

UNIVERSIDAD POLITÉCNICA DE MADRID

MASTER'S THESIS

Novel Pose Estimation System for Precise Robotic Manipulation at CERN

Author:

Jorge Camarero Vera

Supervisor:

Manuel Ferre Pérez

Mario di Castro

*A thesis submitted in fulfillment of the requirements
for the degree of Master in Automation and Robotics*

February 3, 2017



"Everything that is possible demands to exist."

Gottfried Leibniz

Abstract

Novel Pose Estimation System for Precise Robotic Manipulation at CERN

The “mission” of tele-robotics at CERN may be resumed in the following: Ensuring **safety** of personnel improving **availability** of CERN’s accelerators. The robots that are being developed at CERN should offer visual capacities, among them the capacity to estimate the 6D pose of an object. To know the position and rotation of an object could be used in augmented reality for a better information for the tele-operator and for the realization of semi-autonomous tasks.

The goal of this project is to develop a novel and reliable algorithm to estimate the 6D pose, using computer vision, of objects that are going to be telemanipulated by robots. The novel algorithms should work consistently in the CERN unstructured and harsh environments, presenting several constraints like variable luminosity, difficult accessibility and light reflections.

In this project, an algorithm has been developed to detect the position and rotation of objects in the LHC tunnel using 3D cameras. The objects selected for this project correspond to a collimator. The collimators are of vital importance for the correct operation of the LHC and therefore, have great importance in technical inspections. In addition, they are one of the hot spots of radiation in the LHC.

The algorithm has been developed using Point Cloud Library to manage the point cloud created with an RGBD Camera. Due to reflections on the target material, the point cloud contains partial information of the desired object. Once the position and rotation of the object is known, it is possible to reconstruct the lost information using CAD models.

Acknowledgements

I want to thank mainly my parents who have always supported me. I also thank Professor Manuel Ferre and Mario di Castro for the opportunity to spend six months at CERN to develop the project described in this document.

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
1.1 Introduction to CERN	1
1.1.1 CERN's Accelerators	2
1.1.2 Purpose of LHC	4
1.1.3 LHC layout	5
1.2 Tele-Robotics at CERN	8
1.2.1 Difficulties for Tele-Robotics and Inspections at CERN	9
1.3 CERNTAURO	10
1.4 Description of the Problem and Objectives	11
2 State of the Art	13
2.1 Problem Formulation	13
2.2 Introduction to Computer Vision	13
2.3 Computer Vision Systems	14
2.3.1 Monocular Camera	17
2.3.2 Stereo Camera	20
2.3.3 LiDAR	21
2.3.4 RGB-D Cameras	23
Microsoft Kinect V2	25
Intel RealSense	27
Orbbec Astra Pro	28
2.4 Computer Vision Frameworks	30

2.4.1	OpenCV	30
2.4.2	Point Cloud Library	30
	PCL Algorithms	31
2.5	Algorithms	31
2.5.1	LSD-SLAM	31
2.5.2	LINE-MOD	34
2.5.3	PWP3D	37
2.5.4	Sliding Shapes	37
2.6	Deep Learning Algorithms	38
2.6.1	Paper of Amazon Picking Challenge	39
2.6.2	Deep Sliding Shapes	41
3	Pose Estimation System Proposed	43
3.1	Introduction	43
3.2	Development	45
3.2.1	Separator	49
3.2.2	Collimator	61
3.2.3	General Objects	68
4	Implementation and Validation	69
4.1	Introduction	69
4.2	RGBD Camera Class	70
4.2.1	HDL32E Grabber	73
4.3	Tests and Validation	75
4.3.1	Robustness of the Algorithm	78
4.3.2	Accuracy of the Algorithm	81
4.3.3	Degree of Autonomy of the Algorithm	82
4.4	Integration with the CERNTAURO Framework	83
4.5	Integration with Virtual Reality	84
5	Conclusions and Future Development	87
5.1	Conclusions	87
5.2	Future Development	87

6 Planning and Scope of the Project	89
6.1 Work Breakdown Structure	89
6.2 Gantt Diagram	91
Bibliography	97
A Budget	107

List of Figures

1.1	(a) The CERN's site form above, showing the two main sites, the Swiss one in Meyrin and the French one in Preessin. (b) the LHC and SPS tunnel at an average of 100 m underground.	1
1.2	The CERN accelerator complex, including all the secondary experiments like ISOLDE, nToF, CNGS, etc.	5
1.3	(A) Two LHC dipole magnets. (B) An RF Cavity in the LHC tunnel. (C) Two near collimators in LHC Sector 7.	6
1.4	Examples of constraints at CERN.	10
1.5	Two models of separator.	12
1.6	Photograph of an installed ring collimator on its quick plug-in support in IR8 of the LHC[30].	12
2.1	Timeline with some active topics of research in computer vision[34].	14
2.2	Evolution of Imagenet Error Rate. Source: Nervana.	15
2.3	Data from a Intel RealSense camera.	16
2.4	IDS UI-2220SE-C-HQ Industrial Camera.	17
2.5	Deformed image produced by optics.	19
2.6	Set of images to do a photogrammetry.	19
2.7	Photogrammetry of one of the separators.	20
2.8	Stereo camera with two IDS camera	21
2.9	LiDAR Velodyne HDL-32E.	22
2.10	We can see two main problems: a low density of points in the target and a different reflectance in the same material.	23

