POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

Master of Science in Mechanical Engineering

POLITECNICO
MILANO 1863

BEST VIEW METHODOLOGY ENHANCED BY BAYESIAN OPTIMIZATION FOR ROBOTIC MOTION PLANNING IN QUALITY INSPECTION TASKS

Supervisor:

Prof. Giuseppe Bucca

Co-Supervisor:

Eng. Ph.D. Loris Roveda

Candidates:

Marco Maroni
En. No. 919691

Loris Praolini
En. No. 920676

Academic Year: 2019-2020

The truth in life is that we will all make choices and eventually regret them. It is called failure, and it can be something good because it teaches how to be better people. As we age, our goal should be to minimize the number of failures that leads to repentance. In the end, we are our choices.

Jeff Bezos

La verità nella vita è che tutti faremo delle scelte e alla fine ce ne pentiremo. Si chiama fallimento, e può essere una cosa positiva perché ci insegna come essere persone migliori. Invecchiando il nostro obiettivo dovrebbe essere quello di ridurre al minimo il numero di fallimenti che portano a pentirsi. Alla fine siamo le nostre scelte.

Jeff Bezos

Ringraziamenti

A conclusione di questo lavoro di Tesi, è doveroso porre i miei più sentiti ringraziamenti alle persone che ho avuto modo di conoscere in questo importante periodo della mia vita e che mi hanno aiutato a crescere sia dal punto di vista professionale che umano. E' difficile in poche righe ricordare tutte le persone che, a vario titolo, hanno contribuito a rendere migliore questo periodo.

Un ringraziamento necessario al Professor Giuseppe Bucca, relatore di questo elaborato, per l'esempio che mi ha dato, e che io considero indimenticabile, di come sia possibile amare e far amare lo studio e la conoscenza di una materia non sempre facile, attraverso dialoghi e confronti mai banali ma anzi sempre costruttivi. È un bagaglio importante, sia culturale che di impostazione mentale, che porterò ovunque con me.

Al correlatore, Ingegner Loris Roveda, per la capacità che ha dimostrato di saper stimolare il mio interesse per l'argomento qui discusso, di mostrarmi aspetti relativi allo stesso che senza di lui probabilmente avrei ignorato, e infine, per la sua pazienza, fiducia ed entusiasmo che mi ha trasmesso.

Al Professor Mario Covarrubias Rodriguez, per aver messo a disposizioni gli spazi del Suo laboratorio per condurre gli esperimenti necessari allo svolgimento dell'elaborato quando le condizioni al contorno, per svariati motivi, non lo avrebbero permesso.

Grazie

Contents

List of Plots

List of Tables

Abstract

Combined robot-vision systems are increasingly popular nowadays, leading to a wide range of manufacturing tasks, such as locating tools, inspecting parts geometry, and checking alignments in assemblies. Although these systems are developing in many fields of application, their huge potential is only partially exploited. The quality control of mounted pieces using a sensor attached to a robot is the field in which this algorithm has been applied. The development of an approach, capable of choosing the best vision system pose for a quality control task with motion planning verification, is the goal this work. The entire algorithm coded is divided in two macro sections. The Offline Architecture, that predicts the best sensor pose in function of environment model and camera location. The Online Architecture, to carry out pose estimation of inspecting piece while controlling robot's movement inside the real environment, avoiding any kind of collisions, also not expected and forecast. To validate the code, a series of experiments have been conducted using Franka Emika Panda robot. Results shows that the algorithm, enhanced by Bayesian Optimization, outperforms standard methods as grid point sampling, reducing drastically the numbers of data needed and computation times. In addition, this algorithm does not use the manipulator during the Offline part, allowing further optimization and the use of the robot at the same time.

Sommario

In questi ultimi anni, i sistemi combinati tra robot e sensori di visione sono sempre più diffusi, aprendo la strada allo sviluppo di un'ampia gamma di attività di produzione, come la localizzazione di strumenti, l'ispezione della geometria delle parti ed il controllo dell'allineamento all'interno di assiemi. Sebbene questi sistemi si stiano sviluppando in molti campi di applicazione, il loro enorme potenziale è solo parzialmente sfruttato. Il controllo qualità di componenti, usando un sensore montato su robot, è il contesto sul quale questo lavoro viene basato. Lo sviluppo di un algoritmo, in grado di determinare la miglior posa del sistema di visione con verifica della pianificazione del moto, è lo scopo di questa tesi. L'intero codice è diviso in due macro sezioni. L'architettura Offline, che prevede l'estrazione del miglior posizionamento del sensore in funzione della scena e del punto di vista del sensore stesso. L'architettura Online, per effettuare la stima della posa del componente pianificando i movimenti del robot all'interno dell'ambiente, evitando qualsiasi tipo di collisione, anche non prevista. La validazione dell'algoritmo passa attraverso una serie di esperimenti effettuati con il manipolatore Panda di Franka Emika. I risultati dimostrano che il codice, perfezionato attraverso l'utilizzo dell'ottimizzazione Bayesiana, supera in termini di performance metodi standard quali campionamento di punti, riducendo i dati necessari e i tempi di calcolo. In aggiunta, questo algoritmo non utilizza il robot durante la parte Offline, permettendo ulteriore ottimizzazione e l'utilizzo del manipolatore nello stesso tempo.

Chapter 1

Introduction

1.1 Aims of the Thesis

The integration of 3D vision systems is a crucial aspect for robot environment interaction, allowing reconstruction of the scene in which the manipulator works, making the robot more conscious. In recent years, many industrial applications are gearing up for the use of combined robotic and vision systems to keep pace with technological development. A critical aspect for many companies is the quality control of their product, as it is done with obsolete or manual techniques. This is the context from which this work forms, trying to cope with a need that is, often, underestimated.

For industrial and manufacturing businesses, the cost of assembly and installation errors may be as much as thirty percentage of reported failure and associated repair costs. An accurate manual quality control of the several parts in the industrial assembly would require an enormous amount of work hours compared to the profit that the company would derive from it. For this reason it is often carried out quickly and very roughly. The use of automated systems instead, is often exploited in the wrong way, making the system process a huge amount of data, without taking into account whether that data is actually useful. A solution for this type of trouble is the implementation of algorithm that allows to avoid wasting time and totally automates the process. For data collec-

tion, the main aspect to consider is the vision system pose, since only a partial of the entire set of data acquirable by the sensor, are effectively useful for the task. In many real applications, the setting of the vision system pose can be carry out manually by the operator, but that would require huge idle times and user experience.

Algorithms capable of identifying, in a short time, the most effective vision system pose to carry out quality control could be the turning point for many real applications. In this thesis an approach based on a 3D vision system mounted on the end effector of a robot arm is presented, which through the estimation of the optimal vision system pose, carried out by our algorithm, allows to overcome the issue of time consuming, automating the entire process and obtaining a fair quality control. Mounting the vision system on the end effector gives the advantage that it can be moved by the robot to an optimal viewing position.

In particular, this work analyses the response of the system, when it is used to carry out a quality control of industrial parts, using a feature-based algorithm that output the 3D pose of the piece sought.

1.2 Thesis Structure

The thesis presents in the current Chapter, a summary of the state of art inherent to Pose Estimation techniques and a section explaining Next Best View algorithm. In Chapter 2, the chosen Vision System description is provided, in addition to a review of triangulation principle and the sensors performance in state of art.

Continuing in Chapter 3, a briefly description of the main Hardware and Software components used is presented, meaning explanation of the robot's characteristics and control programs. Finally, the coded Offline and Online algorithm architecture are explained in Chapter 4 and Chapter 5 respectively. In these two parts the explanation of the various nodes, with their functionalities, which form the written algorithm, are reported. Finally in Chapter 6 the summary of the entire set of experiments conducted to validate our algorithm is reported.

1.3 State of the Art

1.3.1 Pose Estimation

Pose estimation of 3D objects is a crucial task toward flexible and reliable highly complex autonomous systems. It represents the basis for object inspection, grasping or manipulator system. Visual image processing, object detection and localization are considered as an essential part of object recognition and scene understanding problems, representing one of the main motivations and research directions in the computer vision field. The availability of low cost RGB-D sensors enabled the development of novel methods that can infer scale and pose of the object more accurately even in presence of occlusions and clutter. The main pose estimation techniques can be categorized in feature-based, template matching and machine learning methods.

Feature-based methods, in which the object recognition is based on the use of 3D data, are commonly divided in two groups: local and global methods. The local ones are based on matching descriptors of local surface characteristics and include three main stages: 3D keypoints detection, local surface feature description and surface matching. The first phase is the major one in which a set of points are labelled as keypoints in function of the detection methods exploited (surface sparse sampling, mesh decimation, fixed-scale, adaptive-scale...). These points will be the ones on which object recognition will be based. Once a keypoint has been detected, geometric information of the local surface around the keypoint can be extracted and encoded into a feature descriptor. According to the approaches employed to construct the feature descriptors, it is possible to classify the existing methods into three broad categories: signature based, histogram based, and transform based methods.

Finally, the surface matching step establishes a set of feature correspondences between the scene and the model by matching the scene features against the model features. A comprehensive survey of these existing methods is described in [1]. Global feature-based methods instead,

follow a different pipeline for which the whole object surface is described by a single or small set of descriptors. Global point cloud descriptor is explained in an exhaustive manner in [2].

In general, local feature-based techniques are more robust to clutter and partial occlusions, that are frequently present in real-world application, while global feature-based methods are suitable for model retrieval and 3D shape classification, especially with the weak geometric structure. An approach based on the exploitation of both techniques, using the PCL library, is presented in [3].

In another direction, template matching techniques have also been proposed for RGB-D data, an example is [4] in which a method based on quantized surface normal as depth cue is proposed. In a similar fashion, recently, Hodan et al. [5] applied the concept of multimodal matching of [4] on an efficient cascade-style evaluation strategy.

In conclusion, even techniques based on supervised machine learning have been used for object recognition and pose estimation on RGB-D data. In [6] a review of classification techniques used in supervised machine learning is described, explaining how the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features and the different classification techniques. One of the methods to perform pose estimation is represented by a deep learning approach to category-level 3D object pose tracking on RGB-D data with the use of key points [7]. This algorithm tracks in real time novel object instances of known object categories such as bowls, laptops, and mugs, it learns to compactly represent an object by a handful of 3D key points, based on which the interframe motion of an object instance can be estimated through key point matching.

Another algorithm for real-time 6 DOF pose estimation and tracking of rigid 3D objects, uses a monocular RGB camera [8]. The key idea is to derive a region-based cost function using temporally consistent local color histograms. While such region-based cost functions are commonly optimized using first-order gradient descent techniques, in this paper a

Gauss-Newton optimization scheme is proposed, which gives rise to drastically faster convergence and highly accurate and robust tracking performance. In numerous experiments they demonstrate that the proposed Gauss-Newton approach outperforms existing approaches in presence of cluttered backgrounds, heterogeneous objects and partial occlusions.

HybridPose [9] instead, leverages on multiple intermediate representations to express the geometric information in the input image for pose estimation. In addition to key points, this type of algorithm integrates a prediction network that outputs edge vectors between adjacent key points. As most objects possess a partial reflection symmetry, HybridPose also utilizes predicted dense pixel-wise correspondences that reflect the underlying symmetric relations between pixels.

This hybrid representation enjoys a multitude of advantages. First, HybridPose integrates more signals in the input image: edge vectors include object skeleton structure and symmetry correspondences incorporate interior details. Second, it offers far more constraints than using key-points alone for pose regression, enabling accurate pose prediction even if significant fractions of predicted element are under occlusions. In addition, symmetry correspondences stabilize the rotation component of pose prediction, especially along the normal direction of the reflection plane.

Another work wants to demonstrate that neural networks coupled with a local voting-based approach can be used to perform reliable 3D object detection and pose estimation under clutter and occlusion [10]. An investigation on Deeply learning descriptive features from local RGB-D patches was done and use them afterwards to create hypotheses in the 6D pose space was.

In practice, they train a convolutional autoencoder using random patches from RGB-D images with the goal of descriptor regression. With this network codebooks were created from synthetic patches sampled from object views where each codebook entry holds a local 6D pose vote. In detection phase patches was sampled in the input image on a regular

grid, computing their descriptors and match them against codebooks with an approximate k-NN search. Matching returns a certain number of candidate votes which are cast only if their matching score surpasses a determined threshold. This algorithm allows training on real data, efficient matching between synthetic and real patches and that it can perform a good generalization on unseen data with an extremely high recall.

1.3.2 Next Best View Estimation

The recognition of objects within an everyday human environment is a challenging task for autonomous mobile robots. Object locations are continuously changing, which is why the robot cannot simply rely on fixed set of views from which it can observe an object but it needs to plan in a continuous space where to stand and where to look. The robot must select a view from the infinite set of possible views which allows to observe the sought object. Matters are further complicated by dynamic obstacles which might hinder the robot to take a particular view, or relevant features of an object can be occluded by other objects and/or by the object itself (self-occlusion). Finally, other conditions such as lighting and sensor noise can influence the performance of object recognition tasks. For these reasons, the active planning of the views from which an object could be perceived can significantly improve the overall performance of the activity. The approach, relating to view planning [11], is based on online aspect graphs and selects the next best view after having identified an initial object candidate. This approach is mainly composed of two phases: in the first one the visibility of the "candidate" object is analyzed from a series of "candidate" views reachable by the robot. Subsequently, the visibility of object features is analyzed by projecting the model of the most likely object into the scene.

The next best view planning is based on using a realistic sensor model of an RGB-D camera to generate online aspect graphs and takes the kinematic constraints of a mobile robot platform into account. "Aspect Graph" is a simulation method that allows to estimate which features of the object can be seen from different angles around the robot.The Figure 1.1 represents in a synthetic way the main steps of the approach. The fist part relates to the analysis of the environment, in this phase the available poses are evaluated, i.e. those reachable by the robot within the workspace, which allow good visibility of the object. These poses are then included in the analysis of the model. Within this phase, the visibility of the main characteristics of the object is evaluated. The combination of these two analysis determines the Next Best View.

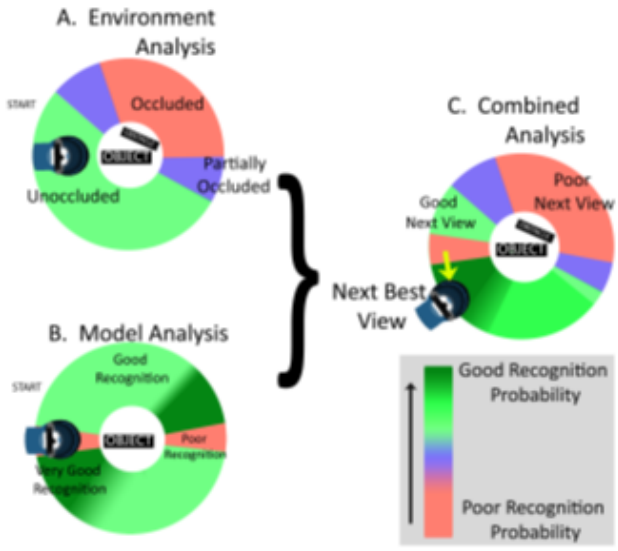

Figure 1.1: Next Best View Steps

A hypothesis about an object's identity and its estimated pose are the inputs into the planner. As output, the planner provides the next best view from which it determines the robot has the best chance of identifying the object. The view planning process carried out after receiving the identity of a candidate can be summarized in the following points:

- Potential viewing locations are checked for dynamic obstacles on the local cost map.
- Views that are reachable by the robot are subjected to an environment analysis to determine if any visual occlusion block the view of the candidate object.
- Views which survive environmental analysis undergo model analysis to determine the amount of visible surface area from each point of view.

After the environmental and model analysis, at each pose is assigned a score representing its visual coverage of the candidate object, and the pose with the highest score is defined as the Next Best View, which will be sent to the component of robot navigation. Once in the new location, the robot will accept or reject the initial identity estimate or

start determining another next best view if the first one did not lead to recognition. The results of the experiments carried out show that this type of approach improves the performance of object recognition compared to that performed in a single view, and also allows the robot to better differentiate between objects that have similar characteristics.

Chapter 2

Vision System

2.1 Sensor's Overview

A 3D vision system, from hardware point of view, is a transducer that measures the light intensity and produces depth image in function of it. These devices usually produce simple depth images or videos but can also generate point clouds or meshes. They can also be considered shape acquisition devices as they return x, y and z coordinates of the 3D objects. For this work, the choice of the sensor is a fundamental aspect, allowing the manipulator to locate the target object, whose geometry is known, and to determine its position and orientation with respect to the robot reference frame, in order to verify the correct execution of the assembly phase in the industrial process.

Today exists different techniques to acquire depth information from the scene based on triangulation principle, as in Figure 2.1. This method estimates the depth by observing the target from different perspectives. Figure 2.1 shows a simplified model for the triangulation principle. It simulates a stereoscopic sensor in which the two cameras (with sensor center in O_l and O_r respectively) have the same focal length, parallel optical axes and observe a point P located a distance z. The figure also underlines p_l and p_r that represents the optical projections of P on the left and right camera plane. The depth z can be estimated through

Equation 2.1, pointing out the inverse proportionality with the disparity d (x_r - x_l), fundamental concept that represents the horizontal parallax between projections p_l and p_r.

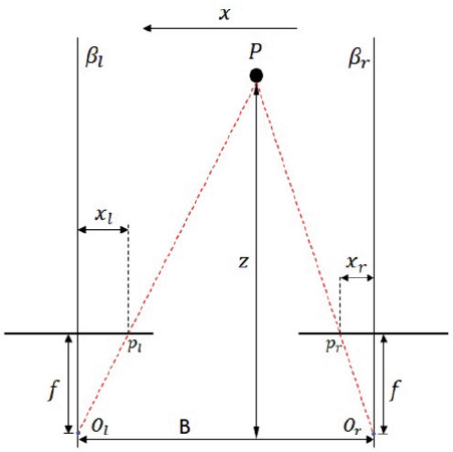

Figure 2.1: Depth Estimation through Triangulation Principle

$$z = \frac{B \cdot f}{x_r - x_l} \tag{2.1}$$

B indicates the baseline, optical centers distance, and f the focal length of the sensors. Stereoscopic techniques are generally grouped into two main categories, Active and Passive, in function of the sensor's working principle; the latter, like pure Stereoscopy, perform depth measurements using two cameras to triangulate over homologous key points on the scene. In these sensors the triangulation is created between the object and two sensors, then an algorithm is implemented to establish the correspondence between image features in different views of the scene and to calculate the relative displacement between features coordinates in each image. Active methods instead enhance the acquisition of depth information through controlled source of structured energy emission, such as scanning laser source or projected pattern of light.

The two major families of devices that use this method are Structured Light sensors, that perform triangulation through the use of a single camera and projecting an IR laser, and Active Stereoscopy sensors in which the triangulation is done by two different cameras and the projected laser is used to better represent the scene in critical contest.

Furthermore, a new class of active depth sensing system based on the time-of-flight (ToF) principle is emerging on the market. These devices estimate the depth by sending lighting signals on the scene and measuring the time that light signal goes back-and-forth. ToF cameras based on the principle of pulsed light sources measure the time that it takes for a light pulse to travel from the emitter to the scene and then back after reflection, while Time-of-Flight devices that use continuous waves detects the phase shift of the reflected light, so modulating the amplitude is possible to create a light source of sinusoidal form with desired frequency and estimate the phase shift through the detector. In Figure 2.2 the triangulation methodologies introduced above are represented in a simple way.

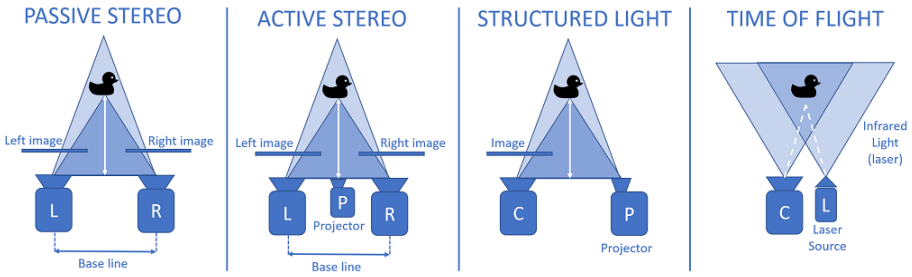

Figure 2.2: Triangulation Methods

The devices used to carry out the tests for the thesis are both active stereoscopic sensors. This choice is mainly due to the fact that this type of sensors responds better to situations in which the context is not well defined (texture-less and dark areas) and faster in their work space (0.5-4.5m) than passive sensors. For these reasons, the choice of an active sensor compared to a passive one seems more logical for this kind of application.

In detail, the sensors used are Intel RealSense D415 and Intel RealSense D435. Both are active stereoscopic sensor with the possibility to change manually the laser power value, having the opportunity not only to compare the two sensors but also to compare Active and Passive principle, simply by turning off the laser.

2.2 Intel® RealSense D400 Series

Intel® RealSense Depth Camera D400 Series is a family of depth camera based on infrared active stereoscopy technology, in which the depth is estimated in-hardware, as a proportional inverse of the pixel disparity from the left IR image to the right IR image. Following Eq. 2.1 introduced above, through an imaging ASIC (Application Specific Integrated Circuit) that processes the infrared streams together with the RGB one, performs frames correlation with census cost function to identify homologous point and reconstructs the disparity.

Figure 2.3: Intel RealSense D435 in the foreground and D415 in the background

The infrared projector enhances the stereo camera system's ability to determine depth, by projecting a static infrared pattern onto the scene to increase its texture. A comparison between passive and active stereoscopy is presented in Figure 2.4, in which is possible to understand how areas characterized by no depth information, are drastically decreased by the IR projection. This active stereo depth computation makes these depth cameras suitable for acquisitions both indoors and outdoors under reasonable illumination.

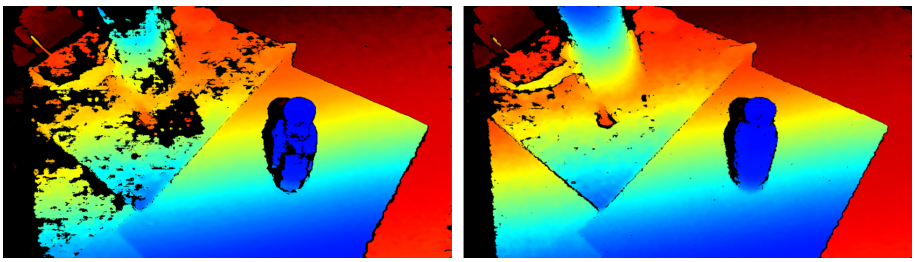

Figure 2.4: Passive and Active Stereoscopy Evaluation

The depth data generated with stereo vision uses the left image as the reference for stereo matching, resulting in a non-overlapping region on the field of view as is visible in Figure 2.5. Therefore, there is no depth data at the left edge of the frame. Closer scenes result in a wider invalid depth band than scenes at further distances.

