Robotics, “IFR International Federation of Robotics,” [Online]. Available: http://www.ifr.org.

[2] International Federation of Robotics, “IFR International Federation of Robotics,” 2017. [Online]. Available: https://ifr.org/downloads/press/Executive_Summary_WR_2017_Industrial_Robots.pdf.

[3] S. M. LaValle, Planning Algorithms, Cambridge University Press, 2006.

[4] Industrial Path Solutions Sweden AB, “Industrial Path Solutions,” [Online]. Available: http://industrialpathsolutions.se/.

[5] Fraunhofer-Chalmers Centre, “Fraunhofer-Chalmers Centre for Industrial Mathematics (FCC),” [Online]. Available: http://www.fcc.chalmers.se.

[6] J. S. Carlson, R. Söderberg, R. Bohlin, L. Lindkvist and T. Hermansson, “Non-nominal Path Planning of Assembly Processes,” in *ASME 2005 International Mechanical Engineering Congress and Exposition*, Orlando, Florida, 2005.

[7] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, Principle of Robot Motion, The MIT Press, 2005.

[8] VINNOVA, “VINNOVA - Sweden's Innovation Agency,” [Online]. Available: https://www.vinnova.se.

[9] P. J. Besl and N. D. McKay, “A method for registration of 3-D shapes,” *IEEE Transanctions on Pattern Analysis and Machine Intelligence,* vol. 14, no. 2, pp. 239-258, 1992.

[10] K. S. Arun, T. S. Huang and S. D. Blostein, “Least-squares fitting of two 3-D point sets,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vols. PAMI-9, no. 5, pp. 698-700, 1987.

[11] B. Jian and B. C. Vemuri, “Robust Point Set Registration Using Gaussian Mixture Models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence ,* vol. 33, no. 8, pp. 1633 - 1645, 2011.

[12] O. D. Faugeras and M. Hebert, “The representation, recognition, and locating of 3-D objects,” *The International Journal of Robotic Research,* vol. 5, no. 3, pp. 27-52, 1986.

[13] B. K. P. Horn, “Closed-form solution of absolute orientation using unit quaternions,” *Journal of the Optical Society of America A,* vol. 4, no. 4, pp. 629-642, 1987.

[14] B. K. P. Horn, H. M. Hilden and S. Negahdaripour, “Closed-form solution of absolute orientation using orthonormal matrices,” *Journal of the Optical Society of America A,* vol. 4, no. 4, pp. 629-642, 1987.

[15] M. W. Walker, R. A. Volz and L. Shao, “Estimasting 3-D location parameters using dual number quaternions,” *CVGIP: IMage Understanding,* vol. 54, no. 3, pp. 358-367, 1991.

[16] A. Lorusso, D. W. Eggert and R. B. Fisher, “A comparison of four algorithms for estimating 3-D rigid transfromations,” *Machine Vision and Application,* vol. 9, no. 5-6, pp. 272-290, 1997.

[17] O. Sorkine-Hornung and M. Rabinovich, “Least-Squares Rigid Motion Using SVD - Interactive Geometry Lab,” [Online]. Available: https://igl.ethz.ch/projects/ARAP/svd_rot.pdf.

[18] S. Gold, A. Rangarajan, C.-P. Lu, S. Pappu and E. Mjolsness, “New algorithms for 2D and 3D point matching: pose estimation and correspondence,” *Pattern Recognition,* vol. 31, no. 8, pp. 1019-1031, 1998.

[19] A. Myronenko and . X. Song, “Point Set Registration: Coherent Point Drift,” *IEEE Transactions on Pattern Analysis and Machine Intelligence ,* vol. 32, no. 12, pp. 2262 - 2275, 2010.

[20] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm,” in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling* , Quebec City, Quebec, Canada, 2001.

[21] iMatix Corporation, “ZeroMQ,” [Online]. Available: http://www.zeromq.org/.

[22] D. Crockford, “Introducing JSON,” [Online]. Available: https://www.json.org/.

[23] ECMA International, “ECMA-404 The JSON Data Interchange Syntax,” December 2017. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[24] Open Source Robotics Foundation, “Robot Operating System (ROS),” [Online]. Available: http://www.ros.org.

[25] Allied Business Intelligence, “ABI Research,” [Online]. Available: https://www.abiresearch.com/.

[26] T. M. Anandan, “Robotic Industries Association,” 19 5 2016. [Online]. Available: https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/The-Business-of-Automation-Betting-on-Robots/content_id/6076.

[27] Universal Robots, “Universal Robots,” [Online]. Available: https://www.universal-robots.com/.

[28] J. v. d. Zijp, “FOX Toolkit,” [Online]. Available: http://www.fox-toolkit.org/.

[29] C. Larsen, “Including Collaborative Robot in Digital Twin Manufacturing System,” 2018.

[30] M. Weinmann, Reconstruction and Analysis of 3D Scenes, Karlsruhe: Springer, 2015.

[31] ATS Advanced Technical Solutions AB, “ATS,” [Online]. Available: http://www.ats.se.

[32] C. Harris and M. Stephens, “A Combined Corner and Edge Detector,” in *Proceedings of the 4th Alvey Vision Conference*, 1988.

[33] S. M. Smith and M. J. Brady, “SUSAN - A New Approach to Low Level Image Processing,” *International Journal of Computer Vision,* vol. 23, no. 1, pp. 45-78, 1997.

[34] E. Rosten and T. Drummond, “Machine Learning for High-Speed,” *European Conference on Computer Vision,* pp. 430-443, 2006.

[35] H. Bay, T. Tuytelaars and L. Van Gool, “SURF: Speeded Up Robust Features,” *European Conference on Computer Vision,* pp. 404-417, 2006.

[36] D. G. Lowe, “Object Recognition from Local Scale-Invariant Features,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision* , Kerkyra, Greece, 1999.

[37] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” *IEEE International Conference on Robotics and Automation ,* 2011.

[38] Open Pernception, Inc., “Point Cloud Library,” [Online]. Available: http://www.pointclouds.org/.

[39] J. W. Eaton, “GNU Octave,” [Online]. Available: https://www.gnu.org/software/octave/.

[40] The MathWorks, Inc., [Online]. Available: https://www.mathworks.com/products/matlab.html.

[41] M. A. Peshkin, J. E. Colgate, W. Wannasuphoprasit, C. A. Moore, R. B. Gillespie and P. Akella, “Cobot Architecture,” *IEEE Transactions on Robotics and Automation ,* vol. 17, no. 4, pp. 377-390, 2001.

[42] J. E. Colgate, M. A. Peshkin and W. Wannasuphoprasit, “Nonholonomic Haptic Display,” in *Proceedings of IEEE International Conference on Robotics and Automation*, Minneapolis, MN, USA, 1996.

[43] J. E. Colgate, W. Wannasuphoprasit and M. A. Peshkin, “Cobots: Robots for Collaboration with Human Operators,” in *Proceedings of the ASME Dynamic Systems and Control Division*, Atlanta, GA, USA, 1996.