2.11 Mode Region Scanning of <i>RobotEye RE05</i> . This mode allows the user to define a rectangular region within the scanner's azimuth and elevation field in which to concentrate the scanning[42].	23
2.12 The black zones are null data in the depth field caused by reflections of the light of the projector.	24
2.13 3D reconstruction of the target done in real time with camera tracking by using <i>Kinect Fusion</i>	25
2.14 Microsoft Kinect V2[48].	26
2.15 Point cloud of the object of interest from a Kinect V2. It has a dense point cloud, but with a high noise.	26
2.16 Intel RealSense cameras used in this project.	27
2.17 Diagram of Intel RealSense R200. It has a color camera, 2 IR cameras and an IR laser projector[50].	27
2.18 Diagram of Intel RealSense SR300. It has a color camera, an IR camera and an IR laser projector[51].	28
2.19 Intel RealSense cameras used in this project.	29
2.20 Overview over the complete LSD-SLAM algorithm[64].	32
2.21 Logitech C920 HD Pro.	33
2.22 Point clouds of the object of interest using LSD-SLAM. The image on the left shows reflections.	33
2.23 Point clouds of a computer created using LSD-SLAM.	34
2.24 Texture-less 3D objects detected in realtime under different poses over heavily cluttered background using an image and its depth map[70]	35
2.25 Example of PWP3D detecting a soda can with motion blur[77].	37
2.26 Graphical explanation of <i>Sliding Shapes</i>	38
2.27 System purpose in[80].	40
2.28 Problems with objects which generate holes and noise in the point cloud as the plastic bottle.[80].	40
2.29 3D Amodal Region Proposal Network.[82].	41
2.30 Joint Object Recognition Network[82].	42
3.1 Prove of concept using a Kinect V2.	43

3.2	Final result of the algorithm.	44
3.3	Different operations using Paraview with PCL.	45
3.4	Objects to detect.	49
3.5	Input point cloud.	50
3.6	Point cloud with the most distant points filtered.	51
3.7	Planar segmentation. The red zone is the extracted plane.	53
3.8	Clustering of the surface.	56
3.9	Point clouds of the CAD models.	59
3.10	Reconstruction of the separator.	61
3.11	Input point cloud of the collimator.	62
3.12	Filtered point cloud of the collimator.	62
3.13	Lateral view of the point cloud of the collimator.	63
3.14	Extraction of the oblique plane.	64
3.15	Clustering of the oblique plane.	64
3.16	The red dots belong to the oblique plane previously detected and the white dots are those that are saved after using the <i>HeightFilter</i>	65
3.17	Segmentation of the upper part of the collimator.	66
3.18	Point cloud of the collimator CAD model	67
3.19	Reconstruction of the collimator.	67
4.1	Mobile platform with robot arm being developed at CERN. On the arm tool is the Orbbec camera.	69
4.2	UML implementing the factory pattern for the management of RGBD cameras.	70
4.3	Point cloud datasets of the separator and collimator.	75
4.4	There are bad point clouds in the dataset, because a problem in the Orbbec camera. These point clouds are useless.	76
4.5	The dimensions of the LHC tunnel are limited. The datasets were made in a model of a segment of LHC with similar dimensions.	76
4.6	Reconstruction of different point clouds of the dataset.	77
4.7	Object detection using YOLOv2[93].	82
4.8	Object segmentation using <i>DeepMask</i> [94] and <i>SharpMask</i> [95].	83

4.9	The developed in this project corresponds to RGBD Cameras and 3D Object Tracking.	84
4.10	Segment of LHC tunnel in Virtual Reality using Unity.	85
6.1	Work Breakdown Structure.	90
6.2	Work Breakdown Structure.	92
6.3	Work Breakdown Structure.	93
6.4	Work Breakdown Structure.	94
6.5	Work Breakdown Structure.	95

List of Tables

2.1	Sensor Specification of an IDS UI-2220SE-C-HQ Industrial Camera[37].	17
2.2	Main specifications of Microsoft Kinect 2[48].	25
2.3	Some features of Intel RealSense R200[50].	28
2.4	Some features of Intel RealSense SR300[51].	28
2.5	Orbbec Astra Pro technical specs[53].	29
A.1	Costs of Human Resources.	107
A.2	Costs of Material.	108
A.3	Total Cost of the Project.	108