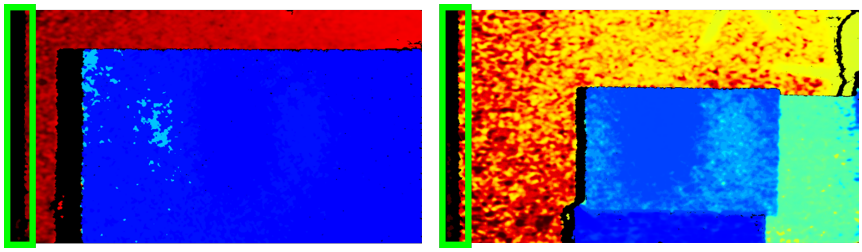

Figure 2.5: D415 No Depth Data due to Non-Overlappping Region of FOV at 400 mm distance (*left*) and 800 mm (*right*)

Both the sensors estimate depth with an active stereoscopic technology but with two different intents. The D415 is intended for finer reconstructions in a narrow field-of-view, whereas the D435 is intended for acquisitions over a much wider field-of-view, but with a worst spatial resolution. This stands out immediately from the cameras' technical data as reported in table below: the D415 has a larger baseline and a greater focal length than the D435. Also, it is worth noting that the D435 has a global shutter sensor, whereas the D415 has a rolling-shutter one, hence the latter is expected to be more accurate when dealing with static scenes, but would not perform well in highly dynamic scene.

	D415	D435	MU
IR Camera Resolution	1280×720	1280×720	pixel
RGB Camera Resolution	1920×1080	1280×720	pixel
Maximum Frame Rate	90	90	Hz
Optical Centers Distance (B)	55	50	mm
Horizontal FOV (HFOV)	69.4	91.2	$\circ$
Vertical FOV (VFOV)	42.5	65.5	$\circ$
Focal Length (f)	1.88	1.93	mm
Measurement Range	$0.3 - 10$	$0.2 - 10$	m

Table 2.1: Sensors Technical Data

2.3 Sensor Performances

A metrological qualification of the Intel D400 Active Stereoscopy Cameras has been carried out in [12], in that book the theoretical random error in space due to pixels spatial quantization and the real one is compared. The theoretical random error in space has been calculated considering the focal length and the baseline. The spatial resolution along the z-axis has been computed assuming a unity disparity error. Figure shows the results for both sensors in function of the depth. As is possible to see the D415 has better theoretical performances.

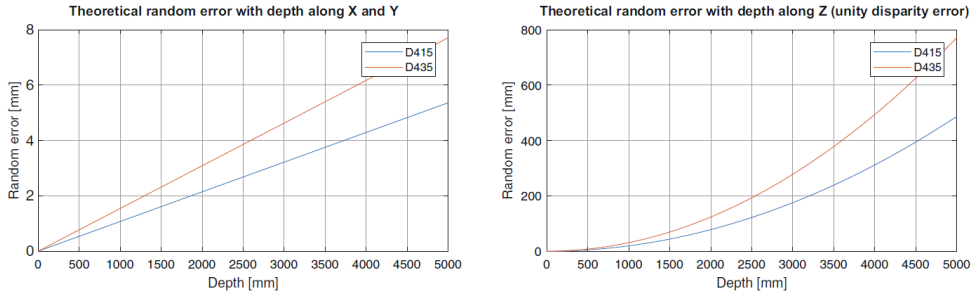

Figure 2.6: Theoretical Random Error Sensors comparison

The test to predict the random error in operating conditions has been realized fixing on the robot end effector the sensor and a target on a second robot. The random component of the uncertainty in space is evaluated along the depth and the results show that is bounded by 2 mm for distances below 1100 mm. As expected from the theoretical depth resolution the random error grows quadratically with the depth. The systematic component of the uncertainty instead has been evaluated as the difference between the displacement measured by the camera at each step and the reference one imposed by the robot arm. The bias is bounded by 5 mm for distances below 1100 mm and grows quadratically. The conclusion of the study [12] is that the D415 is more precise than D435, this is partially due to the larger FOV of the D435 that leads into a worst spatial resolution, hence a worst precision in the triangulation process.

Chapter 3

Hardware and Software

3.1 Franka Emika Panda Manipulator

The robot used to carry out the experimental phase is the Franka Emika Panda robot, with seven revolute joints, each of them closed-loop controlled and equipped with torque sensor, with maximum payload of $3\,kg$.

Figure 3.1: Panda Robot by Franka Emika

One of the main aspects of this collaborative manipulator is redundancy. Indeed, with its seven degrees of freedom, can reach the same

position in different configurations or, if a robot joint reaches its mechanical limit, probably other joints can make up for execution of the final prescribed movement. Technically, the size of the operating space is smaller than the size of the joint space. This characteristic allows also an increase on flexibility in performing tasks, and thanks to torque sensors mounted on all joints, can detect real-time applied forces. All these aspects lead to consider Franka Emika Panda as an excellent solution for carrying out assembling, testing or inspecting tasks, as in our case.

The robot can be controlled by Franka Control Interface (FCI), able to provide, via $libfranka$ interface at a maximum frequency of $1\,kHz$, joint positions and velocities, as well as link side torques. Manipulator can be controlled in different modalities, according to the user requirements:

- *torque mode*: user provides a joint torque vector to the robot motors.
- *joint position mode*: user gives the desired joint position vector.
- *velocity mode*: user defines the desired joint velocity vector.

3.2 ROS: Robot Operating System

Developing a robot with a computer brain requires a bunch of software tools on the computer side as software drivers, third party tools for computer vision and simulation tools. Instead of reinventing the wheel every time, some frameworks can help you by gathering all these tools and managing how you develop code for your robot. ROS is one of these frameworks initially developed by the Stanford AI Laboratory in 2007 for developing robots. The Open Source Robotics Foundation now maintains ROS.

The concept of ROS goes far beyond just a framework. ROS is an OS in concept, because it provides all the services that any other OS does, as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. Even though ROS is still a framework that isn't a standalone operating system and isn't the only framework for robots, it seems to be adopted widely and have a large developers community.

The Robot Operating System, best known as ROS, is a very powerful collection of libraries and tools aimed at configuring and writing robots' software. In this work, ROS Melodic, LTS version, has been installed on Ubuntu 18.04. The reason why this operating system is spreading more and more, is because of the innovative way of describing and handling the many parts that are going to characterize the final behavior of the intended robot: industrial manipulators, drones as well as other mobile platforms up to highly complex humanoids with state-of-the-art AI capabilities. In practice, this result has been achieved with a modular programming method, stimulating collaborative work. As a matter of fact, ROS is organized mainly in nodes, topics and messages. A programmer builds a node, a mini-core of the overall structured software, that adds one or more particular capabilities and is able to send and receive messages through channels called topics.

To keep communication organized, a master node handles the most

basic functions such as subscriptions and publications on topics. In broad terms, this application requires to read mechanical values from the arm, to elaborate suitable trajectories, to collect data from an RGB-D camera and to detect a specific object. All of these will be the main cores of the software, and they will exchange information or services thanks to proper messages that the sender is able to write and the listener is able to receive and process.

The following are some of the key terms for understanding ROS:

- *ROSCORE*: master coordination node, it administrates all the nodes connected to it, managing their parallel execution and communication between them.
- *NODE*: executable script which can be either a user-developed code or a software package provided by ROS.
- *TOPIC*: sort of temporary node in which other nodes can write and read numbers or strings.
- *MESSAGES*: specific data type sent to a topic.
- *SERVICES*: client-server type messages that expect a reply to a sent data

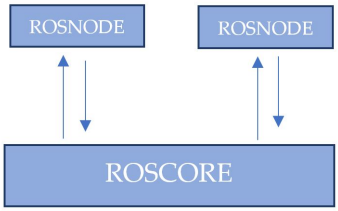

Figure 3.2: Schematic Representation of Communication between Roscore and different Nodes

The communication between nodes can be assigned to different types methodologies:

- *Publisher and Subsciber*: asynchronous communication mode in broadcasting which consists in writing a message on a topic, made

available by roscore. All nodes wishing to receive the message can request it from the roscore via the topic.

Figure 3.3: Schematic Representation of Communication between Nodes through Publisher and Subscriber

- *Service*: synchronous communication mode, according to the request and reply semantics. A node sends a request to all nodes that can fulfill it, from these nodes it will receive a response.

Figure 3.4: Schematic Representation of Communication between Nodes through Server and Client

3.3 MoveIt! and RVIZ

MoveIt! is a state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, RD and other domains. The development of open-source frameworks like ROS and MoveIt! has made robotics more accessible to new users in the last years, both in research and consumer applications. MoveIt! provides the core functionality for manipulation in ROS.

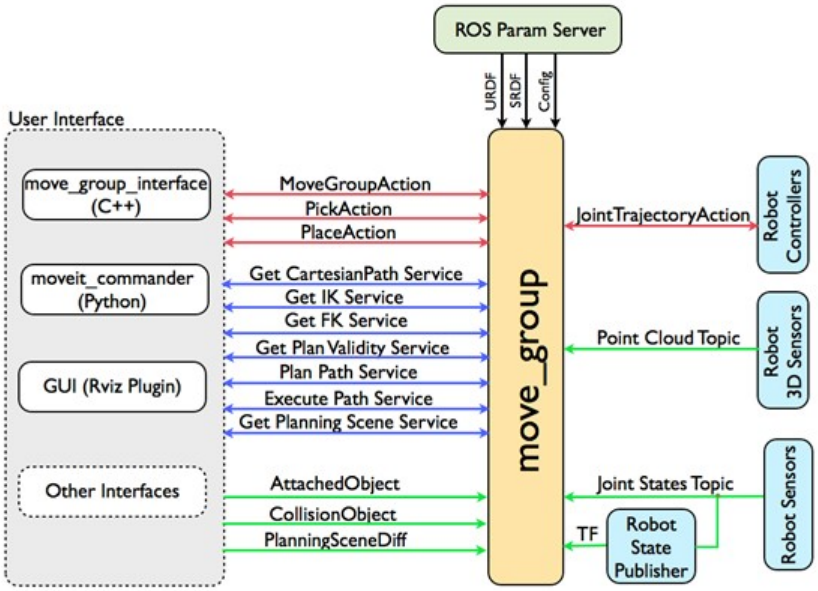

Figure 3.5: MoveIt! Architecture

In the top section of the Figure 3.5, we notice that *move_group* reads three parameters representing the model of the robot, in a format that this software is able to understand. The Unified Robot Description Format, URDF, lists all the details of the kinematic chain, links and joints, transforms tree hierarchy and whichever constraint we may set as effort, position, velocity and acceleration. Its correspondent Semantic Robot Description Format, SRDF, is generated after the first configuration is

completed, and contains the most useful information on Panda robot as joint limits, pre-defined positions, planning groups, end-effector structure and finally a list of links that are always or never in collision, in order to slim computations. From robot's sensors, the central *move_group* node always keeps track of the overall structure's position and orientation. Indeed the TF tree is refreshed at high frequencies, up to 1 kHz, to cope with dynamic environment. With obstacles spawning in arm's workspace it allows planning respecting chain's design limits and avoiding any kind of collisions. Once a trajectory is successfully planned, execution is possible. The Robot Controllers, in this case Franka Control Interface, are interfaced with the *move_group* node to perform this task.

The way in which a programmer can interact with this Moveit! architecture is basically described by the topics, actions or messages that can be exploit to communicate with the *move_group* node. Particular actions exist for a pick and place application, the user may define collision object by script and place them in the map or attached to the robot: in the latter case, the planner keeps that object into account in elaborating trajectories. Moreover, is possible to exchange information on Inverse Kinematics calculations, generated paths and planning scene structures.

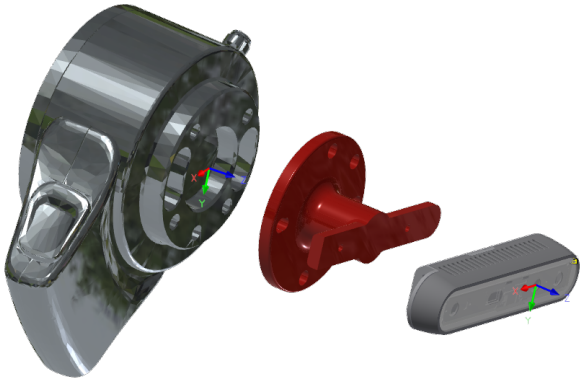

Figure 3.6: Flange Designed to mount Sensor on Robot's End Effector

RVIZ is a powerful GUI used to have graphical feedback from ROS and MoveIt! applications. It listens to various topics in order to visualize results coming from motion planning algorithms. The most basic ones allow sending commands as well as retrieve information on the overall structure of the robot as kinematic chain, parent-child relationships, and

reference frames from all the rigid bodies. This last one is the Transforms tree, TF. At this point it is important to notice that in this work application, the sensor must be mounted on robot's end-effector. To allow the assembly, a flange was designed and then 3D printed. Figure 3.6 can clarify the idea of mounting.

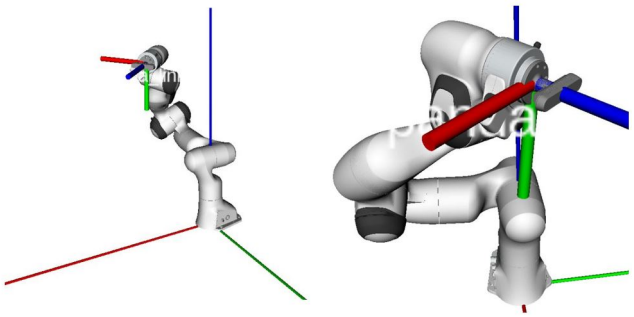

Figure 3.7: Null Robot's Reference Frame on the left, particular of End Effector TF on the right

Obviously, to consider possible self-collisions and external one, a modification to robot's URDF and SRDF was carried out. The sensor and flange collision mesh files were uploaded linking them throw geometrical TF to robot's end effector reference frame. For this purpose, two different figures are proposed. Figure 3.7 indicates the null robot reference frame at its base, and the TF of the end effector. Figure 3.8 instead aims to clarify the idea on the difference between the camera and end effector reference frames.

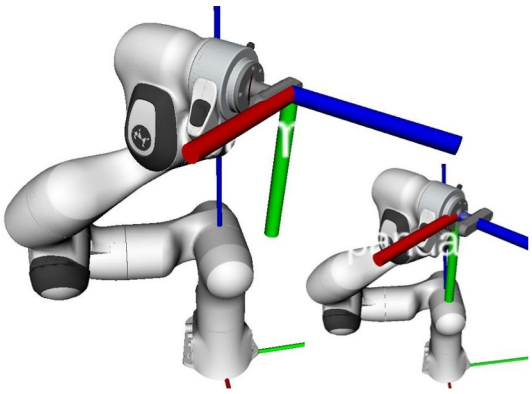

Figure 3.8: Difference between Sensor's TF and End Effector's one

Chapter 4

Planner Offline

4.1 General Overview

This part of the whole algorithm consists in sampling the best robot configuration, and consequently, sensor position and orientation to perform the pose estimation of the studied piece inside the real environment. This portion is all computer-based and no real interfacing with reality is performed. It is mainly composed into five different nodes in circular motion, each of them with a specific task to be completed. Communication among nodes is insured by customized and standard ROS messages, allowing not only data transfer but also timing optimization. The necessary inputs to this part of the code are a CAD model of the scene or a reconstructed point cloud acquisition and a CAD model of the piece to be recognized. A schematic representation of the algorithm is proposed:

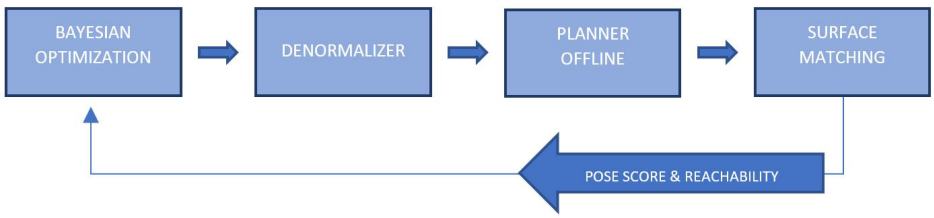

Figure 4.1: Schematic Representation of the Planner Offline Algorithm

The decision of the best robot configuration is assigned to a Bayesian

Optimization algorithm, which, through a cost function and other parameters, enables time-saving sampling of the point surrounding the object on study. Secondly, a De-Normalizer node elaborates data coming from Bayesian Optimization. It calculates from space coordinates, the position and orientation of the sensor to achieve a good visual, and finally the end effector pose to assign to robot controller. During third step, after controlling the reachability of the pose, a collision-free path for the robot is generated. Whereupon, to perform the virtual matching, a sensor view simulation algorithm is implemented to extrapolate only the surfaces that the real sensor will acquire. Obtained the cut sensor view, a surface matching algorithm returns the possible pose of piece with respect to the reference frame with an assigned score, that represents the confidence of the possible pose. Once all is done, the cycle restarts until Bayesian Optimization reaches the determinate number of iterations.

4.2 Bayesian Optimization Node

Bayesian Optimization, from now on also BO, is a model-based, black-box optimization algorithm that is tailored for very expensive objective functions. It is suited for optimization over continuous domains and tolerates stochastic noise in function evaluations. As a black-box optimization algorithm, Bayesian optimization searches for the maximum of an unknown objective function from which samples can be obtained. Like all model-based optimization algorithms, Bayesian optimization creates a model of the objective function with a regression method, uses this model to select the next point to acquire, then updates the model, etc. It is called Bayesian because, in its general formulation, this algorithm chooses the next point by computing a posterior distribution of the objective function using the likelihood of the data already acquired and a prior on the type of function. Without loss of generality, Bayesian Optimization addresses the general problem of identifying the maximum of a real valued objective function:

$$x^* = argmax f(x) \tag{4.1}$$

This problem could be solved by optimizing the objective function directly. However, when individual evaluations of the objective incur high costs, algorithms which rely on many evaluations are inappropriate. To reduce the number of function evaluations the BO approach uses a Bayesian prior model of the objective function and exploits this model to plan a sequence of objective function queries. Essentially, BO algorithm trades computational resources, expended to determine queries points, for a reduced number of objective function evaluations. The process proceeds as follows:

- A query is selected by optimizing a measure of improvement. Typically, the improvement measure incorporates an exploration strategy that directs search to poorly modeled regions of the solution space.

- The query is evaluated by the true objective function through real data gathered from the optimized system.
- The system observes the performance at the query point and updates the posterior model of the objective function.
- The process returns to the first step.

The next figure aim to clarify the idea on how Bayesian Optimization works.

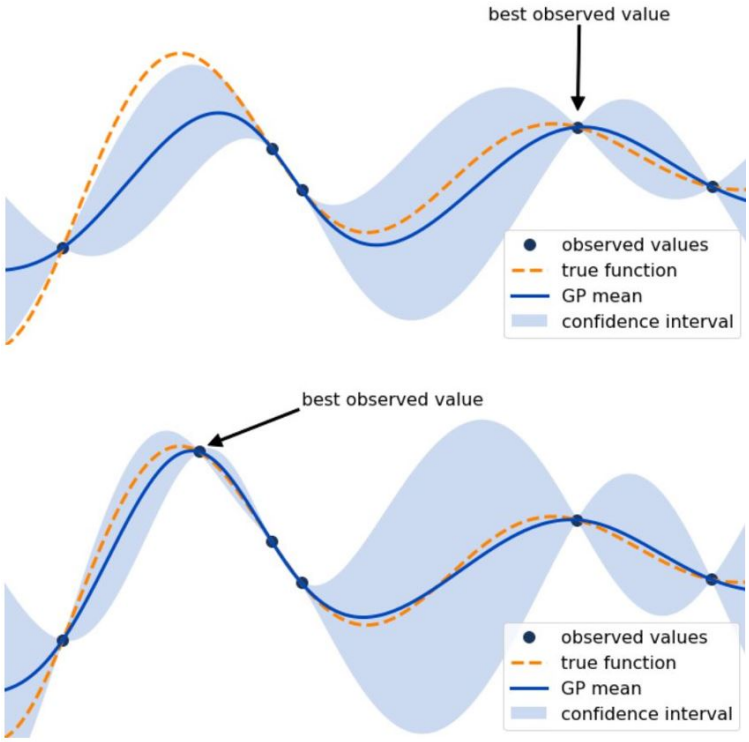

Figure 4.2: Iterative Process Representation of the Bayesian Optimization Algorithm

The implementation of Bayesian optimization uses this Gaussian process model to search for the maximum of the unknown objective function $f(x)$. It selects the next variable value to test by selecting the maximum of the acquisition function, which balances exploration, improving the model in the less explored parts of the search space, and exploitation, favoring parts that the models predicts as promising. Once an observation is made, the algorithm updates the Gaussian process to take the new data into account. In classic Bayesian optimization, the Gaussian

process is initialized with a constant mean because it is assumed that all the points of the search space are equally likely to be good. The model is progressively refined after each observation.

The developed algorithm starts choosing a set of two random coordinates. In this case, these numbers represent the two angles ϕ and θ generating a point belonging to a sphere of given a radius. The sphere representation, stated its center, allows to achieve a different prospective vision of the object to be studied. Bayesian Optimization will so generate points only belonging to the sphere, described by the coordinates ϕ and θ. Two are the parameters that in the analyzed case will be given as input to this node: reachability, which represents if the robot can reach that specific position, and pose score, representing the surface matching quality. Whereupon when the pose score and index of reachability, this node performs the optimization on the cost function:

$$J_{BO} = f(reachability, pose\,score) \tag{4.2}$$

After cost function minimization, the new set of coordinates are ready to be fed again into the algorithm. It is important to remember that the two numbers exiting from Bayesian Optimization node are normalized between zero and one, so a denormalization process is necessary.

To enhance the understanding of BO's power, it will be compared with a simple uniform sampling of ϕ and θ coordinate, creating a grid around the sphere. The comparation will be exploited both in terms of time and accuracy reaching the best scored pose.