Listings

2.1	IDSCamera.h	18
3.1	header	45
3.2	example.cpp	48
3.3	DepthFilter	50
3.4	ExtractSlice	51
3.5	ExtractLegs	54
3.6	FindCorners	56
3.7	ExtractCenter	58
3.8	ModelTranslation	60
3.9	HeightFilter	65
4.1	RGBDCamera.h	71
4.2	RGBDCameraFactory.hpp	71
4.3	rgbGrabber.cpp	72
4.4	HDL32E.h	73
4.5	CloudMapping.hpp	78

List of Abbreviations

3D	3 Dimensional
6D	6 Dimensional
ALARA	As Low As Reasonably Achievable
ALICE	A Large Ion Collider Experiment
AD	Antiproton Decelerator
ALSM	Airbone Laser Swath Mapping
ATLAS	A Toroidal LHC ApparatuS
BSD	Berkeley Software Distribution
CAD	Computer-Aided Design
CCC	CERN Control Center
CERN	Conseil Européen pour la Recherche Nucléaire
CERNTAURO	CERN Telemanipulation semi-Autonomous Unit for Robot Operations
CNGS	CERN Neutrinos to Gran Sasso
CMS	Compact Muon Solenoid
ConvNet	Convolutional Network
CUDA	Compute Unified Device Architecture
FCN	Fully Convolutional Network
FOV	Field Of View
IR	InfraRed
LSVRC	Large Scale Visual Recognition Challenge
ISOLDE	Isotope Separator On-Line DEtector
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty
LHCf	Large Hadron Collider forward
LEIR	Low-Energy Ion Ring

LiDAR	L ight D etection A nd R anging
LINAC	L INear A Ccelerator
LSD-SLAM	L arge S cale D irect M onocular S LAM
MoEDAL	M onopole and E xotics D etector A t the L HC
nTOF	n eutron T ime- O f- F light
OCR	O ptical C haracter R ecognition
OpenCV	O pen C omputer V ision
ORN	O bject R ecognition N etwork
PCL	P oint C loud L ibrary
PS	P roton S ynchrotron
PSB	P roton S ynchrotron B ooster
RF	R adio F requency
RGB	R ed G reen B lue
RGB-D	R ed G reen B lue - D epth
RPN	R egion P roposal N etwork
SLAM	S imultaneous L ocalization A nd M apping
SPS	S uper P roton S ynchrotron
SVM	S upport V ector M achine
ToF	T ime o f F light
TOTEM	T OTAl E lastic and diffractive cross section M easurement
VGA	V ideo G raphics A rray

For/Dedicated to/To my...

Chapter 1

Introduction

1.1 Introduction to CERN

CERN, the European Organization for Nuclear Research, is considered one of the world's largest and most respected laboratories for scientific research, and it is located in the North-west suburbs of Geneva on the Franco-Swiss border. It currently has twenty European member states, as well as Israel and Pakistan, which are the only non-European countries granted full membership. Nevertheless, scientist coming from non member states can collaborate with CERN, making it one of the most multicultural and diverse environments in the World. Established in 1954 and evolving ever since, CERN has been at the forefront of science research and several important discoveries have taken place in it, leading among others to two Nobel physics prizes and the invention of the World Wide Web.

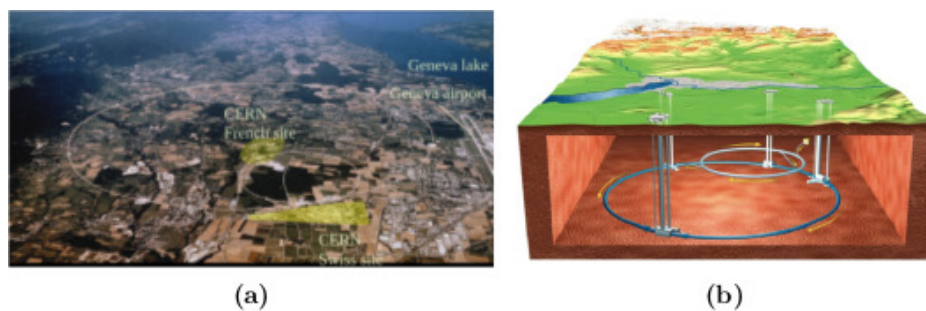


FIGURE 1.1: (a) The CERN's site form above, showing the two main sites, the Swiss one in Meyrin and the French one in Preessin. (b) the LHC and SPS tunnel at an average of 100 m underground.

Currently, more than 12000 scientists (belonging to 113 different nationalities and representing 608 universities and research facilities in the world) are collaborating at CERN[1].

CERN's main function is to provide the particle accelerators and other infrastructure needed for high-energy physics research. As a result, numerous experiments have been constructed at CERN following international collaborations. Around the construction and the maintenance of the accelerators, a lot of research is done involving medicine, engineering and computer science in order to provide the best research facilities all over the world.

1.1.1 CERN's Accelerators

The accelerators complex at CERN is a succession of machines that accelerate particles to increasingly higher energy. It is located near Geneva at an average of 100 meters underground (50 m near Geneva's Lake, 175 m near the Jura Mountains). The Large Hadron Collider (LHC)[2] is the biggest particles accelerator all over the world, and it is the last accelerator of a long chain that increases the energy of the particle to 7-8 TeV per beam.

As soon as the particles are produced from the hydrogen ionization, protons are injected into the Linear Accelerator 2 (LINAC 2)[3], the first in the chain, that increases the energy of the particle to 50 MeV. The beam is then injected into the Proton Synchrotron Booster (PSB)[4] that increases the energy to 1.4 GeV. Before the Booster received its first beams on May 26th 1972, protons were injected directly from the LINAC into the Proton Synchrotron (PS)[5], where they were accelerated to 26 GeV. The low injection energy of 50 MeV limited the number of protons that the PS could accept. The Booster allows the PS to accept over 100 times more protons, which greatly enhances the beam's use for experiments.

The Booster is followed by the PS, which pushes the beam to 25 GeV. It is a 628 meters circular accelerator with 277 magnets (room-temperature), including 100 dipoles to bend the beam around the ring.

Particles are sent to the Super Proton Synchrotron (SPS)[6]: it measures 7 km circumference and accelerates the beam to 450 GeV using 1317 room-temperature magnets. In 1983, through protons-antiprotons collisions, SPS experiment discovered two new particles, W

and Z, taking the Nobel Prize to CERN[7].

The protons are finally transferred to the two beam pipes of the Large Hadron Collider (LHC). It takes 4 minutes and 20 seconds to fill each beam pipe, and it takes 20 minutes for each beam to reach the energy of 4 TeV. Then the collisions can start, in the four detectors around the LHC ring (ALICE[8], ATLAS[9], CMS[10], LHCb[11]) where the total energy at the collision point is 15 TeV. Inside LHC, two high-energy particle beams travel close to the speed of light before they are made to collide. The beams travel in opposite directions in separate beam pipes, inside two tubes kept at ultra-high vacuum. They are guided around the accelerator ring by a strong magnetic field (around 7 Tesla) maintained by superconducting electromagnets[12]. The electromagnets are built from coils of special electric cable which operates in a superconducting state, efficiently conducting electricity without resistance or loss of energy. This requires chilling the magnets to -271.3C, a temperature colder than outer space. For this reason, much of the accelerator is connected to a distribution system of liquid helium, which cools the magnets, as well as to other supply services. These include 1232 dipole magnets 15 metres in length, which bend the beams, and 392 quadrupole magnets, each 5-7 metres long, which focus the beams. Just prior to collision, another type of magnet is used to "squeeze" the particles closer together to increase the chances of collisions. The particles are so tiny that the task of making them collide is akin to firing two needles 10 kilometres apart with such precision that they meet halfway. All the controls for the accelerator, its services and technical infrastructure are housed under one roof at the CERN Control Centre (CCC) in the Preessin site[13].

The accelerator complex includes the Antiproton Decelerator (AD)[14] and the Online Isotope Mass Separator (ISOLDE) facility[15], and feeds the CERN Neutrinos to Gran Sasso (CNGS) project[16] and the Compact Linear Collider test area[17], as well as the neutron time-of-flight facility (nTOF)[18].

Protons are not the only particles accelerated in the LHC. Lead ions for the LHC start from a source of vaporized lead and enter LINAC 3[19] before being collected and accelerated in the Low-Energy Ion Ring (LEIR)[20]. Then, they follow the same route to maximum

energy as the protons.

After the first long shut-down from the beginning of 2013 to the end of 2014[21], all the accelerators have been improved, allowing LHC to increase the maximum energy from 4 TeV to 7-8 TeV per beam, with a collision energy of 15 TeV. Additionally, LINAC 4 has been built[22], replacing the LINAC 2 and increasing the injection energy to the PS Booster to 180 MeV.

1.1.2 Purpose of LHC

The LHC was built in order to answer to some fundamental open questions in particles physics, especially concerning the Standard Model, and to validate 40 years of predictions in theoretical physics.

One of the main goals was the validation of the Higgs mechanism theory, through the observation of the Higgs Boson. On July 4th, 2012 CERN announced that the two independent experiments, ATLAS and CMS, made the same discovery[23]. Using the combined analysis of two interaction types (known as channels), both experiments independently reached a local significance of 5-sigma - less than a one in three-and-a-half million chance of error. When additional channels were taken into account, the CMS significance was reduced to 4.9-sigma. At the end of 2013 Peter Higgs and François Englert won the Nobel Prize for their prediction of the Higgs mechanism.

Another important theory that has to be confirmed, is the existence of Super-symmetric partners[24]. Physicists hope that the LHC will help answer some of the fundamental open questions in physics, concerning the basic laws governing the interactions and forces among the elementary objects, the deep structure of space and time, and in particular, the interrelation between quantum mechanics and general relativity, where current theories and knowledge are unclear or break down altogether.