4.3 De-Normalizer Node

As explained before, from Bayesian Optimization node a normalized vector containing ϕ and θ coordinates is obtained. In this algorithm ϕ and θ are denormalized in the given range to obtain the sphere coordinates. The range varies on the position and orientation of the piece that must be studied, remembering that all the coordinates must be referred to robot origin reference system with the aim to obtain comparable results. After that, sphere coordinates are transformed in cartesian coordinates and the sensor orientation is calculated. The expression describing sensor direction is computed in rotation matrix form, that can be also manipulated to obtain a quaternion which will be given to the robot control program.

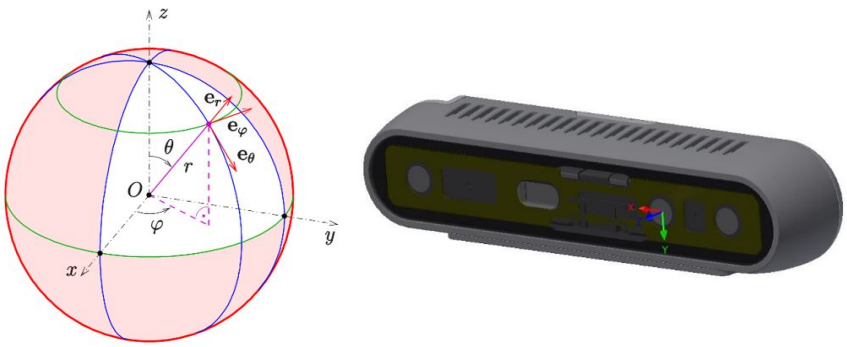

Figure 4.3: Spherical Reference System and Sensor Origin

The figure above explains the direction of the radial, tangential, and circumferential versors e_r, e_θ, e_ϕ, generated by the sphere coordinates. For having the sensor pointing towards the center of the sphere, where the object is located, a further orientation transform is needed. Imagine to constitute a right-hand reference system with the sphere versors e_r, e_θ and e_ϕ corresponding to axis x, y, and z.

Once known the sensor reference system, as in the image, it is possible to obtain a transform to get z axis pointing radially towards the sphere center and axis x along circumferential direction. The transform that allows this kind of transformation is a simple rotation among e_θ versor of a $\gamma = -90$ degrees angle. To perform the transformation, we had to define a rotation matrix using Euler theorem:

$$R(\vec{u},\gamma) = R(\vec{e_\theta}, -\frac{pi}{2}) = \tag{4.3}$$

$$\begin{bmatrix} e_{\theta_x}^2 m + cos(\gamma) & e_{\theta_x} e_{\theta_y} m - e_{\theta_z} sin(\gamma) & e_{\theta_x} e_{\theta_z} m + e_{\theta_y} sin(\gamma) \\ e_{\theta_x} e_{\theta_y} m + e_{\theta_z} sin(\gamma) & e_{\theta_y}^2 m + cos(\gamma) & e_{\theta_y} e_{\theta_z} m - e_{\theta_x} sin(\gamma) \\ e_{\theta_x} e_{\theta_z} m - e_{\theta_y} sin(\gamma) & e_{\theta_y} e_{\theta_z} m + e_{\theta_x} cos(\gamma) & e_{\theta_z}^2 m + cos(\gamma) \end{bmatrix}$$

in which $m = 1 - cos(\gamma)$ and e_{θ_x}, e_{θ_y}, e_{θ_z} are the e_θ components among the three reference axis. We can finally identify the homogeneous matrix describing the sensor orientation towards sphere center from origin as:

$$[T_{(o-sens)}] = [T_{(o-sc)}][R(\vec{u},\gamma)] \tag{4.4}$$

in which $[T_{(o-sens)}]$ represents the homogeneous matrix from origin to sphere coordinates and $[R(\vec{u},\gamma)]$) the homogeneous matrix from sphere coordinates to sensor reference system allowing the pointing towards the center of the sphere.

Another consideration that must be done represents the fact that robot planner is able to plan and execute trajectories only referring to end effector reference system. After designing and 3D printing a support to anchor the sensor to robot's end effector, the pure geometrical translation and quaternion vector between this and sensor camera reference system can be calculated:

$$\vec{t} = \begin{bmatrix} -0.02 \\ 0 \\ 0.05285 \end{bmatrix} \quad \vec{q} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Obviously, due to 3D printing errors of the flange connecting end effector to sensor, the real vectors differs from analytical ones, so a calibration procedure will be exploited. Described as $[T_{o-ee}]$ the homogeneous matrix from origin to the robot's end effector, $[T_{o-sens}]$ from origin to sensor and $[T_{ee-sens}]$ from the end effector to sensor, it is possible to determine the end effector position and orientation by:

$$[T_{o-ee}] = [T_{o-sens}][T_{ee-sens}]^{-1} \tag{4.6}$$

After the calculation, the algorithm will transform $[T_{o-ee}]$ into a vector of three translation and a quaternion for representing the orientation. Furthermore, these two elements are packed in a customized ROS message and sent to the next node, in which will be elaborated. An example of the ROS message is reported:

x	y	z	qx	qy	qz	qw
float64	float64	float64	float64	float64	float64	float64
0.5000	0.2000	0.2000	0.8212	0.0000	0.5023	0.2851

Table 4.1: Example of ROS message sent from De-normalizer Node: Sensor Position and Quaternion Orientation

4.4 Hidden Point Removal Node

The aim of this part of code which simulated the cut sensor's vision is deleting points of a complete scene CAD model which are not visible from a given viewpoint. In this way it is possible to enhance matching results dependency from sensor position. At this scope, is necessary $[T_{o-sens}]$ homogeneous matrix, representing sensor position and orientation from null reference frame.

This function is based on the work of Katz,Tal and Basri [13]. Given a set of points $P = \{p_1 \,|\, 1 \leq i \leq n\} \subset R^D$, which is considered a sampling of a continuous surface S, and a viewpoint, camera position C, the goal is to determine $\forall\, p_i \in P$ whether p_i is visible from C. The main function is divided into two steps: inversion and convex hull construction.

- *Inversion*: starting from the set of points and the viewpoint, placed on the origin, a spherical flipping is performed in order to reflect a point $p_i \in P$ with respect to the sphere accordingly to the following equation:

$$\widetilde{p_1} = f(p_i) = p_i + 2(R - ||p_i||)\frac{p_i}{||p_i||} \tag{4.7}$$

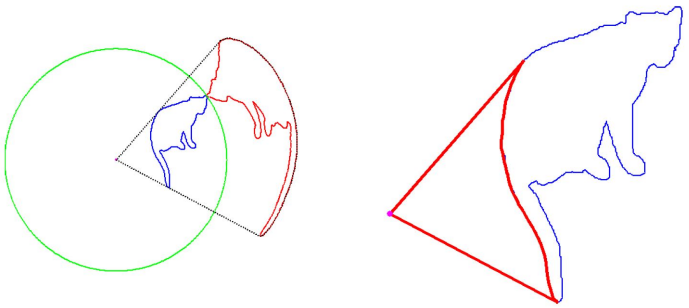

Figure 4.4: HPR Operator - Left: spherical flipping (in red) of a 2D curve (in blue) using a sphere (in green) centered at the view point (in magenta). Right: back projection of the convex hull.

Intuitively, spherical flipping reflects every point p_i internal to the sphere along the ray from C to p_i to its image outside the sphere, as shown in Figure 4.4.

- *Convex Hull Construction*: as can be understood, points can be considered visible if their reflection lies on the convex hull. Mathematically this can be expressed as a point $p_i \in P$ is marked visible from C if its inverted point $\widehat{p_i}$ lies on the convex hull of $P \cup \{C\}$.

The main features of this algorithm can be summarised as:

- Low complexity $o(nlog\,(n))$, with n the number of points in the cloud.
- All the points marked visible by HPR operator are visible from C.
- Convex shapes and slanted planes are correctly handled, while concave sections only if the curvature is sufficiently low.

The main parameter influencing the results HPR function is the radius R of the sphere. By increasing this value, the number of points that are considered not visible while they are instead visible, false negative, decreases. On the other hand, if R increases, the amount of false positive also increases.

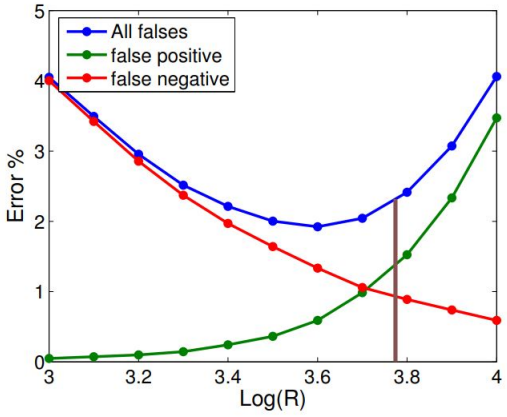

Figure 4.5: False positives,negatives and their sum, of a specific model with 70K points. The automatically calculated R is shown in brown.

One of HPR limits includes that the real sensor's field of view is not taken into account. This could lead to wrong results if the distance between sensor and the scene is too small. Secondly, the density of

the point cloud is not function of the distance of the points from the viewpoint. This could lead to different results if the density of the real point cloud differs too much with the distance.

4.5 Planner Node

The customized ROS message transfers data from the previous node to the one that is going to be analyzed. Indeed, the robot's end effector position and orientation are given as input through two vectors. The aim of this algorithm is understanding if the pose generated by the Bayesian Optimization is reachable, according to robot's joints limits and self-collision. Not only these two factors are considered, also the CAD model of the scene or a point cloud can be loaded to warn the planner of the possible real obstacles, avoiding them, if possible.

Once determined if the pose is reachable, the ROS message coming from the De-normalizer node is updated, adding a more number. Number zero will be written if the position is reachable, number -1 if not. Assume that the planner generates a -1, in this case the message is returned to Bayesian Optimization node asking for generating a new set of coordinates. In this case the cost function will be immediately calculated and the not reachability of the position will generate a huge negative number, moving away the optimizer in calculating a new position near the not reachable one. Let us now analyze the 0 case. In this instance, the number is written into the ROS message and sent to the next part of the algorithm. The ROS message now appears:

x	y	z
float64	float64	float64
0.5000	0.2000	0.2000

qx	qy	qz	qw	reach
float64	float64	float64	float64	float64
0.8212	0.0000	0.5023	0.2851	1.0000

Table 4.2: Example of ROS message sent from Planner Node: Sensor Position, Quaternion Orientation and Reachability Index

4.6 Surface Matcher Node

This part is aimed to describe how the matching score between the CAD model and the cut scene is extrapolated. For this work, two different types of feature-based algorithms have been used. The first is a commercial one, created by $MVTec^{©}$ and named HALCON, while the second is open source and called Point Pair Feature. Further investigation and experimentation on the matching quality and certainty will be exploited with some comparison between the two algorithms. A brief description of how HALCON and PPF works will be presented in the next pages. $MVTec^{©}$ HALCON is a standard software for machine vision with an integrated development environment, HDevelop, and an HelpWindow allowing an overview of the main function available. HALCON's flexible architecture facilitates rapid development of any kind of machine vision application. The software provides the latest state-of-the-art machine vision technologies, such as comprehensive 3D vision and deep learning algorithms. For our application, Surface Matching libraries has been used.

The entire code is based on *find_surface_model* operator which serves the purpose to find the best matches for a target model in a 3D working environment returning the homogeneous transformation matrix between the environment and the model. The matching of this function is based on three main steps:

- *Approximate matching*: the approximate surface model's poses inside the scene are searched. Firstly, scene points are sampled uniformly, controlling the sampling distance through the parameter *RelSamplingDistance*. Then, a set of keypoints is selected from the sampled scene points with the parameter *KeyPointFraction*. For each selected keypoint, the optimum pose of the surface model is computed under the assumption that the keypoint lies on the surface of the object. This is done by pairing the keypoints with all other sampled scene points and finding the point pairs on the surface model that have a similar distance and relative orientation.

The sampled scene pose for which the largest number of points lies on the object is considered to be the best pose for this keypoint. The number of sampled scene points on the object is considered to be the score of the pose. From all keypoints the poses with the best scores are selected and used as approximate poses.

- *Sparse pose refinement*: in this second step, the approximated poses are further refined. This will increase their accuracy and the significance of the score value. The sparse pose refinement uses the sampled scene points from the approximate matching. The pose is optimized such that the distances from the sampled scene points to the plane of the closest model point are minimal. The plane of each model point is defined as the plane perpendicular to its normal. Since each keypoint produces one pose candidate, the total number of pose candidates to be optimized is proportional to the number of key points. The score of each pose is recomputed after the sparse pose refinement.

- *Dense pose refinement*: accurately refines the poses found in the previous steps. This step works like the sparse pose refinement and minimizes the distances between the scene points and the planes of the closest model points. The difference is that only the poses with the best scores from the previous step are refined and all points from the original scene are used for the refinement. Taking all points from the scene increases the accuracy but is slower than refining on the subsampled scene points.

The final accuracy of the refined pose depends on several factors. The internal refinement algorithm has an accuracy of up to 10^{-7} times the diameter of the model. This maximal accuracy is only achieved for best possible conditions. Other factors affect the final accuracy of the algorithm such as the shape of the model, the number of points in the scene, the noise of the scene points, the visible part of the object instance and the position of the object.

From now on, the second surface matching functioning algorithm is presented. First introduced by Drost et al.[14], the Point Pair Features

voting approach is a feature-based solution combining a global modeling and a local matching stage within a local pipeline using sparse features. The method details have been explained in several publications[14][15], however, it is considered important to offer a general overview of the approach with special emphasis on some specific points. Using point cloud representations of oriented points (i.e., points with normals), the method relays on four-dimensional features extracted from pairs of points to globally describe the whole object from each surface point in a way that later the object can be locally matched with the scene. This four-dimensional feature, called Point Pair Feature or PPF, defines an asymmetric description between two oriented points by encoding their relative distance and normal information, as shown in the figure below.

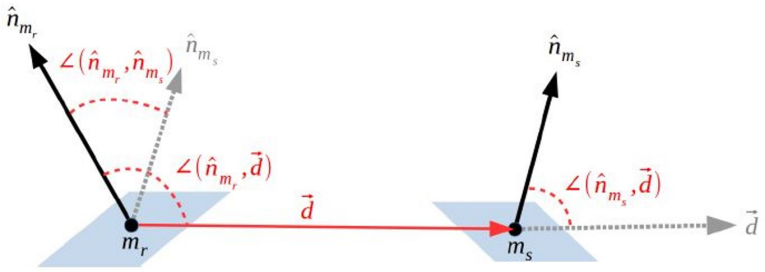

Figure 4.6: The Point Pair Feature definition for a model's point pair (mr, ms).

In detail, having a set of points in the 3D space $M \subset R^3$ representing the model object, for a given 3D point $m_r \in M$, called reference, and a given 3D point $m_s \in M$, named second, such that $m_r \neq m_s$, with their respective unit normal vectors $\hat{n}_{m_r}$ and n $\hat{n}_{m_s}$, a model four-dimensional feature $f^m \in (F^m \subset R^4)$is defined by the next Equation:

$$F_{rs}(m_r, m_s, \hat{n}_{mr}, \hat{n}_{ms}) = \tag{4.8}$$
$$\left[||\vec{d}||, \quad \angle(\hat{n}_{m_r}, \vec{d}, \quad \angle(\hat{n}_{m_s}, \vec{d}), \quad \angle(\hat{n}_{m_r}, \hat{n}_{m_s})\right]$$

where $\vec{d} = (m_{s_x} - m_{r_x}, m_{s_y} - m_{r_y}, m_{s_y} - m_{r_y})$ and $\angle(\vec{a}, \vec{b})$ is the angle between the vector $\vec{a}$ and $\vec{b}$. In the same way, having a set of point $S \subset R^3$ representing the scene data, the function F_{rs} can be applied to compute a scene PPF using a pair of scene points $s_r, s_s \in S$ such that $s_r \neq s_s$, with their respective unit normal vectors $\hat{n}_{s_r}$ and $\hat{n}_{s_s}$. Notice

that, if the object model has $|M|$ points, the total number of features is defined by $|F^m| = |M^2| - |M|$. With the aim to reduce the effect of this square relation on the method performance, the input data of both model and scene are downsampled with respect to the model size, effectively decreasing the complexity of the system. The method can be divided into two main stages: modeling and matching. On modeling, the global model descriptor is created by computing and saving all the possible model pairs with their related PPF. During the matching stage, the model pose in the scene is estimated by matching the scene pairs with the stored model pairs using the PPF. This matching process consists of two distinctive parts: find the correspondence between the pairs using the four-dimensional features and group the correspondences generating hypotheses poses.

The correspondence problem between similar point pairs is efficiently solved by grouping the pairs with the same quantized PPF on a hash table or, alternatively, a four-dimensional lookup table. Quantizing the feature space defines a mapping from each four-dimensional space element to the set of all point pairs that generate this specific feature. In particular, for the object model, this mapping from quantized features to sets of model pairs defines the object model description expressed by the function $L : Z^4 \rightarrow P(M_{pp})$, where $M_{pp} = \{(m_r, m_s) \,|\, m_r, m_s \in M, m_r \neq m_s)\}$ and $P(X)$ represents the power set of X. In other words, point pairs that generate the same quantized PPF are grouped together on the same table position pointed by their common quantized index, effectively grouping pairs with similar features. This process of model construction is done during the modeling stage, as shown in Figure for three sample point pairs.

Using this model description, given one scene pair, similar model pairs can be retrieved by accessing a table position pointed by the PPF quantized index. The quantization index is obtained by a quantization function $Q : Z^4 \rightarrow R^4$ using the step size Δ_{dist} for the first dimension and Δ_{angle} for the remaining three dimensions. The quantization step size will bound the similarity level, i.e., correspondence distance, between matching features, and hence point pairs. Defining a function $N : R^3 \rightarrow$

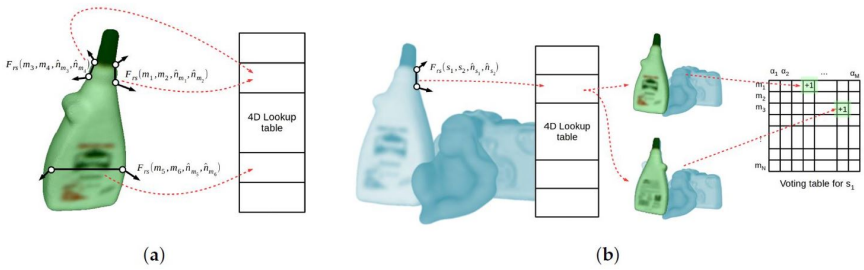

Figure 4.7: Representation of the modeling and matching steps of the Point Pair Features voting method.(a) modeling example for three point pairs from the model; (b) matching example for one point pair from the scene.

R^3 that computes a normal from a point, the correspondence matching subset of model pairs $S \subset M_{pp}$ for a given scene pair (s_r, s_s) and its related quantized feature $\vec{f^s} = Q(F_{rs}(s_r, s_s, \hat{n}_{s_r}, \hat{n}_{s_s})$ is defined by:

$$L(\vec{f^s}) = \{(m_r, m_s) \in M_{pp} \,|\, Q(F_{rs}(m_r, m_s, N(m_r), N(m_s))) = \vec{f^s}\} \quad (4.9)$$

From each scene-model point pair correspondence, a 6D pose transformation, or hypothesis, can be generated. Specifically, for a corresponding point pair $(m_r, m_s) \in A$, the matched reference points (s_r, m_r) and their normals $(\hat{n}_{s_r}, \hat{n}_{m_r})$ constrain five degrees of freedom, aligning both oriented points, and the second points (s_s, m_s), as long as they are non-collinear, constrain the remaining degree of freedom, which is a rotation around the aligned normals. However, the discriminative capability of a single four-dimensional feature from two sparse oriented points is clearly not enough to uniquely encode any surface characteristic, producing wrong correspondences. Therefore, the method requires a group of consistent correspondences to support the same hypothesis. Actually, more correspondences support a single pose, more likely this will be. In this regard, grouping consistent point pair correspondences, or, alternatively, 6D poses obtained from corresponding pairs have a high dimension complexity. In order to effectively solve this problem, a local coordinate, which we will refer to as LC, is used to efficiently group the poses within a two-dimensional space. As with two corresponding pairs, for a given scene point $s_i \in S$ that belongs to the object model, a 6D pose can be defined by only using one corresponding model point $m_j \in M$ and a rotation angle α around their two aligned normals, i.e., $\hat{n}_{s_i}$ and $\hat{n}_{m_j}$.

In this way, for the scene point s_i, a 6D pose transformation candidate $T_M^S \in SE(3)$ can defined by the LC represented by the parameters (m_j, α), as shown in the next Figure. To solve this transformation, both points and normals are aligned respectively with the origin and x-axis of a common world coordinate system $\{W\}$. Taking the scene point, this alignment can be expressed by the transformation $T_S^W = (R, t) \in SE(3)$. The rotation that aligns the normal vector $\hat{n}_{s_i}$ to the x-axis $\hat{e}_x$ is defined by the axis-angle representation $\theta\vec{v}$, where $\theta = \angle(\hat{n}_{s_i}, \hat{e}_x)$ and $\vec{v} = \frac{\hat{n}_{s_i} \times \hat{e}_x}{||\hat{n}_{s_i} \times \hat{e}_x||}$. Therefore, the rotation matrix $R \in SO(3)$ can be efficiently found using the Rodrigues' rotation formula [16]. In turn, the translation $t \in R^3$ is defined by $t = -Rs_i$. Exactly in the same way, the transformation $T_M^W \in SE(3)$ is found for the model point m_j and its normal $\hat{n}_{m_j}$. Using these two transformations and the rotation angle, the 6D pose for a given object instance is defined by:

$$T_M^S = (T_S^W)^{-1} R_\chi(\alpha) T_M^W \tag{4.10}$$

where $R_\chi(\beta) \in SO(3)$ represents a rotation of β angle around the x-axis. Using the LC, the correspondence grouping problem can be individually tackled for any scene pair created from s_i by grouping the corresponding model pairs in a two-dimensional space using the parameters (m_j, α).

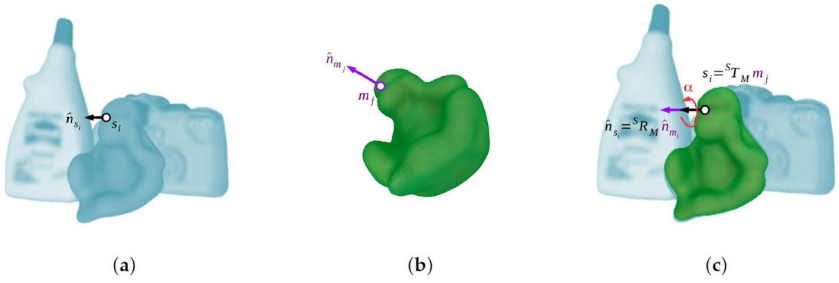

Figure 4.8: Representation of the local coordinate LC system used by the point pair features method; (a) scene oriented point; (b) corresponding object model oriented point; (c) alignment of the model with the scene by using the two oriented points and the α angle.

During grouping and hypothesis generation, for every reference scene point si, the method intends to find the LC, i.e., (m_j, α), which defines

the best fitting model pose on the scene data or, in other words, that maximizes the number of pairs correspondences that support it. This correspondence grouping problem is solved by defining a two-dimensional voting table or accumulator, in a Generalized Hough Transform manner, representing the parameter space of the LC, where one dimension represents the corresponding model point m_j and the other the quantized rotation angle α. In particular, for each possible scene pair generated from s_i, i.e., $(s_r, s_s) \in \{(s_k, s_l) \,|\, s_k, s_l \in S, s_k \neq s_l, s_k = s_i\}$, a LC will be defined by a corresponding pair (m_r, m_s) reference point, i.e., $(m_j = m_r)$, and the rotation angle α defined by the two second points (s_s, m_s). The corresponding model pairs are retrieved from the lookup table using the quantized PPF and, for each obtained LC, a vote is cast on the table, as represented by Figure 4.7b for a single pair. After all pairs are checked, the peak of the table represents the most supported LC, and hence the most likely pose, for this specific s_i point. This process is applied to all or,alternatively, a fraction of the scene points, obtaining a set of plausible hypotheses.

To increase the efficiency of the voting part, which requires to compute the α angle for each pair correspondence, it is possible to split the rotation angle α in two parts; one part related to the model point, α_m, and one part related to the scene point, α_s. In detail, taking into account that in the intermediate world coordinate system the α angle is defined around the x-axis, the rotation on the two-dimensional yz-plane can be divided with respect to the positive y-axis. In this case, the α_m and α_s will be defined as the rotation angles between the positive y-axis vector $\hat{e}_y$ and the yz-plane projection of the vectors obtained by the world transformed second points of the model pair $T_M^W m_s$ and scene pairs $T_S^W s_s$.

As shown in Figure 4.9, these angles can be defined as $\alpha_s = atan2(\frac{a_z}{a_y})$ and $\alpha_m = atan2(\frac{b_z}{b_y})$, where $a = T_S^W s_s$, $b = T_M^W m_s$ and $atan2(\beta, \gamma)$ represents the multi-valued inverse tangent. With this solution, the model angle can be pre-computed during the modeling stage and saved alongside the reference point in the lookup table (m_r, α_m) (mr, am). Later, during the matching stage, for each scene pair, the α angle is computed

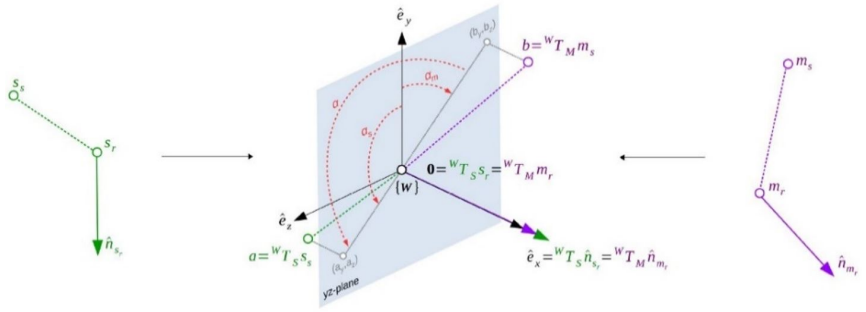

Figure 4.9: Representation of the LC α angle definition from two corresponding pairs (s_r, s_s) and (m_r, m_s).

by adding the two angles. Considering that is defined from the model to the scene, the total angle can be computed as $\alpha = \alpha_s - \alpha_m$. Finally, in order to join similar candidate poses generated from different scene reference points, the method is completed with a clustering approach that groups similar poses that do not vary in rotation and translation more than a threshold.Understood the mechanism on how PPF surface matching works, is important to re-focus the attention on this part of the algorithm. The CAD model of the piece that must be recognized is exported in *.ply* format with all the mesh surface normals.

The point cloud is loaded into the PPF Training part in which all the point pair features are calculated. Remembering that the number of calculations increase quadratically with the number of points, a downsample can be exploited to reduce the training time, taking into consideration that precision will diminish. After that, the cut scene point cloud coming from the hidden point removal node to simulate the sensor vision is loaded, and the surface normals are calculated. In this case, the mesh normals are not calculated directly by the CAD software, because in the real application, the sensor acquires a point cloud and not a mesh. So, the scene normals computation must be calibrated very well to match piece normals. With this aim, two are the main parameters for scene normals calculation:

- *Sphere radius*: for every point in the point cloud, a sphere will be generated and only the point inside will be considered for the normals calculation referred to that specific point.

- *Maximum number of neighbors*: inside the sphere created, a huge amount of points can be present. This parameter allows to select a specific number of points to average the results and create a plane for which normal can be computed.

Once defined all scene point cloud normals, surface matching can be exploited. The matching process terminates with the attainment of the pose. However, due to the multiple matching points, erroneous hypothesis, pose averaging and other factors, such pose is very open to noise and many times is far from being perfect. Although the visual results obtained in that stage are pleasing, the quantitative evaluation shows 10 degrees variation, which is an acceptable level of matching. Many times, the requirement might be set well beyond this margin and it is desired to refine the computed pose. Furthermore, in typical RGBD scenes and point clouds, 3D structure can capture only less than half of the model due to the visibility in the scene as in our case. Therefore, Iterative Closest Point algorithm has been inserted to obtain a robust pose refinement.

At the end of this last part of the offline algorithm, the interest is not on the pose itself, but on the matching score obtained from the matching. In PPF, the pose score is returned as the number of different matching that supports a single pose inside the hash table. Indeed, more the pose is supported by other PPF, more will be the score assigned to the specific homogeneous matrix obtained. On the other hand, a problem arises because no maximum score value can be defined. This generates a small uncertainty on the fact that the matched pose will be effectively right. To avoid the problem, some tests can be done also controlling if the pose is correct and doing a cycle of random generated sphere points to find the maximum value. After that, pose score can be normalized in a number between 0 and 1. As concerns HALCON, the same process is exploited, with the difference that no training stage and no specific attention on surface normals calculation is required. Also, the problem of the matching score does not exist since the output score is already normalized between 0 and 1 and it is not based on the number of votes sustaining a specific pose.

x	y	z
float64	float64	float64
0.5000	0.2000	0.2000

qx	qy	qz	qw	reach	score
float64	float64	float64	float64	float64	float64
0.8212	0.0000	0.5023	0.2851	1.0000	0.5702

Table 4.3: Example of ROS message sent from Surface Matcher Node: Sensor Position, Quaternion Orientation, Reachability and Matching Score Index

Finally, for both PPF and HALCON a response to Bayesian Optimization can be given. The customized ROS message will send reachability and score parameter to the first node in order to calculate the cost function and generate a new optimized point to be observed. After that, all the cycle resume until Bayesian Optimization node reach a minimum value of the cost function. A *.txt* file is generated with all the sensor position and orientation in terms of quaternions also saving the reachability index and the pose score.

x	y	z
0.5000	0.2000	0.2000
0.6123	0.2856	0.3421
0.4523	0.3425	0.1564
0.3654	0.9231	0.5639

qx	qy	qz	qw	reach	score
0.8212	0.0000	0.5023	0.2851	1.0000	0.5702
0.1567	0.7224	0.2853	0.6354	1.0000	0.8765
0.7223	0.1567	0.5843	0.3524	-1.0000	0.9234
0.0000	0.0000	0.0000	1.0000	-1.0000	0.3476

Table 4.4: Part of File Generated by Offline Algorithm

Chapter 5

Planner Online

5.1 General Overview

After that the Bayesian Optimization has found all the positions and orientations guaranteeing the best score matching, the quality control can be performed. The online algorithm is composed by five different sub-programs, each one with a specific task, that, as before, communicates each other through ROS messages.

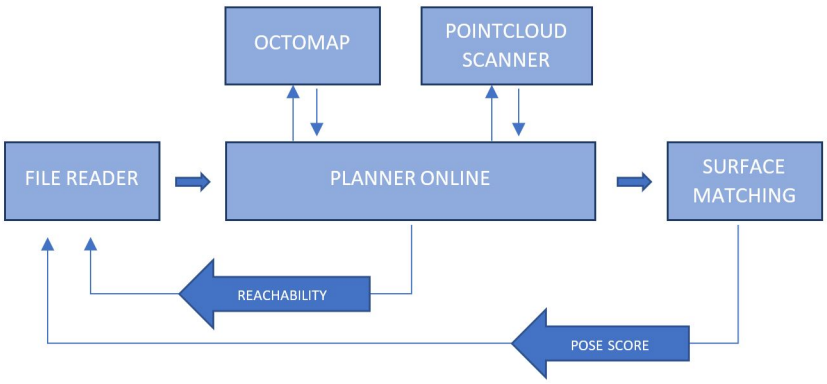

Figure 5.1: Schematic Representation of the Planner Online Algorithm

All the algorithm starts with a node which aim is to read the file generated by the offline code. In this file sensor's three positions and the orientation quaternion are registered. For each of these couples, score and reachability are also assigned. Poses are sort for highest score and

sent to planner node. In real environment, the position with highest score, maybe, cannot be reached due to obstacles that was not considered and, if executed, can lay to collisions. To face this kind of problematic, *Octomap*© library has been used. It is aimed to create occupied volume space inside the real environment reconstruction starting from sensor point cloud data. Therefore, collision can be avoided because robot planner recognizes as obstacles the occupied volume spaces and consequently plan a trajectory not in collision with them. Once the robot is in position, a point cloud scan is acquired and then surface matching is computed. Based on matching score, the algorithm decides if a further scan is necessary to be sure of the object pose. If score is high, the homogeneous matrix between robot reference system and the target object is saved in a file which will be compared to the desired pose.

5.2 End Effector - Sensor Transform Calibration

As explained in the previous chapter, knowing the real transform from robot's end-effector to sensor is crucial to obtain valuable results. The pure geometry transform is not enough, indeed, from firsts experimentations, errors in order of few centimeters were achieved. Obviously, this scale of variations is not acceptable. To overcome the problematic, an algorithm able to calculate the transform was used. The code can be divided into three main nodes. The first is aimed to place the robot in different positions and extract, from robot server, the transform between the null reference frame and robot end effector $[T_{o-ee}]$, to publish it on a specific topic. The second one extrapolates the transform from a QR Code framed by the camera and the camera itself $[T_{sens-QR}]$, with the aim to publish it in a different topic. The third one reads the two transforms and execute calculations in order to achieve the homogeneous matrix from end effector to the sensor's point cloud origin $[T_{ee-sens}]$.

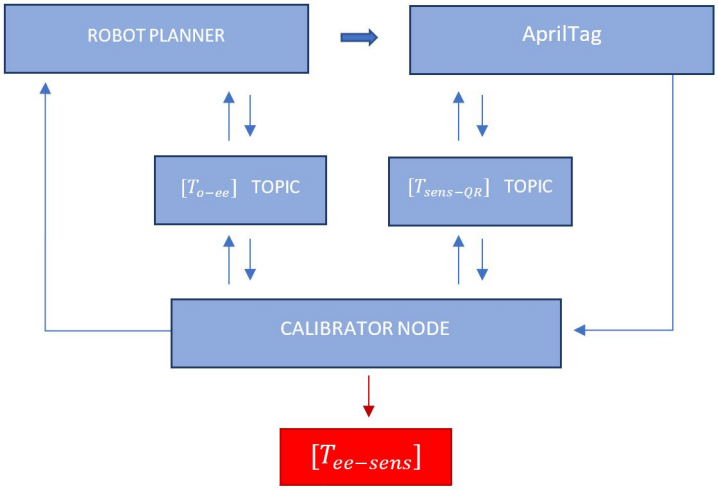

Figure 5.2: Schematic Representation of the Calibrator Algorithm

Let's now analyze in details the three different nodes. Initially, a robot position and orientation are generated and executed, so that we have robot in a specific position. The robot's transform $[T_{o-ee}]$ is extrapolated from the server and published on the related topic. At this

point, AprilTag node starts. AprilTag code, uses an embedded 2D-coded marker for QR tag detection. AprilTag is one of the most used fiducial markers that can be used both indoors and outdoors for ground truth generation in 6 DOF.

The proposed research has established that both distance and orientation of viewing camera from the target tag effects accuracy. However, uncorrected orientation uncertainty is a more significant source of accuracy degradation. AprilTag's accuracy is maximum when the viewing camera is pointed towards the center of the tag. The visual QR marker tag can be of any size with a square dimension. The tag is printed on a white background with a black outline square. Inside the square is an embedded black bar-code. AprilTag uses a unique detection algorithm for fast, robust detection and to minimize the effect of small occlusions. Figure 5.4 shows the algorithmic passages of AprilTag. In the first step, it computes the magnitude and direction of a gradient at every pixel in an image that contains the AprilTag. Afterward, these calculated gradients are grouped into clusters called components based on similar gradient attributes using a graph-based method. By using a weighted least square technique, a line is fitted on every component such that the direction of the gradients determines the direction of the fitted line.

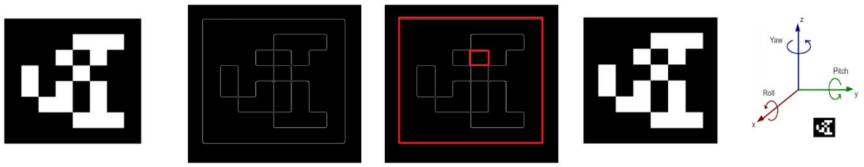

Figure 5.3: From Left to Right: AprilTag Input image - Step 1: Detection of line segments using the least square method on clusters of similar pixel gradients. - Step 2: Based upon the gradient direction, all possible quads are detected in an image. - Step 3: A quad with a valid code scheme is extracted to detect the pose. - Step 4: A pose of AprilTag in camera frame of reference is returned using homograph and intrinsic estimation.

Moreover, gradient direction determines the direction of the line segments. Hence each line has a dark side on its left and a lighter side on its right. Furthermore, after identifying all lines, possible quad shapes are detected. The quad shape with a valid code scheme is extracted out.

Also, a 6 DOF pose of the tag in the camera frame of reference is returned by using homography and intrinsic estimation over an extracted tag.

Once that also the $[T_{sens-QR}]$ has been published on the relative topic, the process of generating a new coordinate for robot, executing and detecting AprilTag is repeated for n number of times. At this point on the two different topics will be present$n$ $[T_{o-ee}]$ and $[T_{sens-QR}]$ transforms.

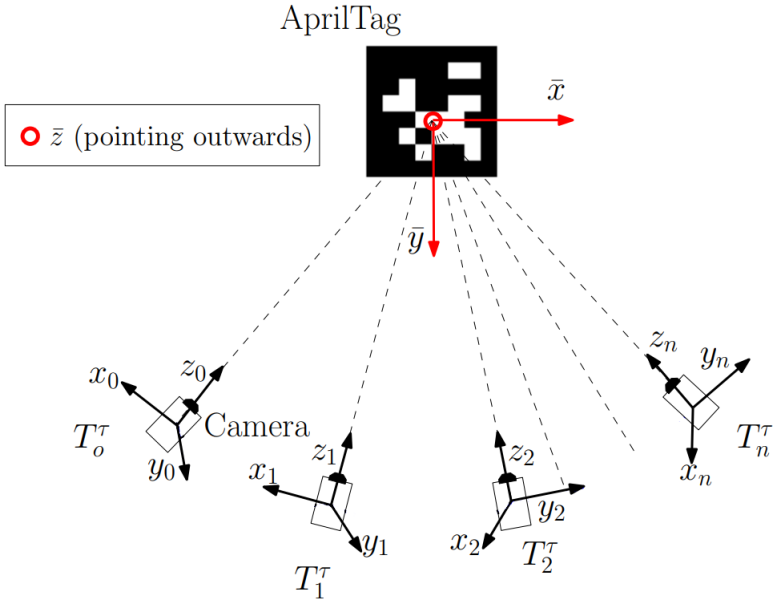

Figure 5.4: Representation of Robot's different Position and of AprilTag's Reference System

Calibrator node reads the $2n$ transforms from the topics and starts calculating $[T_{ee-sens}]$. The process is based on the equality of QR code pose respect to null reference frame. Indeed, also if the position of the robot changes, the post-multiplication sequence of transforms must be equal, because QR code has not been moved. To understand better this concept, assign to number 1 the first pose reached by robot, with 2 the second and so on. For the first couple $1-2$, we can write:

$$\begin{aligned} [T^1_{o-ee}]\,[T^1_{ee-sens}]\,[T^1_{sens-QR}] &= [T^1_{o-QR}] \\ [T^2_{o-ee}]\,[T^2_{ee-sens}]\,[T^2_{sens-QR}] &= [T^2_{o-QR}] \end{aligned} \tag{5.1}$$

Remembering that the pose of the QR does not change during all calibration process, the following equation is accomplished:

$$\begin{gathered}\left[T^1_{o-ee}\right]\left[T^1_{ee-sens}\right]\left[T^1_{sens-QR}\right] = [T^2_{o-ee}][T^2_{ee-sens}][T^2_{sens-QR}] \\ \left[T^2_{o-ee}\right]^{-1}\left[T^1_{o-ee}\right]\left[T^1_{ee-sens}\right] = [T^2_{ee-sens}][T^2_{sens-QR}][T^1_{sens-QR}]^{-1} \\ [A]\left[T^1_{ee-sens}\right] = [T^2_{ee-sens}][B] \end{gathered} \tag{5.2}$$

The solution of equations as $[A]\,[X] = [X][B]$ can be found in [17] and has been discussed in linear algebra. Theoretically, only a pair of points is necessary to extract the value of $[T_{ee-sens}]$ but noise meddle. Noise can vary in dependence of many factors as lightning conditions, centering of the camera respect to QR, reflectivity of the QR code printing and so on.

To overcome this kind of problematic, for every pair of points reached by robot, $[T_{ee-sens}]$ will be calculated. At the end, a square minimization error procedure will be performed to extract the best candidate transform. Accomplished this passage, another aspect must be considered. The $[T_{ee-sens}]$ calculated, is referred to the RGB camera inside the sensor and not on the RGBD one, because the QR recognition is achieved through the RGB image. So, a further known sensor's factory transform needs to be inserted inside the process.

$$\begin{gathered}[T_{ee-rgb}] = [T_{ee-rgbd}][T_{rgbd-rgb}] \\ \left[T_{ee-sens}\right] = [T_{ee-rgbd}] = [T_{ee-sens}][T_{rgbd-rgb}]^{-1}\end{gathered} \tag{5.3}$$

At the end of calibration procedure, the real position and quaternion orientation linking end effector to sensor that has been found is reported:

$$\vec{t} = \begin{bmatrix} -0.0210606 \\ 0.0004631 \\ 0.0498742 \end{bmatrix} \tag{5.4}$$

$$\vec{q} = \begin{bmatrix} 0.0066612 & -0.0114342 & 0.0115436 & 0.999845 \end{bmatrix}$$

5.3 Point Cloud Scene Reconstruction

Not always a CAD model of the scene can be available. For this purpose, an algorithm able to reconstruct the scene starting from single sensor scans in different positions has been implemented. The idea is to place the sensor attached to the robot's end effector in different locations and acquire different point cloud scans. After that, the scene is reconstructed by referencing all the points to robot reference frame multiplying them for the homogeneous matrix from the reference frame to the camera. Let us define $\vec{p}_{sens}$ as the homogeneous vector of a generic point acquired by the sensor in sensor reference system. Identically, $\vec{p}_o$ will be the same point but in robot reference frame. The reference frame transformation is exploited with $[T_{o-sens}]$ matrix through the following equation:

$$\vec{p}_o = [T_{o-sens}]\vec{p}_{sens} \tag{5.5}$$

In the next image an example of three different scans that are merged is reported.

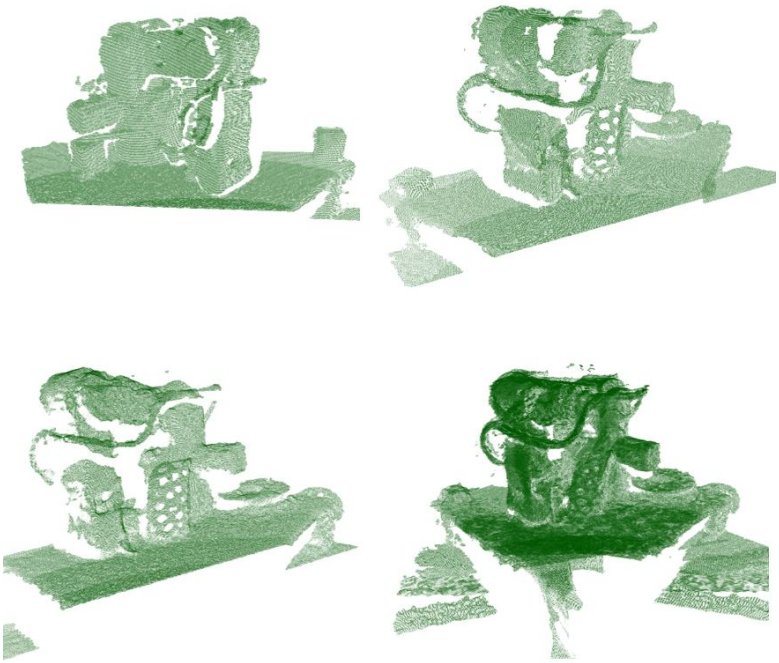

Figure 5.5: Three different Point Cloud Scans of the Scene and its Reconstruction

The positions in which camera will move can be defined in two different ways: manually or through a pre-defined path. In the first case, the robot is moved with the help of an operator which decides the best poses and orientation with the aim to reconstruct the scene. On the other hand, the automatic algorithm has the input necessity of a rough object position, around which a spherical path will be generated. For every position in both manual and automatic case, a sensor point cloud scan is saved with the coordinates and orientation of the camera. As concerns the manual casuistry, the pose and orientation of the camera is asked to the robot server to obtain the most accurate numbers.