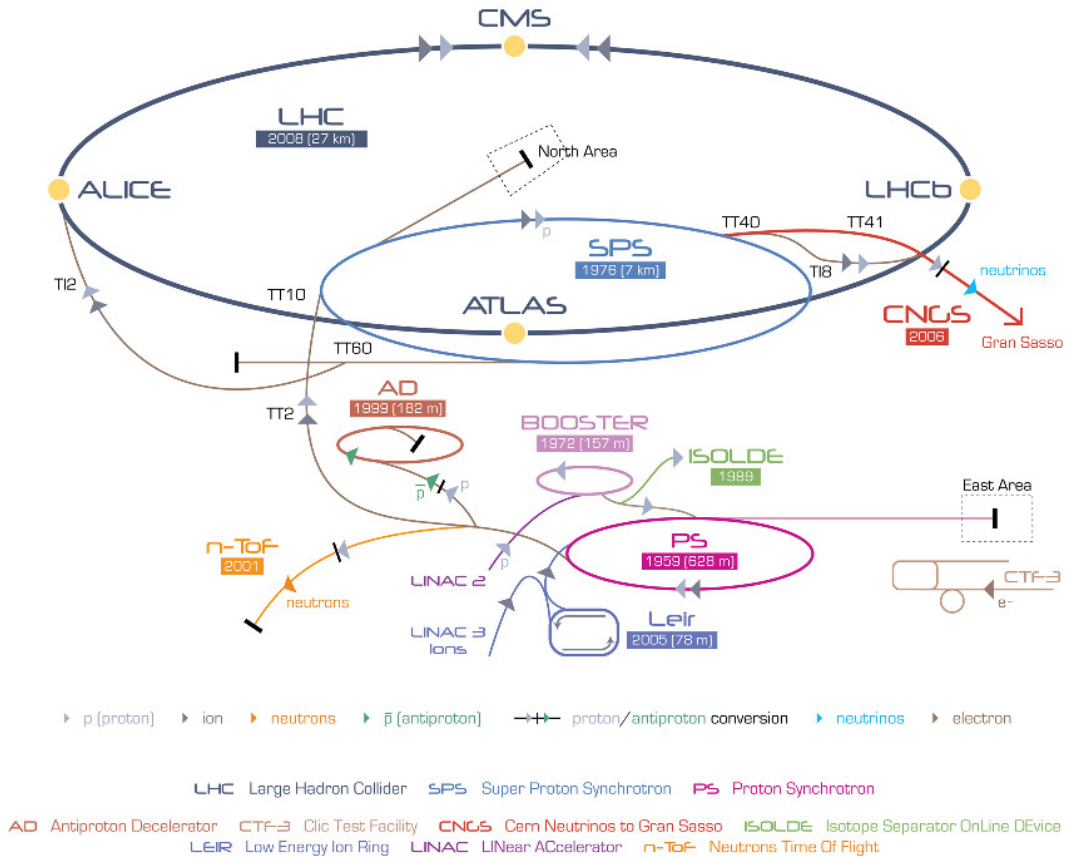


FIGURE 1.2: The CERN accelerator complex, including all the secondary experiments like ISOLDE, nToF, CNCS, etc.

1.1.3 LHC layout

The LHC is composed by several components, which accomplish distinguished tasks. Although vast in number, the main components of the LHC can be summarized in the following fashion. The detectors, of course, are the key elements. Four particle detectors have been constructed and housed in huge underground caverns, corresponding to the four collision points of the two beams. These are ATLAS (A Toroidal LHC ApparatuS), CMS (Compact Muon Solenoid), ALICE (An LHC Ion Collision Experiment) and LHCb (LHC beauty). ATLAS and CMS are designed to study as broad a variety of physics as possible and are considered to be general-purpose experiments. ALICE is dedicated to heavy ions collisions, and it recreates conditions similar to those just after the Big Bang in order to study and analyse the properties of the quark-gluon plasma. The LHCb is dedicated to B-physics phenomena;

in other words, it focuses on B-hadron decays, which should give better insight into the differences between matter and anti-matter as well as possibly providing indirect evidence for SM-violating processes. In addition, there are also the TOTEM (Total Cross Section, Elastic Scattering and Diffraction Dissociation at the LHC)[25], LHCf (Large Hadron Collider forward)[26] and MoEDAL (the Monopole and Exotics Detector At the LHC)[27].

Other important elements in the LHC operation are the superconducting magnets, Figure 1.3a. They are responsible for keeping the beam stable and precisely aligned. Specifically, dipole magnets are some of the most complex parts of the LHC and are used to bend the trajectory of the particles. This is absolutely necessary as the LHC is a circular accelerator, which means that the beam circulates many times in closed orbits before reaching its peak energy level, in contrast to a linear accelerator, where charged particles travel along a straight trajectory and go through a number of accelerating structures. There are 1232 main dipoles installed in the LHC, each 14.3 metres long and weighing in at 35 tonnes. Other vital parts are the quadrupole magnets. These help to keep the particles in a tight beam; in other words, they focus the beam. The idea behind this is that when particles are bunched together, they are likelier to collide in greater numbers when they reach the LHC detectors. In addition, to the dipole and quadrupole magnets, there are also smaller sextupole, octupole and decapole magnets to correct small imperfections in the magnetic field at the extremities of the dipoles.