In automatic modality instead, the position and orientation are calculated by the computer and then sent to the robot controller. Unfortunately, the coordinates that are established not always are executed with high accuracy, some tenths of millimeter can vary. This variation causes amplification of noise in the scene reconstruction, meaning to a not completely true representation of the reality. For avoiding the problem, also

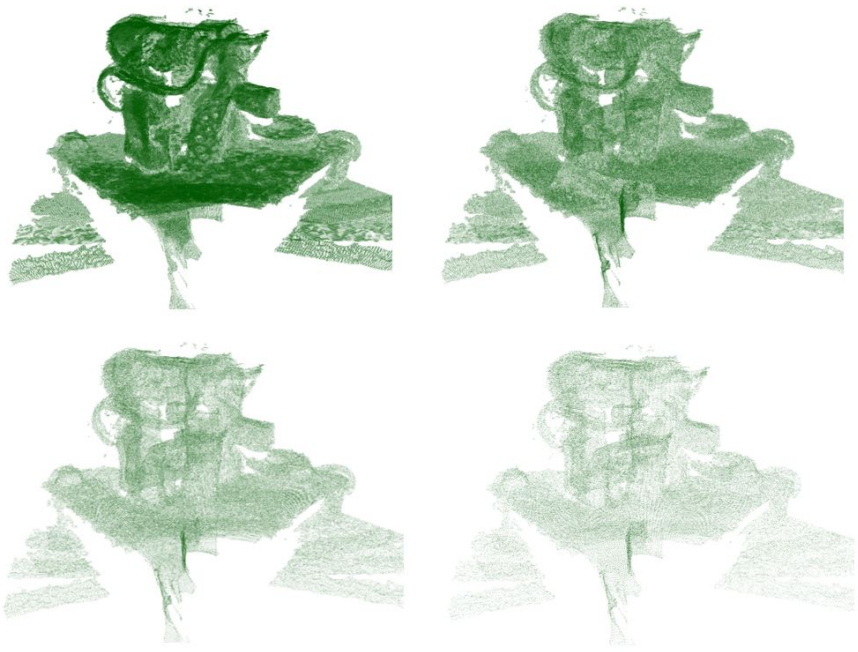

Figure 5.6: Different examples of *voxel_downsampling_size*. Top Left: Original Reconstructed Scene. Top Right: Reconstructed Scene with *voxel_size=1 mm*. Bottom Left: Reconstructed Scene with *voxel_size=2 mm*. Bottom Right: Reconstructed Scene with *voxel_size=3 mm*

in this case, the exact position and orientation are asked to the robot server to obtain more precise numbers.

Once that all scans points have been referenced to robot reference frame, the problematic of multiple points describing the same target object parts must be faced. At this purpose, a special function able to concentrate all very close points into a single one has been used. The only input to this function is called $voxel\_size$ and represents the side dimension of a cube. For all the points in the point cloud, a voxel size cube is created and all the point inside are merged into a single one averaging by distance. In Figure 5.6, is possible to notice that lower the $voxel\_size$ parameter lower will be the density of points and more precise respect to reality will be the processed point cloud.

5.4 Online Algorithm

The first node of the online algorithm has the aim to read the file generated by the previous part, the offline algorithm. This .txt file contains all the sensor position that was sampled by Bayesian Optimization. For each of this position are associated orientation, reachability and matching score indexes. In the next table a piece of the file is presented to allow a better comprehension.

x	y	z
0.5000	0.2000	0.2000
0.6123	0.2856	0.3421
0.4523	0.3425	0.1564
0.3654	0.9231	0.5639

qx	qy	qz	qw	reach	score
0.8212	0.0000	0.5023	0.2851	1.0000	0.5702
0.1567	0.7224	0.2853	0.6354	1.0000	0.8765
0.7223	0.1567	0.5843	0.3524	-1.0000	0.9234
0.0000	0.0000	0.0000	1.0000	-1.0000	0.3476

Table 5.1: Components of Output File exiting from Offline Algorithm: Position, Quaternion Orientation, Reachability Index, Pose Score Index

Once the file has been loaded into the program as a matrix, it is sorted by highest matching score, which characterizes the best pose to observe the piece. Also, a control on the reachability index is performed. As instance, if the position with the highest score is not approachable by the robot due to joints limits, the second one will be chosen and so on. Determined the robot position based on the parameters just explained, pose and orientation are packed into a customized ROS message and sent to the second part of code. Planner online has the aim to verify if pose and orientation received from reader node are achievable in robot joints space. Before explaining in detail this section, as introduced before, a specific set of cartesian coordinates cannot be reached by the

robot, maybe due to obstacles that were not considered in the offline part. Aiming to resolve this kind of issue, Octomap library has been used.

Before planning robot trajectory based on reader information, the sensor is started. The Octomap node capture, for every frame, the point cloud generated by the sensor and publish the points on a specific ROS Topic. A standard HD resolution of 1280×720 pixels generates near a million points. Higher the resolution, higher will be the point cloud density and precision, but also higher will be the computational power needed to handle this kind of data.

Once published the points in the ROS Topic, Octomap library perform a downsample, and for every point creates a cube of occupied volume space of pre-defined size. In that cube and near it, Move-It cannot plan a robot trajectory, avoiding possible robot collisions. Octomap library can be customized with different parameters as input:

- *Sampling number*: This parameter determines the sampling number. As instance, if it is set to 5, a fifth of the points will be used.
- *Refresh rate*: indicates at which frequency the Octomap is updated.
- *Occupied Volume Size*: represents the width size of the cube of occupied volume space generates around the specific point.
- *Maximum sample depth*: is the maximum depth for which a point is considered. If a specific point exceeds this value in depth coordinate, will be discarded in Octomap creation.

During running of Octomap node, robot is moved through joints trajectory previously decided and surely collision-free because controlled by the operator. This movement is aimed to create a sort of map of the real volume occupied before executing the quality control preventing any sort of collision. Finished the Octomap creation, sensor is stopped.

After Octomap has been created, planner online node assign to the robot planner the coordinates and orientation coming from the reader. If

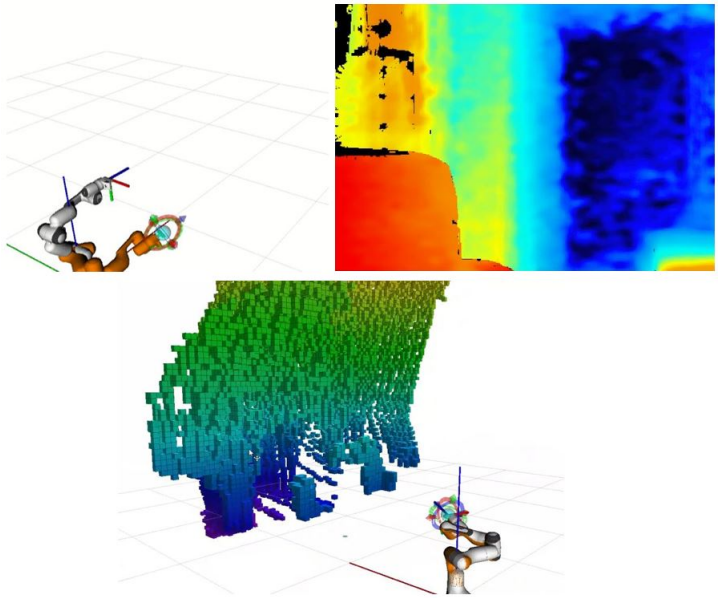

Figure 5.7: Sequential images: before creating Octomap, Sensor Depth View and after Octomap Creation.

the set of numbers can be planned also considering Octomap, end effector positions and orientation are calculated thanks to Equation 4.6 and trajectory will be executed by the robot. Once reached the placement, a ROS message is sent to point cloud Scanner node to start the sensor. The first frames are skipped due to light exposure adjustments, the point cloud scan is acquired and saved in a *.ply* file. Sensor is stopped and the file is read by the matching node. Surface Matching between the CAD model and the real point cloud scanned by the senor is performed, using both the algorithms of PPF and HALCON. The homogeneous matrix between robot reference frame and the object is saved and compared to the desired one to understand if the object is correctly mounted.

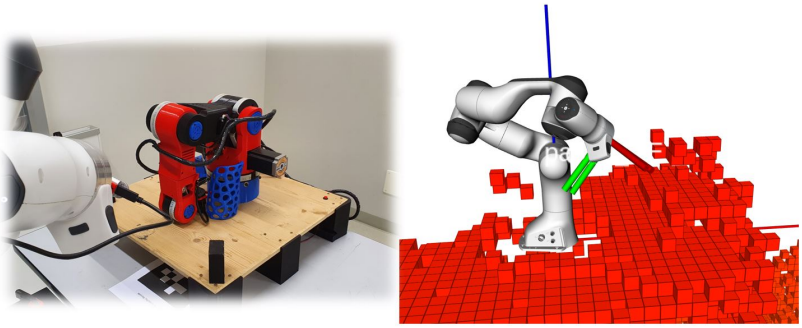

Figure 5.8: Octomap of the Reconstructed Scene

Chapter 6

Experimentation Results

The proposed Offline algorithm, described in Chapter 4, and the Online one, referenced in Chapter 5, have been validated in order to show their performances, strengths and weaknesses on real applications. In particular, this part of the overall work will presents:

- A description of the Graphical User Interface created for simplifying code usage
- A comparison between the two presented matching code *HALCON* and *PPF* and the one relatively chosen
- A comparison between the Grid Sampling and Bayesian Optimization in Offline mode
- A method for validating Offline algorithm respect to reality coming from sensor located in the best view positions
- A sensitivity analysis to achieve a calibration on the main Bayesian Optimization parameters with different Cost Functions
- An application on real scene with high occlusion rate
- The results of the developed algorithm on a reconstructed scene

6.1 Graphical User Interface

For easier parameters setting during experiments and comparisons, a graphic interface has been coded. Through the GUI, is possible, in a fast and visible manner, to understand if the points of the sphere are effectively reachable by robot and whether such poses are useful.

Once all the parameters are effectively settled, the configuration is saved in a *.txt* file, which will be read by the nodes of the developed algorithms. Another functionality allows to change directly the wanted parameters from file and then update all the variables in the GUI for a visual check.

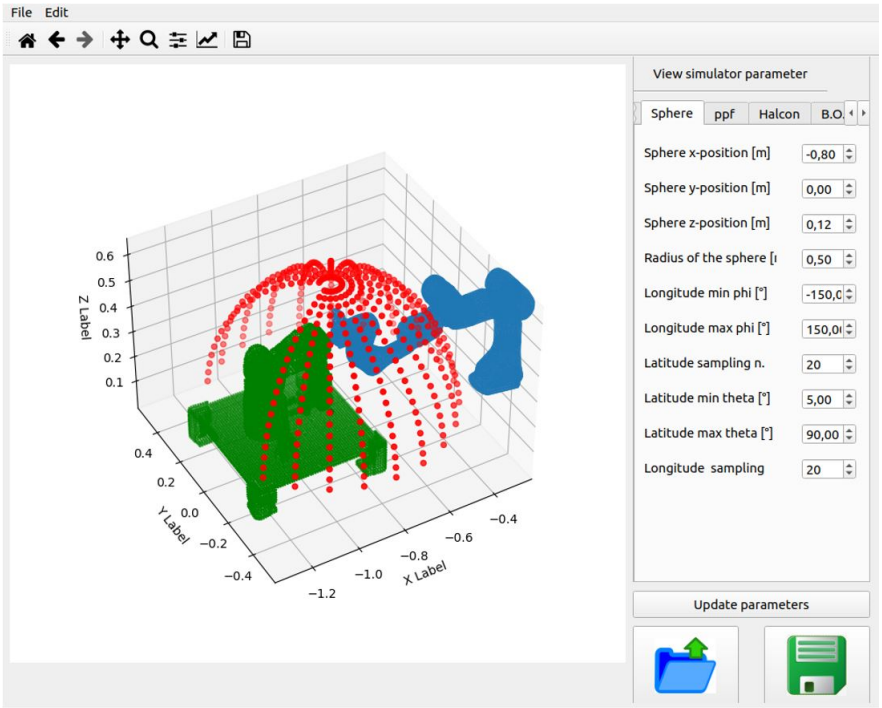

Figure 6.1: Graphic User Interface Coded. In blue is possible to see the robot representation, in green the scene uploaded and in red the points generated on the surface of the sphere

In the initialization of the graphic interface, it is possible to insert the point cloud of the scene and the robot model, respectively in green and blue in Figure 6.1.

The View simulator parameter sections allows to set:

- The positions and orientation of the scene respect to the robot null reference frame
- Sphere features as center, radius, minimum and maximum azimuth and polar angles
- The matching Parameters of both *PPF* and *HALCON* Algorithms
- The Bayesian Optimization Cost Functions Parameters

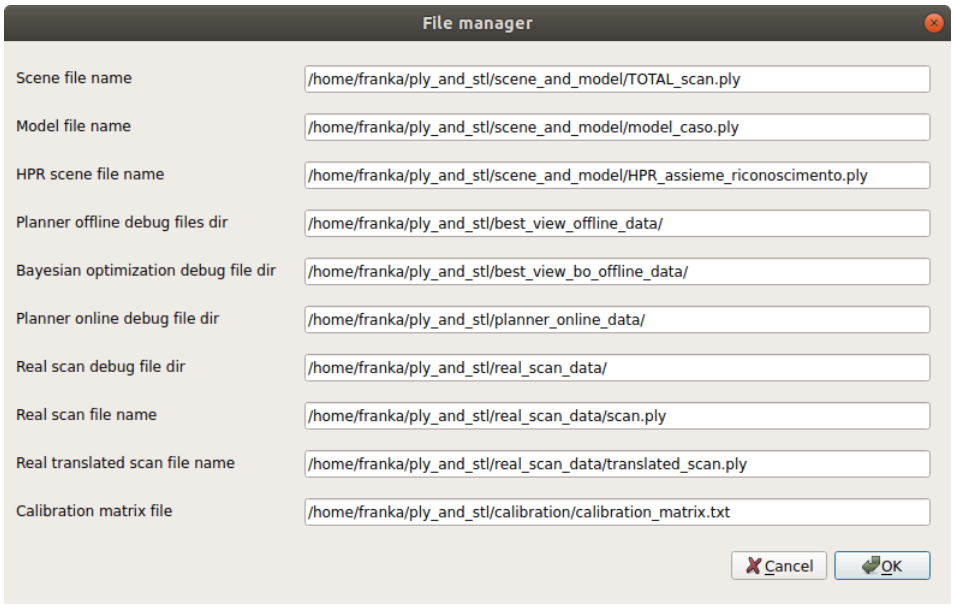

Figure 6.2: Graphic User Interface: File Manager

The File Manager section instead is the one that allows to understand which files are used by the presented algorithm. Every node in the algorithm that has been coded, starts with reading the file generated by the GUI and extracting the necessary parameters of the specific program. In this way, the user is helped in all the modifications of the parameters values because is all condensed in a simple file instead of opening different scripts and modify the specific lines of code.

6.2 Model and Scene

It has been decided to test the algorithms using, as scene, a 3D printed robot that we built in a previous university course. This scene had been chosen because presents an enhanced complexity with the possibility to occlude objects in function of the given viewpoint.

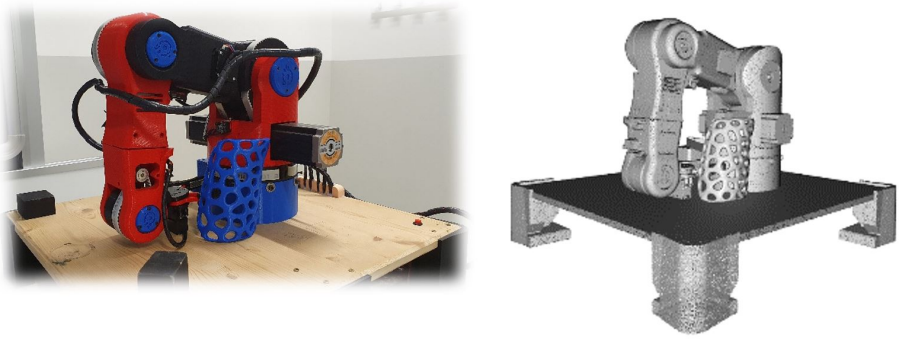

Figure 6.3: Virtual and Physical Models of the chosen Scene

As model to be recognized, it has been chosen to use a 3D printed prosthesis prototype. This model presents curved surfaces and holes, which enhance the complexity recognition of the matching algorithm.

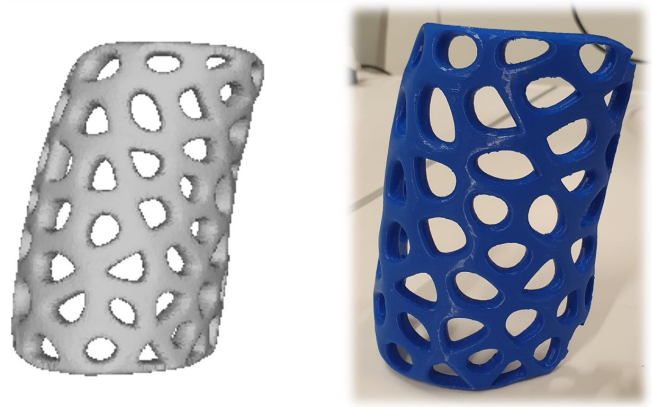

Figure 6.4: Virtual and Physical Models of the chosen Object to be Recognized

It has been decided to introduce the model inside the scene in a specific position, making it not visible from all the possible viewpoints. This choice is useful to test if the BO algorithm is able to to find a position in which the object can be seen.

6.3 *PPF* and *HALCON* Pose Estimation Algorithms Comparison

As presented in the previous chapters, two algorithms for Pose Estimation have been tested. The first, *PPF*, is an open source code based on Point Pair Feature, while the second, *HALCON*, is a proprietary software for which an academic licence was asked. For choosing which one can be considered the best for this application, some analysis have been performed.

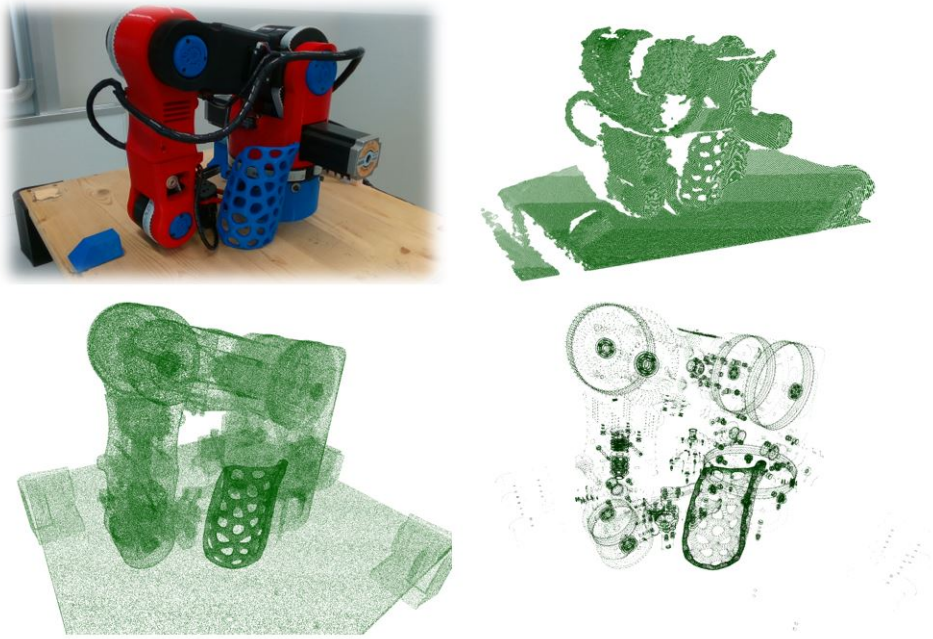

Figure 6.5: RGB Image *(Top Left)*, Sensor's Point Cloud *(Top Right)*, Uniform Sampled CAD Scene *(Bottom Left)* and *.stl* CAD Scene *(Bottom Right)*

As can be exploited by the figure, some problematic in scene simulation need to be faced. The use of scenes generated by *.stl* CAD file characterized by a not uniform point distribution can lead to the rising of errors in matching phase. This aspect is due to the fact that this file extension is focused on surfaces geometry reconstruction, consequently all the sampled points are associated to a specific face of the object. Higher the complexity of the geometry, higher will be the number of points used to describe it. As instance, a curve will require a number of

sampled points drastically increased respect to a plane. Obviously, the real point cloud acquired by the sensor is not comparable with this type of extension since HPR operator in simulation will delete a considerable amount of points depending on the sensor's location. In order to simulate in a correct manner the point cloud generated by the sensor, a uniform sampling of the CAD geometry has been necessary. To enhance the understanding of this concept, further experiments with the two different matching codes have been performed.

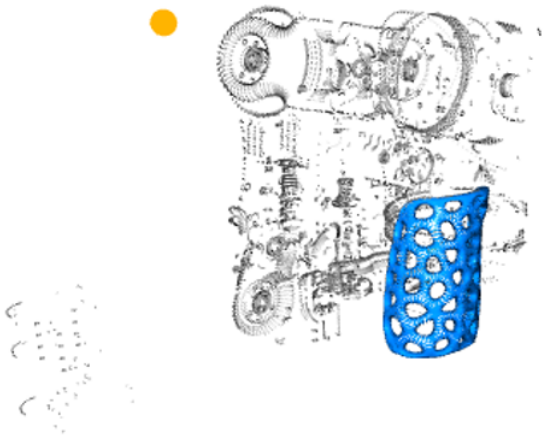

Figure 6.6: HPR Error due to Lack of Surface Points in *.stl* CAD Scene

In Figure 6.6 the visualization of an incorrect HPR scene elaboration is reported. The lack of points in the *.stl* case leads to a fail in the Hidden Point Removal process, which is no longer able to represent the cut point cloud from the viewpoint in a proper manner. This is probably due to HPR process that leads to remove hidden points in function of the scene point density. High variations in density may lead to errors in the actual estimation of the visible points. Indeed, as emphasized by the figure, the recognition of the model happens even if it is not actually visible from the analyzed pose.

In order to evaluate the differences between the *.stl* and uniform sampled scene, a grid with 400 sampled points for the evaluation of $3D$ Surface Score Distribution has been performed with *HALCON* and *PPF* matching methods. In terms of understanding the settings, both are valid choices, while comparing the easiness in finding the specific values of

parameters to be assigned to a peculiar scene and model, the proprietary one overcomes the other. In addition, the biggest limit in the open source algorithm is its inability in returning an absolute and normalized value of the matching score, representing the quality and safety of the performed matching. Indeed, the only value that can indicate how good is a pose is the number of votes for possible poses inside the hash table (Figure 4.7).

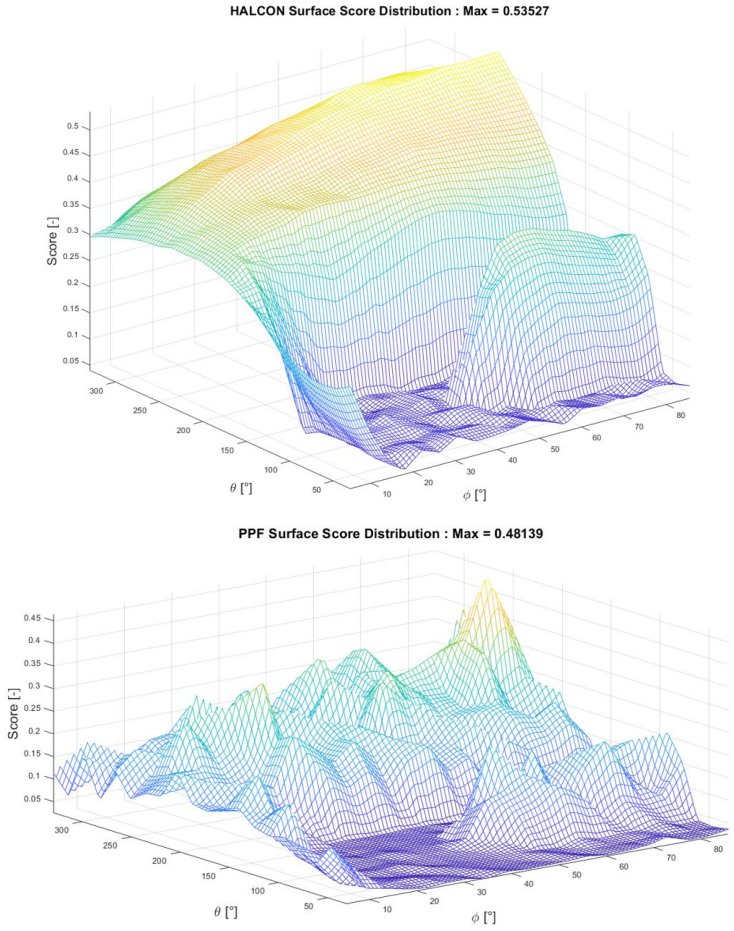

Figure 6.7: 3D Surface Score Distribution of *HALCON (Top)* and PPF *(Bottom)* Matching Algorithms with a *.stl* Scene

It can be noticed that the score evaluated by the two software shows significant variations using a *.stl* scene. Both *HALCON* and *PPF* are able to correct estimate the pose of the model inside the scene. The former presents a smooth tendency in score distribution. The latter evidences a high function gradient variation for the input variables ϕ and θ underlined by multiple peaks.

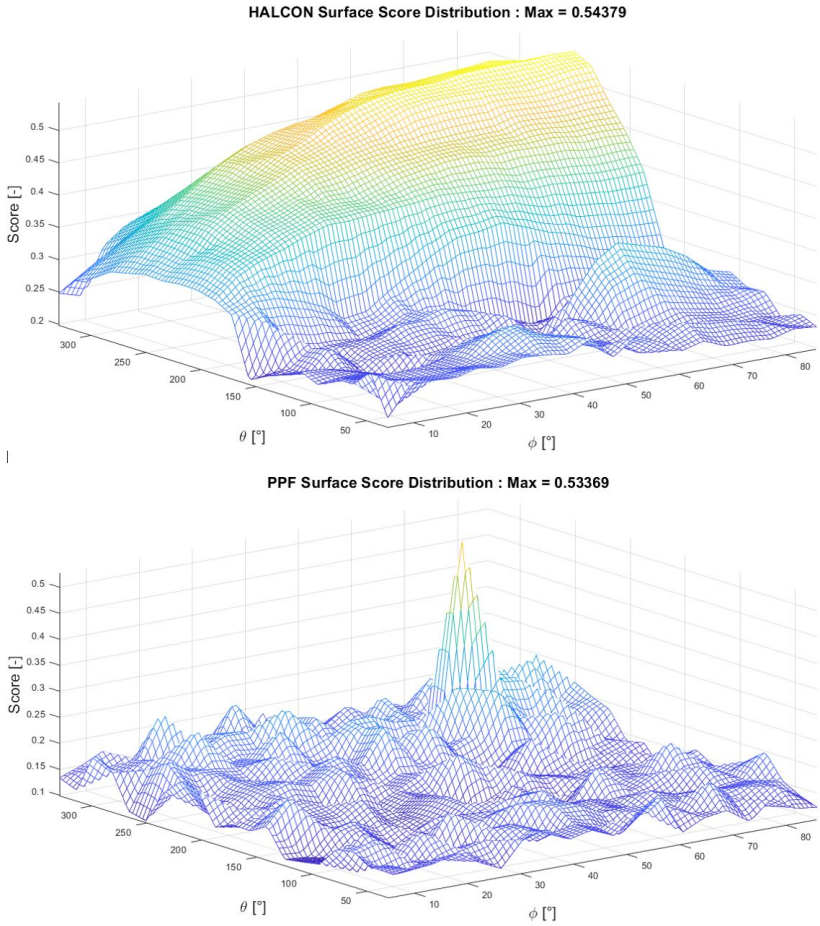

Figure 6.8: 3D Surface Score Distribution of *HALCON (Top)* and PPF *(Bottom)* Matching Algorithms with Uniform Point Sampled Scene

As before, also for the uniform sampled scene, the same trend for both algorithms is shown. In this case the peaks problematic is emphasized due to difficulties of the open source code to find the model in the scene. This causes the generation of a smaller area evidenced in Figure 6.8 in which score can be considered acceptable. Score function smoothness, proper stable matching and correct pose estimation are starting points for the definition of a reliable optimization.

The last factor to be considered is the time consume in finding model pose. As example, with the model and scene described in Figures 6.3 and 6.4, the open source code employs considerable higher time for every matching procedure. For the motivations just expressed, all the following tests were performed with *HALCON* algorithm.

6.4 Bayesian Optimization and Sphere Grid Algorithms Comparison

To test performances, a ROS node that creates a uniform grid of viewpoints on sphere surface has been coded. Sphere Grid has the aim to compare in terms of time and performances in matching score, how BO behaves in comparison to a simple sampling approach. In this case, for each viewpoint, the matching procedure is performed, regardless if the matching provides a correct pose of the object or not. On the other side, BO consider if the pose score is low and avoid that area.

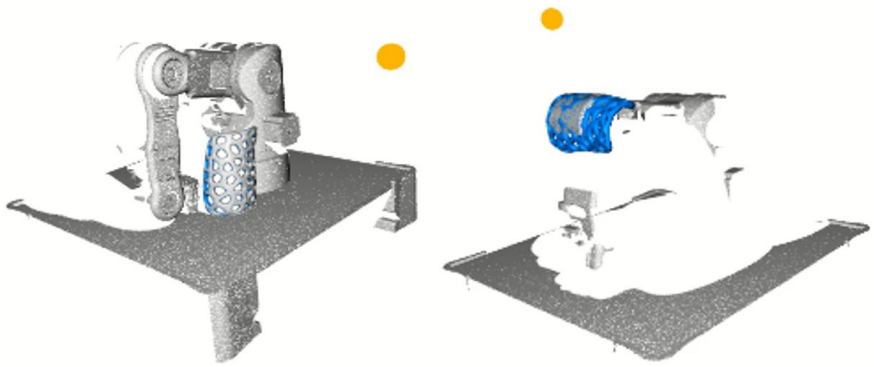

Figure 6.9: Example of Perfect *(left)* and Missed *(right)* Matching between the Model and the Scene. The only parameter that vary is the viewpoint enhanced in yellow, in first case the object is visible, in second one not, leading to a wrong match.

A simulation with 400 points was performed in order to have a good point density useful for comparison. Remembering that ϕ represents the polar angle and θ the azimuth angle as were explained in Figure 4.3, given a fixed radius ρ in accomplish to joint limits of the robot, the main parameters set are reported in Table 6.1. In order to improve performances and smartness this work implies the use of Bayesian Optimization. As discussed before, BO can be seen as black box, which makes forecast and try to optimize an unknown function. The behaviour of the algorithm depends on the cost function, which is a formula able to tell to BO how good the forecast is in a sampled point.

Parameter	Value	MU
ϕ_{min}	5	°
ϕ_{max}	90	°
$\phi\, samples$	20	
θ_{min}	-150	°
θ_{max}	150	°
$\theta\, samples$	20	
ρ	0.5	m
HPR radius	200	m

Table 6.1: Parameters used for Sphere Gridding Simulation

The shape and the parameters taken into account by this cost function are arbitrary and defined by the user. In this work, it has been decided to implement and test two different cost functions:

$$J_1 = K_r \cdot \ Reachability + K_s \cdot (Score - 1) \tag{6.1}$$

$$\begin{aligned} J_2 = K_r \cdot Reachability + K_s \cdot (Score - 1) + \\ -\{K_p \cdot [Score - \max(Score, Score_p)]\}^2 + \\ +K_b \cdot [Score - \min(Score, Score_b)] \end{aligned} \tag{6.2}$$

where K_r represents the Reachability multiplier, K_s the Score one, K_p the Penalty one, K_b the Boost one and $Score_p$ and $Score_b$ the minimum Score Penalty and Boost thresholds.

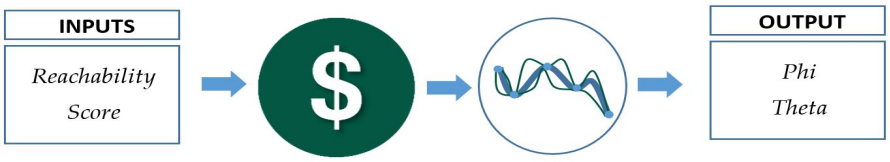

Figure 6.10: Representation of I/O and intermediates passages of Bayesian Optimization: Input Variables, Cost Function Calculation, Mean and Variance Generation and Output Variables

It can be noticed that the cost function J_2 in equation 6.2 contains a quadratic term, which is useful to speed up the optimization, and a boost term to reward if the score is higher than a specific threshold. In addition,

it is important to evidence the fact that Bayesian Optimization algorithm maximizes from $-\infty$ to 0. All of the parameters can be set using the Graphical User Interface created without the necessity to rebuild the code, which would imply idle time for each modification. Also in this case the main used parameters are reported in the table below.

Parameter	Value
Cost Function Used	J_1
K_r	3000
K_s	5000
α	20000
BO samples	10
BO iterations	40

Table 6.2: Parameters used for Sphere Grid Simulation

The last two parameters that need an explanation are BO samples and BO iterations. The former has the aim to initialize the Cost Function, which means that are the calculation executed on random points to start generating Gaussian means, variances and a rough build of the Cost Function before starting the real optimization.

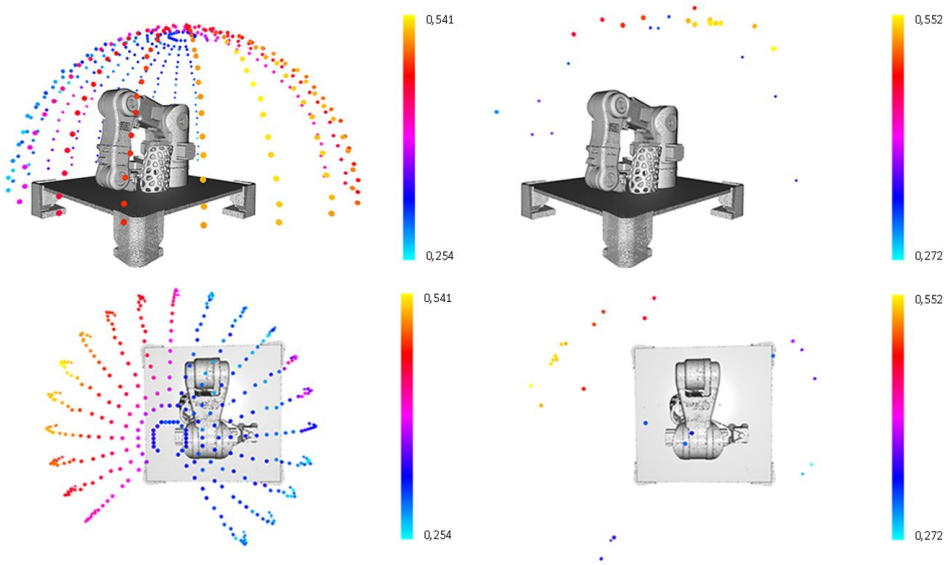

Figure 6.11: Graphical Representation of Sphere Grid (left) and BO with Cost Function J_1 (right)

BO iteration instead identifies after how many points the algorithm stops its research on the Cost Function maximum. Depending also on parameter *alpha*, as explained before, BO samples and iteration must be coherent numbers.

In figure 6.11, the graphical representations of results are reported. It can be clearly understood that Bayesian Optimization algorithm, with very few points sampled, is able to find the optimum viewpoint area. Moreover the maximum score found by the optimization algorithm is slightly increased to the one found by sphere grid method. Indeed, while grid has space between sphere meridians, BO has all the infinite points of the sphere available. In conclusion, BO is able to give slightly better results if compared to sphere grid using less than a tenth of the overall time. Next, two plots representing the variation of Matching Score and Cost Function Values will be proposed using J_1 varying ϕ and θ coordinates.

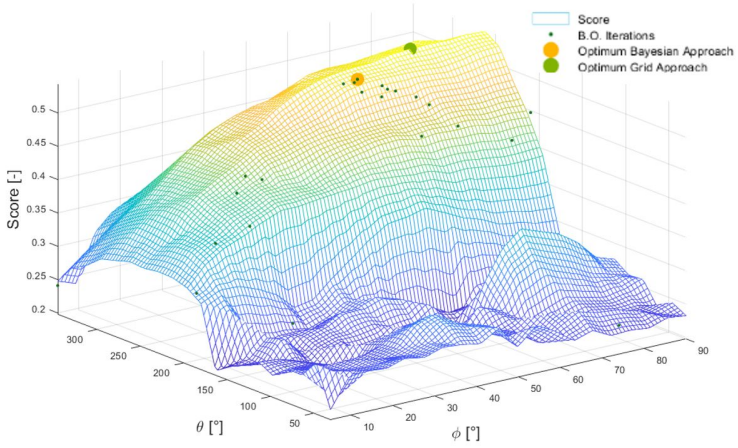

Figure 6.12: Surface Plot Representing Matching Score Values varying ϕ and θ coordinates with data coming from Offline Algorithm

The surface has been created using an interpolation of the Score values coming from the Offline code with grid sampling, while green points represent the Bayesian Optimization Iterations.

As can be possible to exploit, BO climbs the Score surface to reach the maximum values. It is interesting to notice that BO never overcomes the limit of around $\theta = 250°$, also if scores becomes slightly higher. This

because it recognizes the high gradient of the cost function due to the not reachable robot's poses. In order to clarify the concept, also the Figure 6.13 representing Cost Function Values must be compared in the same time.

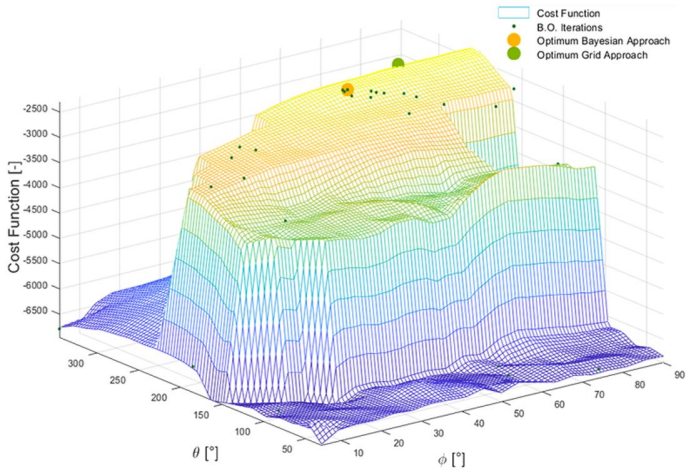

Figure 6.13: Surface Plot Representing Cost Function Values varying ϕ and θ coordinates with data coming from Offline Algorithm

From upper figure, Bayesian Optimization's trend is presented, allowing a better understanding of why some areas of the surface are not considered by BO itself. The best position coming from simulations, for both sphere grid sampling and Bayesian Optimization are now presented.

	Optimal ϕ	Optimal θ	Score
Bayesian Optimization	55 °	244 °	0.552
Sphere Grid Sampling	72 °	262 °	0.541

Table 6.3: Best View Positions from Offline Algorithm

In order to validate the Offline Algorithm respect to reality, a sample of reachable points, in terms of joint limits, has been generated as can be seen below. Only the quarter of the sphere with high matching score was used because the aim is comparing the ones coming from the Offline code with the ones coming from Online, so from real sensor scans in the established positions, to verify if their trend in Offline is equal to the ones in Online.

To allow a better comprehension of validation position, a color gradient based on positions is assigned to each of them and will be used for the plot of results. Matching scores in the colored position can be extracted from the graphs.

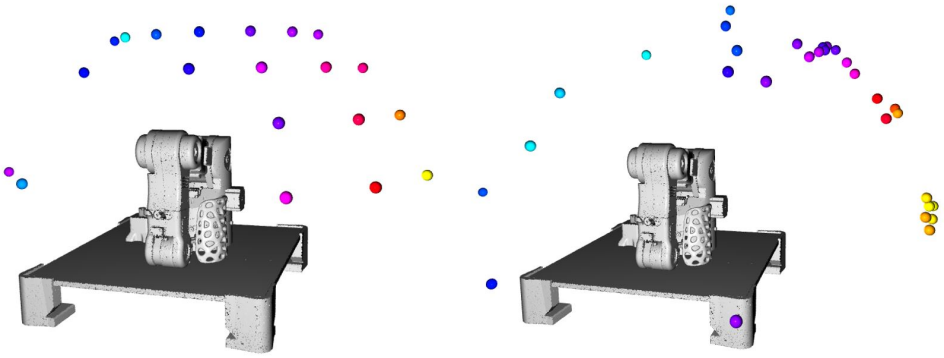

Figure 6.14: Graphical Representation of the Points used for Validation: Sphere Grid (*left*) and Bayesian Optimization (*right*)

Once understood how the validation points has been positioned, the correlation of matching score in simulation and with real sensor data is evidenced by the graph below.

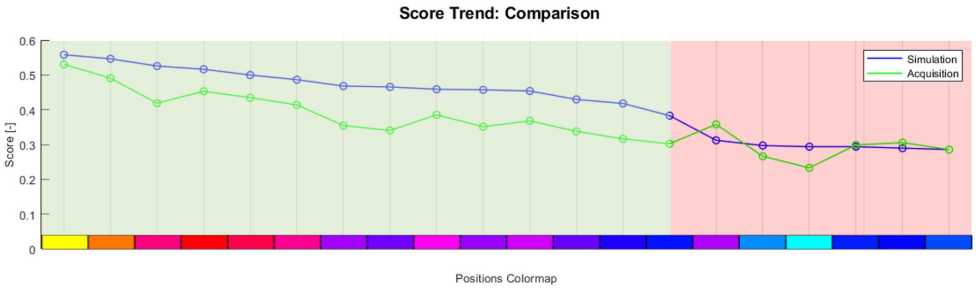

Figure 6.15: Offline and Online Sphere Grid Algorithm Validation

The figure is divided into two different regions: a green one, indicating the good quality of the model matching in the real scene, and a red one reporting the coordinates where the *HALCON* matching code fails the pose estimation, generating false positives due to lack of visibility of the model in real scene with consequently high score gaps. For completeness also wrong results are evidenced. To compare simulation and reality, only the green zone has to be considered.

It is possible to notice that the first positions have the same trend in simulation and reality, the latter evidencing also little peaks and valleys

in score values, probably due to variation of lightning condition between the different points analyzed and random noise. The locations are ordered by score values indicating the best successive sensor's poses to observe the real object to be studied. In addition, the real acquisition score trend curve has an offset if compared to simulation, which increases going further from the best position. This phenomenon can be explained through sensor's point cloud quality. Indeed changing from the best position to the consecutive ones in terms of score, sensor's orientation influence point cloud acquisition: more it is angled respect to the object to study, lower will be the quality of the surfaces, reducing the matching score. Also the point normals calculation influences the result because they are not coming from a mesh with rigidly calculated faces. This further authenticates the fact that the algorithm in reality follows the behavior of simulations validating the all sphere grid system. The same graph indicating Bayesian Optimization score trends is now presented.

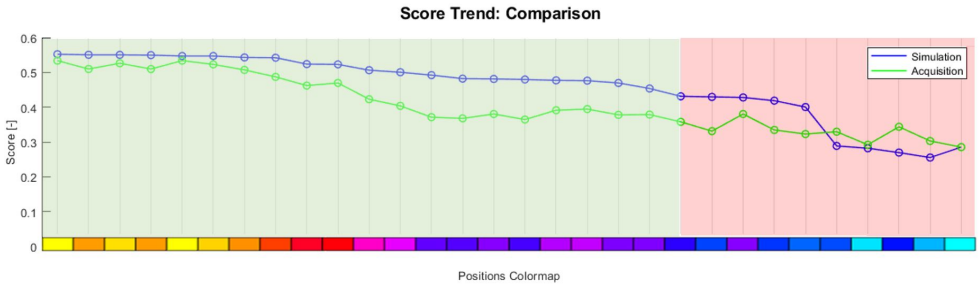

Figure 6.16: Offline and Online Bayesian Optimization Algorithm Validation

As can be noticed, BO reaches the same matching score of sphere grid, the latter executed in its optimum range. It is important to underline that the firsts BO poses are in the neighbourhood of the optimal one presenting a nearly constant behavior in matching values. Also in this case, green and red zone indicates where the object is recognized through matching algorithm and where it is not, due to its complete or too low partial lack of visibility. It is possible to conclude that both sphere grid and Bayesian Optimization Offline codes simulates the reality behavior.