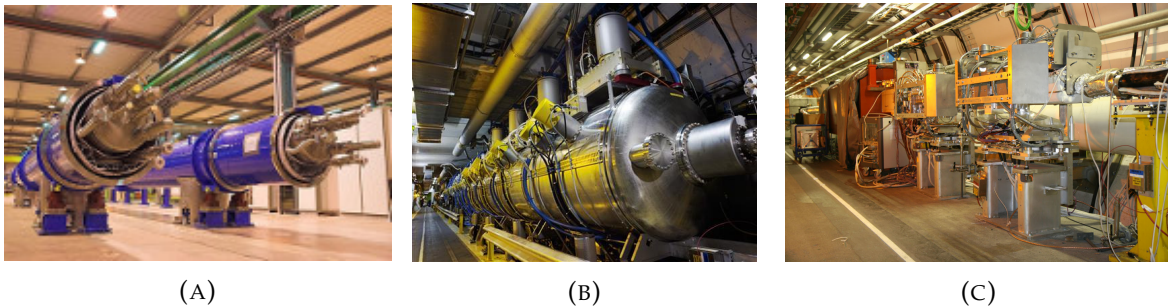


FIGURE 1.3: (A) Two LHC dipole magnets. (B) An RF Cavity in the LHC tunnel. (C) Two near collimators in LHC Sector 7.

Radiofrequency cavities[28], Figure 1.3b, are also of great importance. An RF cavity, whose primary purpose is to accelerate charged particles, is a metallic chamber containing

an electromagnetic field. The RF cavity is constructed in a specific shape and size so that the electromagnetic waves become resonant and build up inside the cavity. The result is that the charged particles passing through the cavity feel the overall force and direction of the resulting electromagnetic field, which transfers energy to push them forward along the accelerator. One important detail is that the field in an RF cavity is made to oscillate (change direction) at a given frequency, so timing the arrival of particles is critical.

Another critical element of the LHC is the beam dump system[29]. Beams do not circulate inside accelerators indefinitely. As particles collide with the sides of the beam pipe or with each other, the beams degrade becoming less likely to give collisions of interest. Accelerator physicists can choose to dump the beams, removing them from the accelerator and sending them to a beam dump (usually a radiation-shielded block deep underground) for a safe absorption. In order to understand the necessity of the beam dump system, the nominal LHC beam contains an unprecedented stored energy of 350 MJ, contained in 2808 bunches with a beam sigma of the order of 0.3 mm. The extremely high destructive power of such a beam imposes an external dump, where the beam must be extracted completely from the LHC, diluted to reduce the peak energy density and then absorbed in a dedicated system.

Last but not least is the LHC collimation system, Figure 1.3c[30]. The high luminosity performance of the LHC relies on storing, accelerating, and colliding beams with unprecedented intensities. The transverse energy density of the nominal beam is 1000 times higher than previously achieved in proton storage rings. Tiny fractions of the stored beam suffice to quench a super-conducting LHC magnet or even to destroy parts of the accelerators. Quenching occurs when any part of a superconducting cable goes from the superconducting to the normal resistive state; it occurs when the critical temperature, the critical current or the critical field is by-passed, or in case of movement of the superconductor by several mm (friction and heat dissipation), generating cooling failures or beam loss. The LHC collimation system[31] provides protection to the accelerator against unavoidable regular and irregular beam losses and achieves the necessary beam halo cleaning. The current system ensures that the peak losses below 0.01% of the energy lost from the beam is deposited in the cold magnets. The fully installed collimator consists of five main parts: (1) a base-support,

which serves as stand for the collimator in the tunnel, (2) an alignment plate, which allows the adjustment of the collimator and (3) a lower plugin, which provides cooling water and electrical connections for the collimator. The rest of the collimation system consists of (4) a vacuum tank with the two jaws and motors for moving the jaws and (5) all needed parts for the installation in the tunnel on a base-support assembly. The LHC collimation system is the largest and most advanced cleaning system ever built for a particle accelerator. It consists of 84 two-sided movable collimators of various designs and materials, which (together with injection protection collimators) translate into a total of 396 degrees-of-freedom. Since collimators actually interact with the beam, they become rapidly one of the most radioactive components in the LHC (after 2 years, a residual radioactivity of $50\mu Sv/h$ has been registered at 40 cm from the source).

1.2 Tele-Robotics at CERN

CERN's experiments involve colliding particles beams together or into stationary targets. When this happens, some of the particles release radiation. Radiation only occurs at CERN when the particle beam is on, and turning it off stops the emissions immediately. Radiation sometimes causes some of the components surrounding the collision points and the components interacting with the particles throughout the accelerators to become radioactive.

Since the LHC experiments are underground, this provides a natural barrier to radiation. CERN is very committed to ALARA (As Low As Reasonably Achievable) implying a job and dose planning for works in CERN's radiation areas aiming to minimize the dose to the workers. Ionizing radiation, along with oxygen deficiency and high magnetic fields, are the main hazards which maintenance personnel are exposed when working in the LHC tunnels or other experimental areas at CERN[32].

Remotely controlled mobile robots, able to carry out maintenance work and inspections, are nowadays considered in applications in hostile and hazardous environments in order to

reduce the human intervention. CERN has identified this need, and is in the process of developing several devices for remote inspection, radiation monitoring and machinery maintenance in order to minimize the personnel exposure to hazards. These remote operation robotic systems possess the autonomy capabilities for easing part of the workload by human operators conducting the remote interventions or inspections. Radiation surveys and visual inspections from a remote location are of great benefit to the Radio Protection personnel at CERN. Generally, electronics can take a radiation dosage of up to 100 Sievert (Sv)[33], which is 5000 times the annual dose allowed by the European Union, and 50000 times the dose allowed by CERN for personnel as an internal rule, making possible tele-robotics in radiation areas. From an analysis of ALARA interventions, about 50% of the dose to personnel was originated from very simple actions, such as visual inspections. By using robots, processes of measuring and inspecting can be automated, decreasing the radiation dose of the personnel and improving the quality of the obtained data. At the moment, the objectives of the robotic projects at CERN, include, radiation inspection, remote machine measurements, automated visual detection of anomalies in underground areas, and autonomous and tele-operated manipulation for machine maintenance.

1.2.1 Difficulties for Tele-Robotics and Inspections at CERN

There are several challenges to face in deploying tele-operated semi-autonomous systems in harsh environments, as those found at CERN. Examples of constraints at CERN, can be the huge spaces or distances that a robot may have to travel, like the 27 km of the LHC, the presence of unpredictable obstacles in the robot's path, Figure 1.4c, poor light conditions, Figure 1.4a, communication difficulties specially in the underground areas, unknown environments in areas which are not reachable by humans, Figure 1.4d, radiation and magnetic disturbances that may alter hardware component behavior, Figure 1.4b, etc.

CERN's environments and technology are not robot-designed. Most of the structures are more than 30 years old. Doors, steps, steep stairs or small corridors are found all around the structures, as it can be seen in Figure 1.4e and Figure 1.4f.

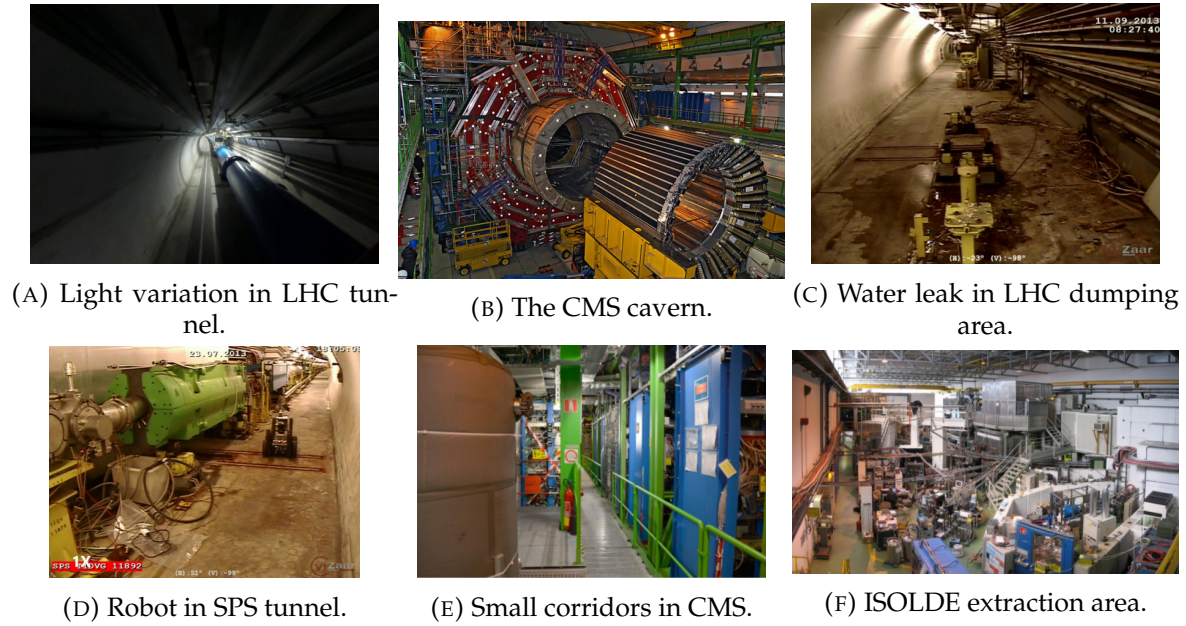


FIGURE 1.4: Examples of constraints at CERN.