6.5 Sensitivity Analysis of Bayesian Optimization Parameters

Starting from default values for the used open source algorithm $LIMBO$ $RESIBOT^{©}$, a proper sensitivity analysis for the main Bayesian Optimization parameters has to be accomplished. The main targets are:

- Improving speed of the research
- Escaping from local minima
- Setting good parameters for Cost Functions
- Understanding which Cost Function from Equations 6.1 and 6.2, accomplish the best performances

6.5.1 *Alpha* Parameter

One of the key steps of the BO algorithm is the definition of the next point to evaluate. The used library provides several methods such as Gradient-based Optimization, Expected Improvement Algorithm and Upper Confidence Bound. In this work has been decided to use the last option because best fits the purposes. The equation that describes the UCB method is:

$$x_{t+1} = argmax\left[\mu_t(x) + \ \alpha \cdot \sigma_t(x)\right] \tag{6.3}$$

Where α is a user-defined parameter that tunes the trade-off between exploration and exploitation. The UCB function can be seen as the maximum value, $argmax$, across all solutions of the weighted sum of the expected performance, mean of the Gaussian $\mu_t(x)$, and of uncertainty, standard deviation of the Gaussian $\sigma_t(x)$, for each solution. This sum is weighted by the α factor. Lower α signifies that BO gives more importance to optimization problem, surely if it is too low, maybe BO can not find a reasonable optimum stopping in a local maximum area,

continuously trying to further searching the best values in the neighborhood of the local maximum itself. Increasing α instead, more importance in exploration of the unknown cost function will be achieved, raising the probability that at the first maximum reached and after a very low number of BO iterations to optimize it, the successive ones will focus on finding another maximum. In these analysis three values of α were tested using J_1 Cost Function with parameters described in Table 6.4:

$$\alpha = 10^3 \qquad \alpha = 10^4 \qquad \alpha = 10^5$$

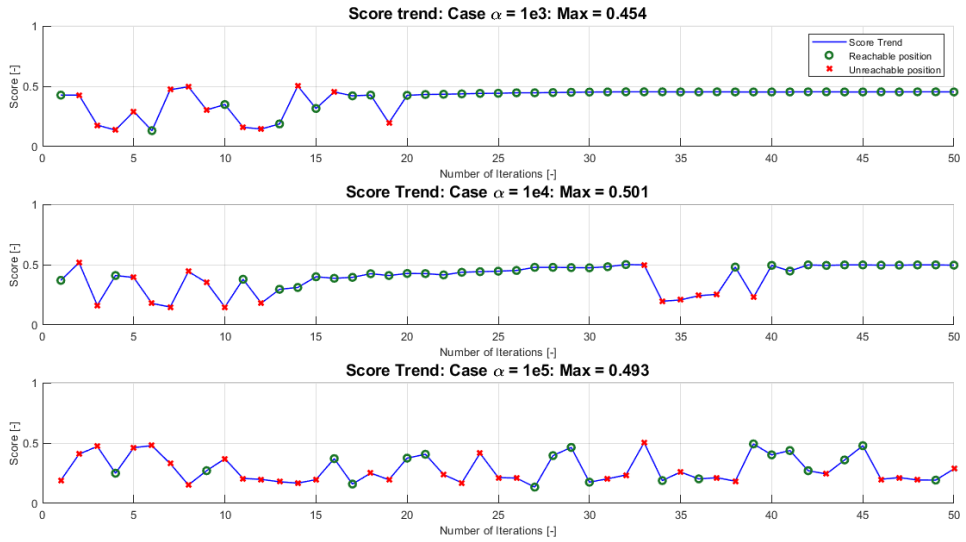

Figure 6.17: Influence of parameter α on Score Matching Values for Cost Function J_1. Red Cross Represent if the specific Position can not be reached by robot's joint limits, Green Circle if can be reached

From the image can be understood how varying α parameter influence exploration and exploitation of Bayesian Optimization. In the first graph, with the lower value of α, BO converges only after ten iterations, remembering that the first ten are random points aimed to have a rough reconstruction of the cost function. In the second one, are found two different optimum values, respectively at tenth and thirtieth iterations. In the last, multiple peaks can be seen, sign that exploration is too preferred on exploitation. Remembering that red crosses evidence the positions that are not reachable by the robot's joints limits, associated to a reachability score equal to -1, is it possible to understand how the penalty score K_r works on reducing the values of J_1 Cost Function.

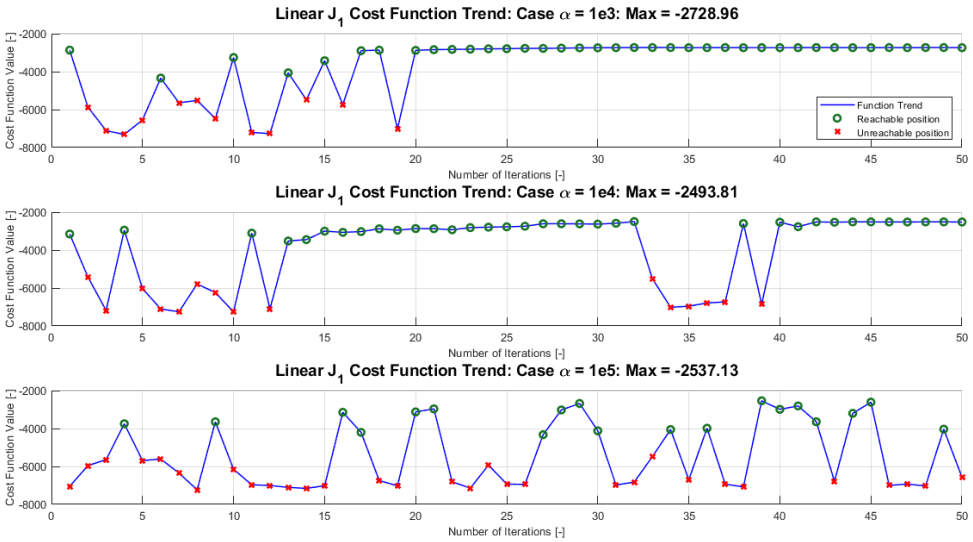

Figure 6.18: Influence of parameter α on for Cost Function J_1 Values

Indeed, in Figure 6.18 multiple negative peaks are in correspondence of red crosses due to the high value of K_r. Convergence is ensured by $K_s\dot{(}Score-1)$, lower the Matching Score value, higher will be the negative number added to the Cost Function Value.

6.5.2 Matching Score and Cost Function Values Comparison between J_1 and J_2

In this section a comparison between Linear and Quadratic Cost Function is presented. For the simulations of J_2, the parameters described in the Table below were used. For both J_1 and J_2, the value of α is set to 10^4.

Parameter	Value	Parameter	Value
K_r	3000	$Score_p$	0.2
K_s	5000	$Score_b$	0.4
K_p	1000	BO samples	10
K_b	2500	BO iterations	40

Table 6.4: Parameters used for J_2 Cost Function

In the graphs below are reported the score values in function of the number of iterations. As can be seen with low values of α the optimiza-

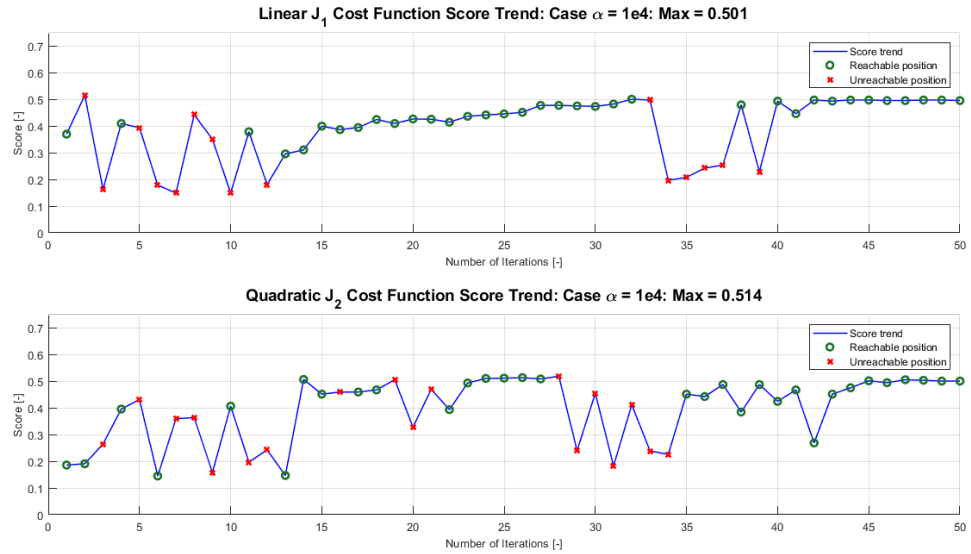

Figure 6.19: Comparison of Matching Values on Linear J_1 and Quadratic J_2 Cost Function with parameter α fixed

tion tends to prefer exploitation with respect the exploration. Indeed, this behaviour could lead to the convergence and stuck of the algorithm in a local minimum as in the first case. The quadratic function J_2 allows a convergence in lower time with a higher matching score, giving time to further exploration on other points.

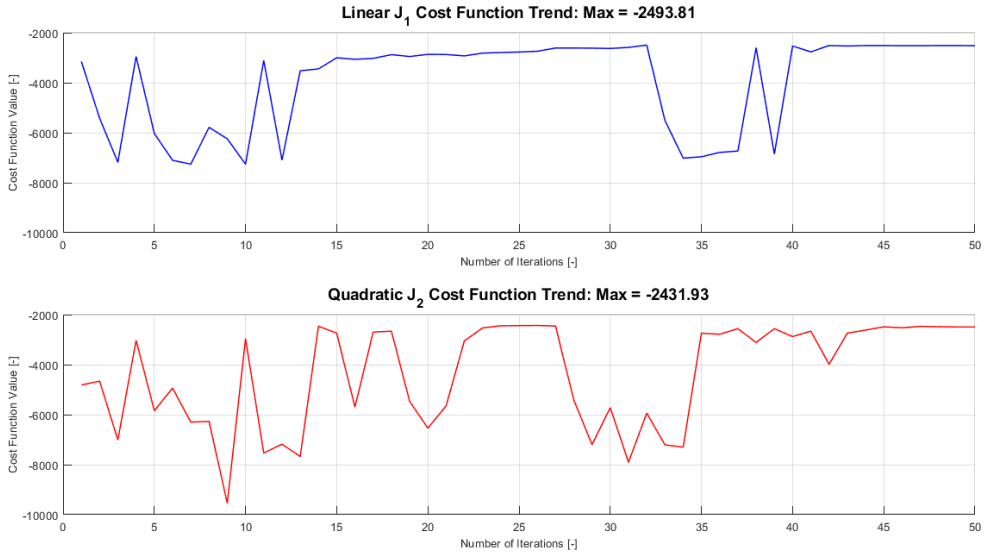

Figure 6.20: Comparison on Linear J_1 and Quadratic J_2 Cost Functions Values with parameter α fixed

The same behavior can be seen also in the graphs that represent the cost functions in relation to iteration number. The cost function is

another key part of the BO that influence its working method.

The behaviour of J_2 respect to J_1 is more uncertain due to the quadratic part of the function, but also more ready and able to react to every factor. Due to this ability, from now on all the experiments will be conducted with J_2 Cost Function and a value of $\alpha = 10^4$, that have been proved to be the better parameters in terms of trade off between exploration and exploitation.

6.6 High Occlusion Level Scene Analysis

Once determined that the entire algorithm is able to produce very good results, further experiments on more complicated scene have been carried out. With this aim, in the scene tested until now, a box is introduced in a specific position to partially occlude the model to be found. The box was placed in order to occlude the best positions generated by Bayesian Optimization and sphere grid expressed in Table 6.5.

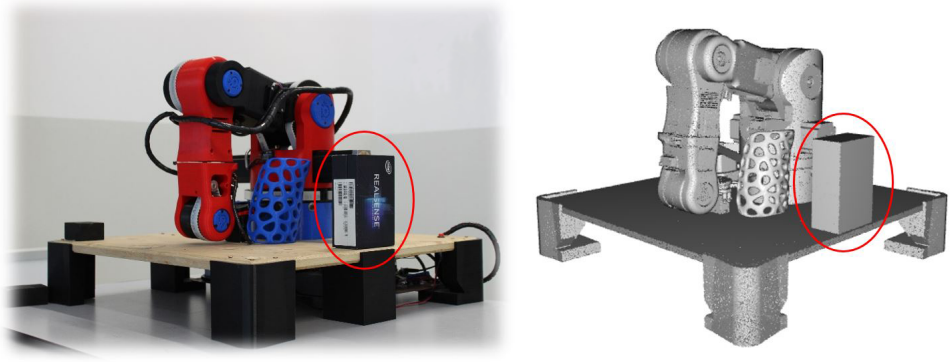

Figure 6.21: Second Scene Tested with higher Occlusion Rate

Using this new scene, all the Offline simulation was carried out both with Bayesian Optimization and Sphere Grid methods.

	Optimal ϕ	Optimal θ	Score
without Occlusion			
Bayesian Optimization	55 °	244 °	0.552
Sphere Grid Sampling	72 °	262 °	0.541
with Occlusion			
Bayesian Optimization	59 °	225 °	0.485
Sphere Grid Sampling	72 °	228 °	0.484

Table 6.5: Best View Positions from Offline Algorithm without and with Occlusion

In the upper table are described the new best position coordinates with relatives score in the new scene simulated with box as an occlusion.

In order to allow a better comprehension of the ϕ_{opt} and θ_{opt} values, in the next image a graphical representation is reported in terms of an arrow, symbolizing also the orientation. First, the results of the non occluded scene are presented and then compared with the occluded one.

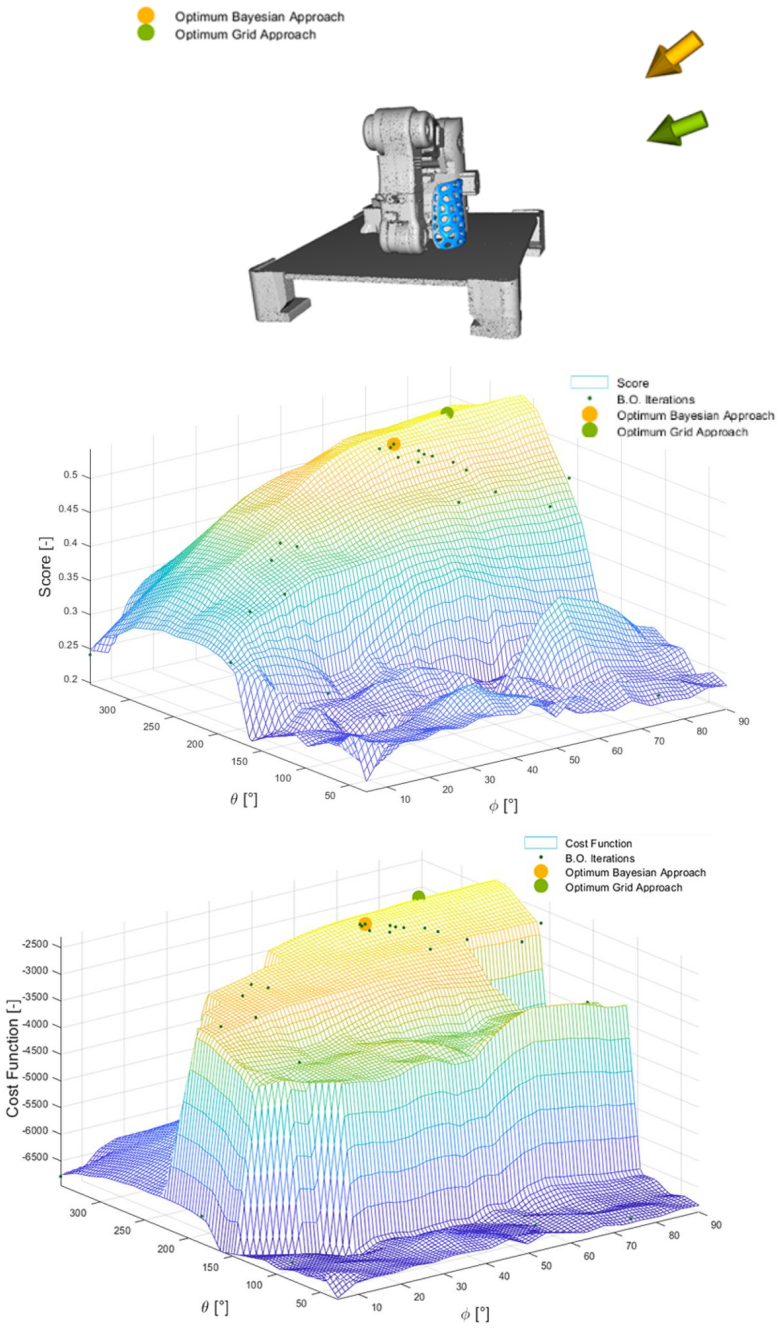

Figure 6.22: Optimal Positions for Model Recognizing in Scene without Occlusion, Score and Cost Function 3D Plots

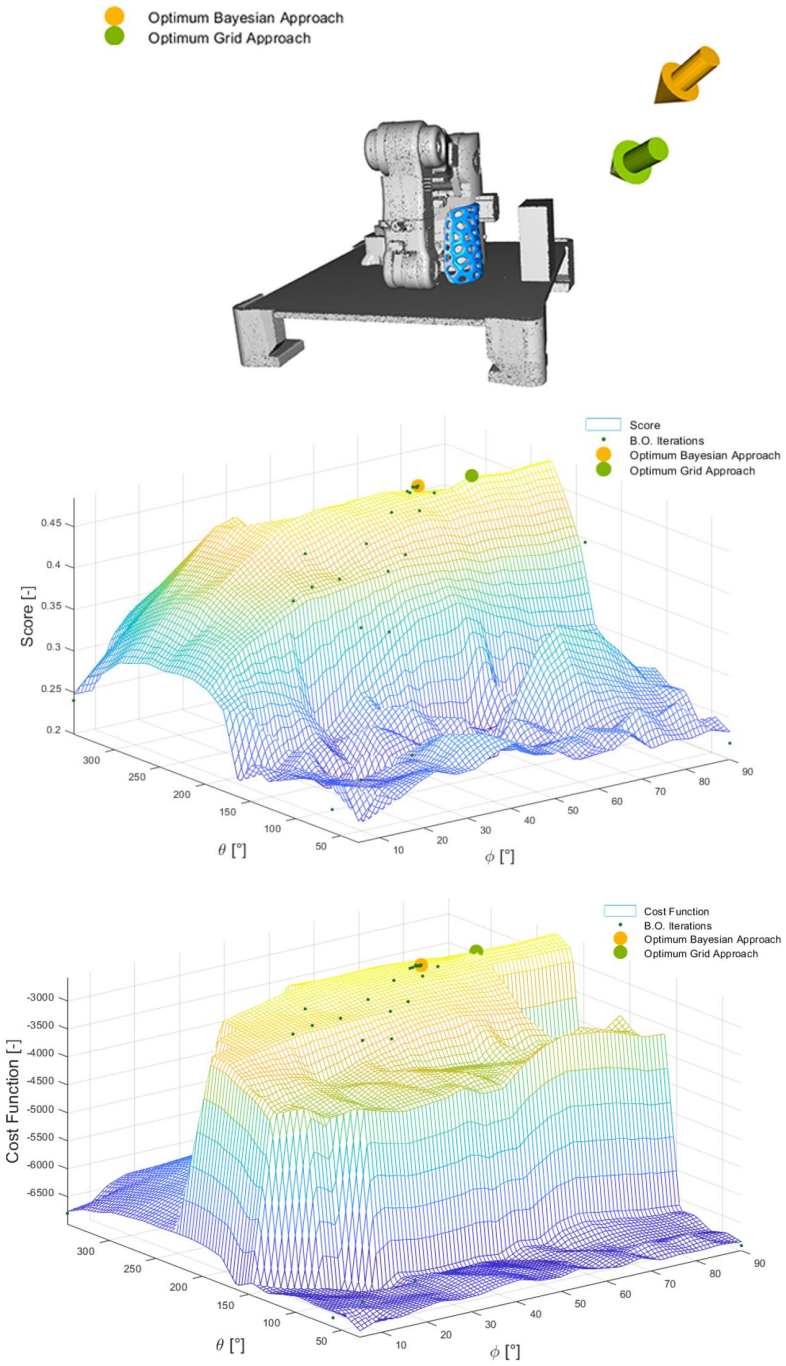

Figure 6.23: Optimal Positions for Model Recognizing in Scene with Occlusion, Score and Cost Function 3D Plots

In this terms, seems that BO can not reach the exact position gained by grid approach. In reality, this is due to the high level of gradient created by the cost function, zone where BO tends to avoid due to dropping values of cost function itself.

To clarify the concept, two different perspectives of surface graphs are now presented describing score and cost function values for the scene with occlusion.

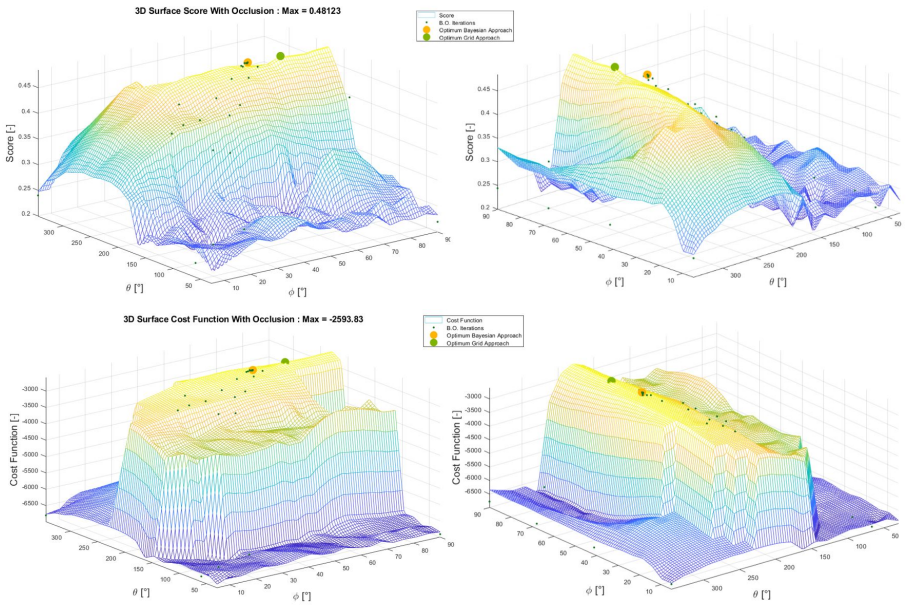

Figure 6.24: Surface Plots Representing Score and Cost Function Values with Occlusion in different Perspectives

The insertion of the occlusion causes a variation in both score and cost function distribution. Indeed, for a value $40° < \phi < 90°$, corresponding to $\theta = 260°$, there was the optimum without occlusion, a huge drop in score and cost function can be exploited, if compared to the ones of Figure 6.22. Where before there was an almost constant plane in cost function, now there is a valley evidencing the decreasing scores of matching. This evidences that the algorithm coded is able to adapts to changes in work environment.

Also in this case, to validate BO respect to Grid Sampling, a reduced number of reachable positions, presented in Figure 6.25, are compared with decreasing score. The same trend as in Figures 6.15 and 6.16, with an offset between score in simulation and reality has been obtained. The same considerations done for the red and green zones are valid, the former represents an optimum matching between scene and model, why the latter a missed one. So, results must be considered valid only inside the green zone.

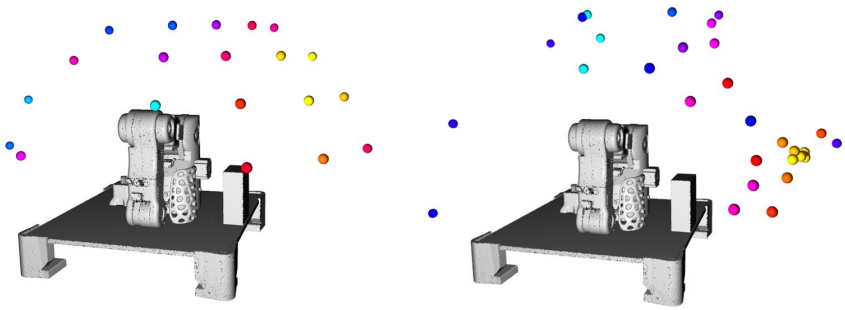

Figure 6.25: Graphical Representation of the Points used for Validation with Occlusion: Sphere Grid (*left*) and Bayesian Optimization (*right*)

The graph below evidences the fact that BO in its optimal position reaches a matching score in reality slightly better than sphere grid methodology. The colormap is used to allow a better comprehension of the robot's in $3D$ space, while for extracting the score at a specific position, in Figure 6.27, color needs to be interpolated with graph points.

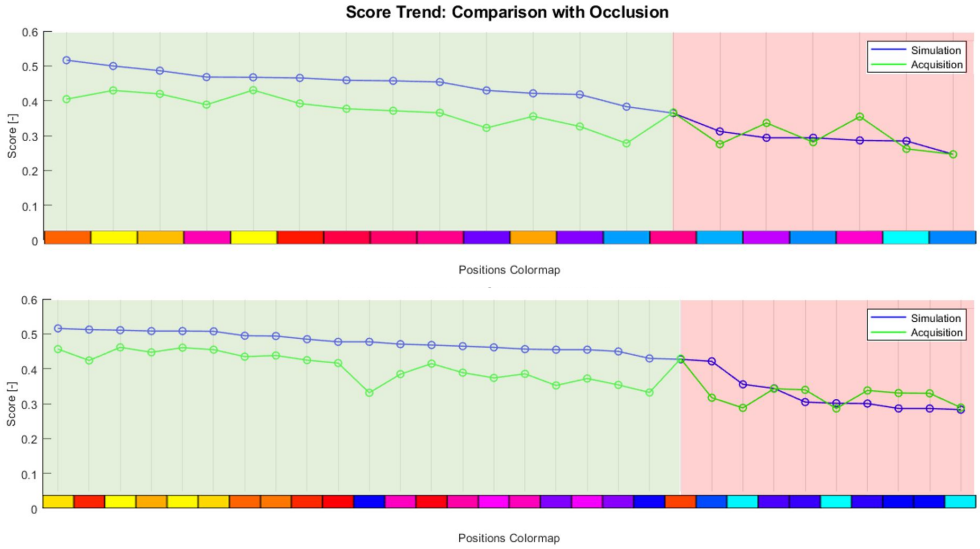

Figure 6.26: Offline and Online Sphere Grid (*top*) and Bayesian Optimization (*bottom*) Algorithm Validation with Occlusion

Concluding, also with an obstacle occluding the view, the code is able to find another optimum position for object studying which has been validated through real experiments.

6.7 Sensor Reconstructed Pointcloud Scene Analysis

In this part of the overall work, attention will be focused on performing the same Offline and Online analysis not on a CAD file of the scene but on the reconstructed one. Indeed, thanks to the methodology explained in Section 5.3, starting from multiple sensor's scans, a single reconstructed point cloud scene was carried out.

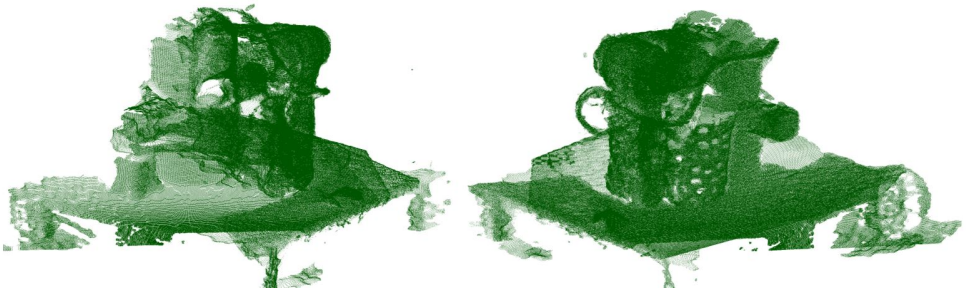

Figure 6.27: Side Views of the Reconstructed Point Cloud Scene starting from five different Scans

As can be possible to imagine, using a point cloud scan as reference model, may lead to raising challenges that were not considered before. As instance, not all the surfaces can be reconstructed due to lack of visibility, generating more difficulties in matching procedure and increasing the probability to obtain false positives. Surface normals used in matching procedure are not rigidly calculated by a mesh and must be estimated, including uncertainties in their esteem. Also matrix $[T_{ee-sens}]$ calculated by calibration and used for the reconstruction of the point cloud will introduce some alignment errors in the reconstruction itself.

All of these problems have been overcome thanks to a fine tuning of the matching, normal calculation and alignment parameters. From now all the graph presented in the previous section of this chapter will be reported with data coming from Offline and Online nodes based on the reconstructed scene.

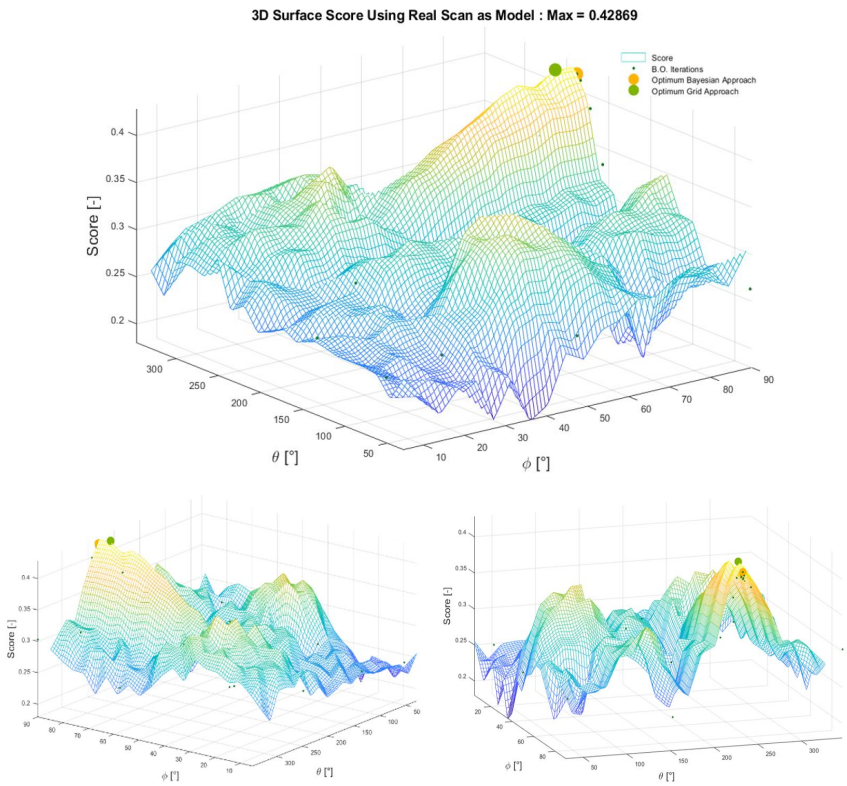

Figure 6.28: Surface Plot Representing Matching Score Values varying ϕ and θ coordinates with data coming from Offline Algorithm based on the Reconstructed Scene

In the upper figure the matching score 3D surface plot varying ϕ and θ is presented. Comparing it with the one coming from Figure 6.12 is possible to notice that using the reconstructed scene as model, the highest matching score decreases by around 20% and the area with score higher than 0.4 is drastically decreased in only a peak.

	Optimal ϕ	Optimal θ	Score
Bayesian Optimization	55 °	244 °	0.552
Sphere Grid Sampling	72 °	262 °	0.541
with Reconstructed Scene			
Bayesian Optimization	89 °	234 °	0.419
Sphere Grid Sampling	84 °	234 °	0.421

Table 6.6: Best View Positions from Offline Algorithm without and with Reconstructed Scene as Model

Table 6.6 evidences the score drop using a reconstructed scene and the drastic change in ϕ for Bayesian Optimization best position and in both ϕ and θ for grid approach. The phenomenon of changing the optimal variables values is strictly correlated and highly dependant on the position in which the point cloud scans has been acquired. Indeed, based on that specific locations, little parts of the model can not be seen or reconstructed in a correct manner. As will be possible to understand also by the results that will be presented, the preferred position for executing the quality control generated by the offline algorithm is possible that will not be the best one, because maybe the reconstructed point cloud misses some details based on the scan location. Also the highest matching score reached by BO is slightly lower than the one obtained by grid approach. This fact can be explained due to the high cost function gradient that is close to that area.

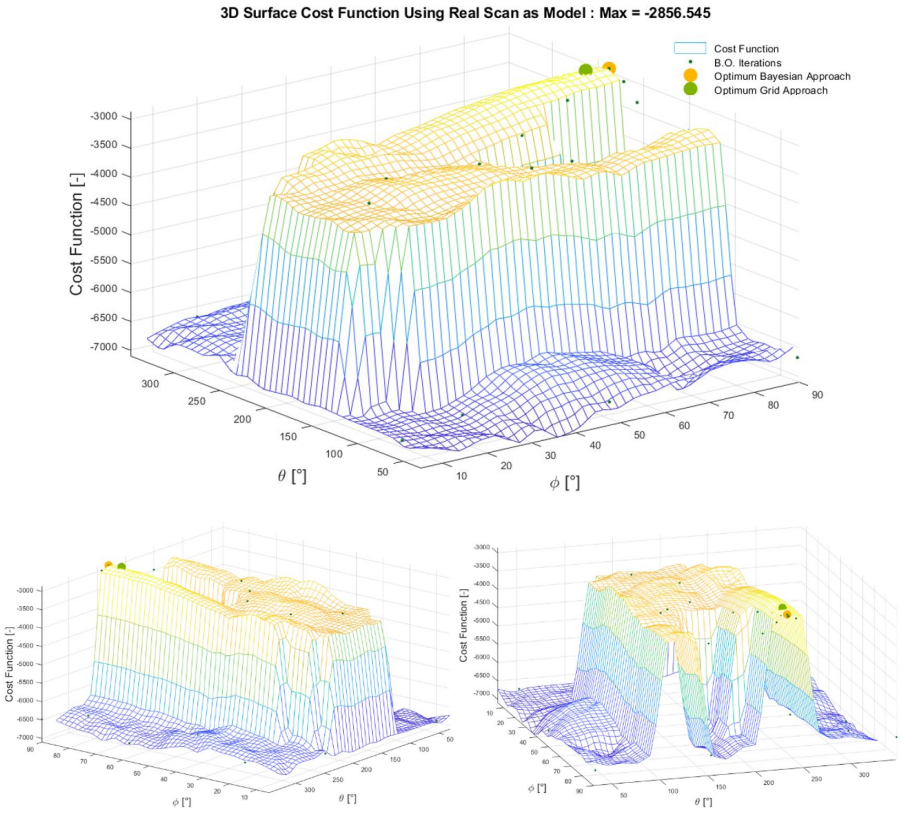

Figure 6.29: Surface Plot Representing Cost Function Values varying ϕ and θ coordinates with data coming from Offline Algorithm based on the Reconstructed Scene

In cost function surface plot instead it is possible to evidence that there are two different areas of lower cost function values. This is due to the two different matching zones that gives high scores enhanced by Figure 6.30. Obviously, the one that sees the model from behind has gained lower score also due to the partial lack of visibility on the model itself.

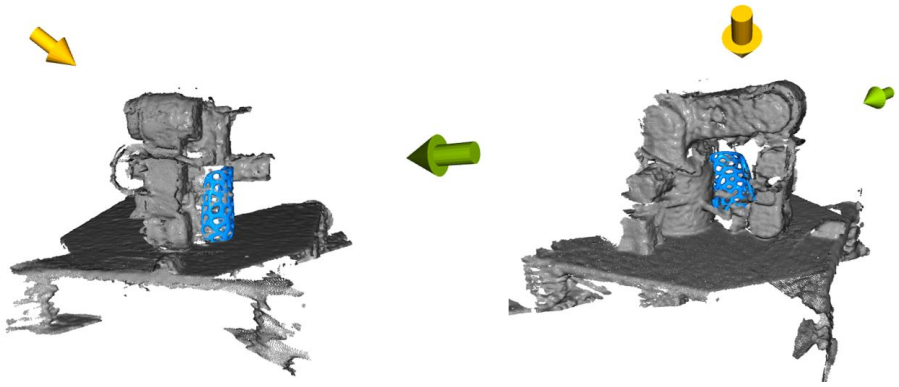

Figure 6.30: Graphical Representation of the Two different Matching Areas

Also in this case, a set of points for validating the Offline algorithm respect to reality has been chosen and are represented in the figure below.

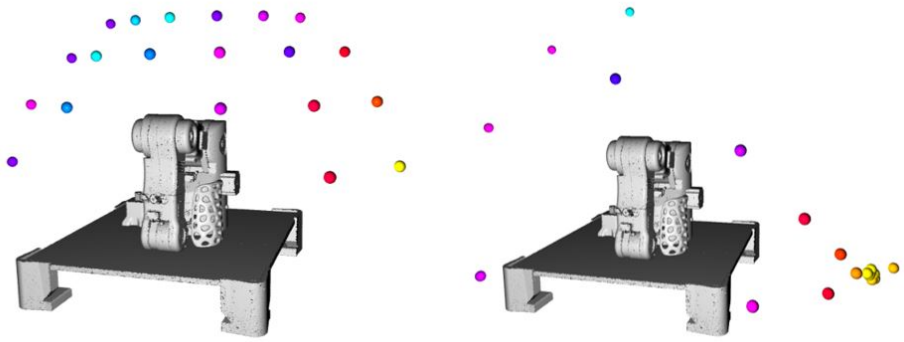

Figure 6.31: Graphical Representation of the Points used for Validation with Occlusion: Sphere Grid ($left$) and Bayesian Optimization ($right$)

In the following plots, comparison between acquisition and simulation score trend, for both Sphere Grid and Bayesian Optimization, concerning the reconstructed point cloud is presented.

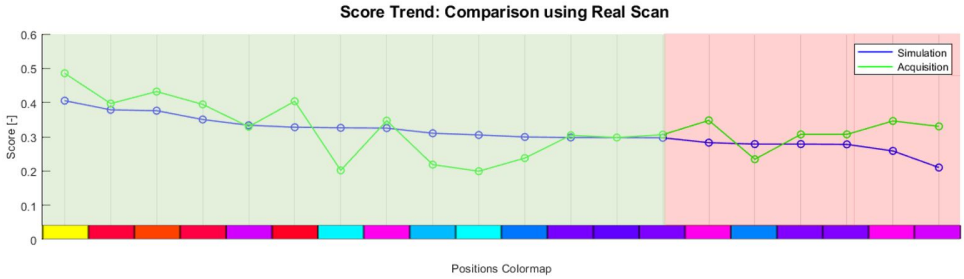

Figure 6.32: Offline and Online Sphere Grid Validation referred to the Reconstructed Point Cloud

Analysing the graph above it is possible to understand how the real score trend differs from the simulated one. This aspect, as mentioned above, is mainly due to the fact that the analyzed poses refer to a reconstructed point cloud, and therefore do not properly represents reality. The real scan acquisitions allow to take into account parts of the model that are missing or not well defined in the reconstructed point cloud leading to obtain better score values in those poses in which the model is only partially matched in simulation. The sphere grid poses analyzed have strong variations in matching score and this greatly emphasizes the difference between real and simulated acquisition, since the processed data are not equal.

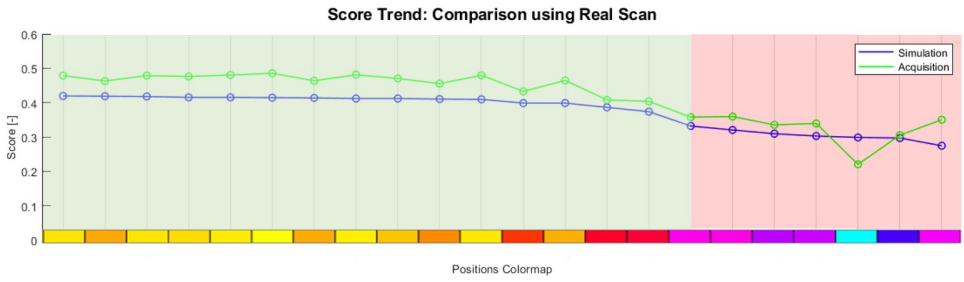

Figure 6.33: Offline and Online BO Validation referred to the Reconstructed Point Cloud

As concerns tests conducted to validate Bayesian Optimization, score progress is shown in the graph above. In this case the acquisition trend follows the simulation one. This is due to the fact that the simulated poses do not change as much as in case of spherical grid but they stay closer to the optimal one. The preferred pose obtained in simulation is similar to one used for the reconstruction, for this reason even small

movements around it bring to comparable results in the real acquisition, as the scans analyzed differ slightly. The high score values obtained are due to the consideration of a single specific point of view for the reconstructed cloud. Probably, real scan evidences characteristics that are not present in simulation leading to increase the matching score.

This analysis highlights as Bayesian Optimization, for reconstructed scene, prevails in estimation of best pose over a simple spherical sampling. The difference in trends found between offline and online grid does not allow the effective estimation of preferred location, as the variation of the poses totally changes the real acquisition from that obtained in simulation. The results obtained through the BO are closely related to reconstructed scene, this may not lead to a global optimal position estimation but allows to obtain comparable results among acquisition and simulation.

Conclusions

In this work the possibility to inspect the quality of mounted pieces was presented using an automated process with the employment of a sensor attached to the end effector of a robot. The problem of choosing the best position of the sensor to extract the object pose in real space was faced and a solution found. The possibility of robot's collisions with the real environment was controlled by sensor's data. The experiments performed demonstrate that Bayesian Optimization produces a time saving option with respect to other proposed methods. In particular, the usage of a quadratic cost function and a parameter that accomplish the best trade off between exploration and exploitation has been motivated for their utilization. In addition, the trends generated by matching scores in simulations and reality had been proved to achieve the same behaviour, validating the model of algorithm proposed. The complex model and scene employed in the application produce an additional confirmation to the initial hypotheses. The insertion of obstacles in the scene had been demonstrated that can be overcome by the algorithm, also in terms of robot's motion planning, thanks to the real time scene update.

In conclusion, the quality inspection was performed positioning the sensor in the best location for recognizing the model inside the scene.

Bibliography

[1] Y. Guo et al. "3D Object recognition in Cluttered Scenes with Local Surface Features: A Survey". In: *IEEE Trans. Pattern Anal. Mach. Intell* (2014).

[2] J.P.S. do Monte Lima and V. Teichrieb. "An Efficient Global Point Cloud Descriptor for Object Recognition and Pose Estimation". In: *IEEE* (2016).

[3] K. Alhamzi, M. Elmogy, and S. Barakat. "3D Object Recognition Based on Local and Global Features Using Point Cloud Library". In: *ResearchGate* (2015).

[4] S. Hinterstoisser et al. "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes". In: *International Conference on Computer Vision* (2011).

[5] T. Hodan et al. "Detection and fine 3D pose estimation of texture-less objects in RGB-D images". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2015).

[6] S.B. Kotsiantis. "Supervised Machine Learning: A Review of Classification Techniques". In: *Informatica* (2007).

[7] C. Wang et al. "6-PACK: Category-level 6D Pose Tracker with Anchor-Based Keypoints". In: *arXiv:1910.10750v1* (2019).

[8] H. Tjaden et al. "A Region-based Gauss-Newton Approach to Real-Time Monocular Multiple Object Tracking". In: *arXiv:1807.02087v2* (2018).

[9] C. Song, J. Song, and Q. Huang. "HybridPose: 6D Object Pose Estimation under Hybrid Representations". In: *arXiv:2001.01869v1* (2020).

[10] W. Kehl et al. "Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation". In: *arXiv:1607.06038v1* (2016).

[11] C. McGreavy, L. Kunze, and N. Hawes. "Next Best View Planning for Object Recognition in Mobile Robotics". In: *Intelligent Robotics Lab School of Computer Science University of Birmingham* (2016).

[12] S. Giancola, M. Valenti, and R. Sala. "A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies". In: *Springer briefs in computer science* (2018).

[13] R. Basri, S. Katz, and A. Tal. "Direct Visibility of Point Sets". In: *ACM Transactions on Graphics* (2007).

[14] B. Drost et al. "Model globally, match locally: Efficient and robust 3D object recognition." In: *Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010).

[15] Hinterstoisser S. et al. "Going Further with Point Pair Features." In: *Computer Vision—ECCV* (2016).

[16] Dai J.S. "Euler–Rodrigues formula variations, quaternion conjugation and intrinsic connections." In: *Mech. Mach. Theory* (2015).

[17] Shiu Y. C. and Ahmad S. "Calibration of Wrist-Mounted Robotic Sensors by Solving Homogeneous Transform Equations of the Form AX=XB". In: *IEEE Transactions on Robotics and Automation* (1989).