The mentioned constraints represent big challenges in the design and implementation of robotic solutions. The entire robotic systems (i.e. platforms, arms, sensors, communication, etc.) must be robust and reliable in order not to harm itself and the surrounding equipment, usually very delicate and expensive.

1.3 CERNTAURO

The CERN Telematipulation semi-Autonomous Unit for Robot Operations, CERNTAURO. CERNTAURO is a software library which is being developed at CERN for management of fully teleoperated or semi-autonomous robots. It must allow telemanipulation for CERN hazardous environments, different robotic arm and robots under test/development and control running on-board and in real-time. Furthermore, we try to keep it “simple” and “efficient”, we are constantly developing gripping/manipulation tools and different force feedback sensors are used/under-test.

CERNTAURO has different modules, including a module for handling distinct sensors, another for communications and another for controlling some robot arms (Schunk Arm or

Youbot Arm) and mobile platforms (YoubotBase). Communications are based on sockets and it uses 3G and 4G communication technology, which leads to some problems: limited bandwidth, network shared with other CERN services and time delay not always predictable. Currently, a graphical user interface is being developed.

In addition, several applications are being developed for robots as visual tracking systems to different objects, people detection, collision avoidance, master-slave teleoperation systems with force feedback, etc.

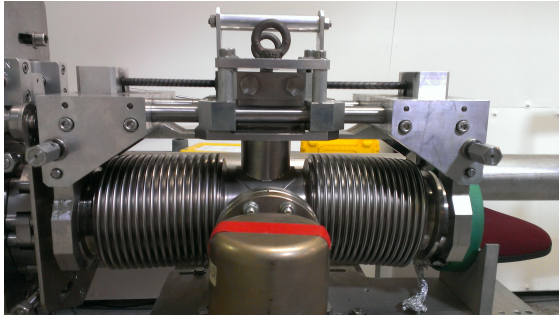
1.4 Description of the Problem and Objectives

The “mission” of tele-robotics at CERN may be resumed in the following: Ensuring **safety** of personnel improving **availability** of CERN’s accelerators. The robots which are being developed at CERN should offer visual capacities, among them the capacity to estimate the 6D pose of an object. To know the position and rotation of an object could be used in augmented reality for a better information to the tele-operator and for the realization of semi-autonomous tasks.

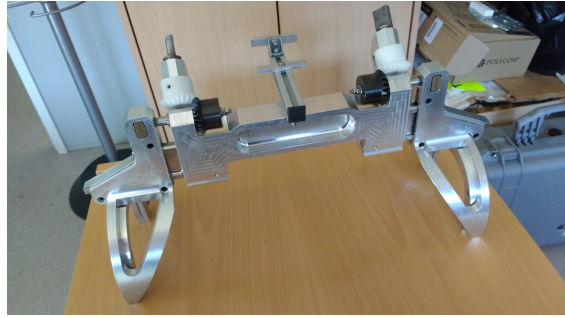
The goal of this project is to develop a novel and reliable algorithms to estimate the 6D pose, using computer vision, of objects that are going to be telemanipulated by robots. The novel algorithms should work consistently in the CERN unstructured and harsh environments presenting several constraints like variable luminosity, difficult accessibility and light reflections.

This project will focus on the detection and position of objects related to collimators. These collimators are objects that have vital importance in inspections in the tunnel of the LHC, since they are in command to keep the beam stable, and are hot points of radiation.

To demonstrate that the algorithm works, we will work with a collimator piece, Figure 1.5, whose job is to dock or detach the collimator from the tube through which the LHC



(A) A type of separator.



(B) Another type of separator.

FIGURE 1.5: Two models of separator.

beam passes. Once the algorithm shows it works, we will work to detect the position and rotation of the entire collimator, Figure 1.6. The algorithm must be robust to changes of lighting and position of the camera within a certain range.



FIGURE 1.6: Photograph of an installed ring collimator on its quick plug-in support in IR8 of the LHC[30].

Chapter 2

State of the Art

2.1 Problem Formulation

2.2 Introduction to Computer Vision

The objective of this project is to know the position and orientation of a desired object, for that it is necessary to use computer vision systems and vision algorithms.

Computer vision is a branch of artificial intelligence and image processing concerned with computer processing of images from real world. Computer vision typically requires a combination of low level image processing to enhance image quality (e.g. remove noise, increase contrast) and higher-level pattern recognition and image understanding to recognize features present in the image¹.

On computer vision, we are trying to describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions[34]. Today computer vision is being used in a wide variety of real-world applications, such as, optical character recognition (OCR), 3D model building (photogrammetry), motion capture, face detection, object recognition, etc.

¹Definition from Free On-Line Dictionary Of Computing, <http://foldoc.org/Computer%20Vision>

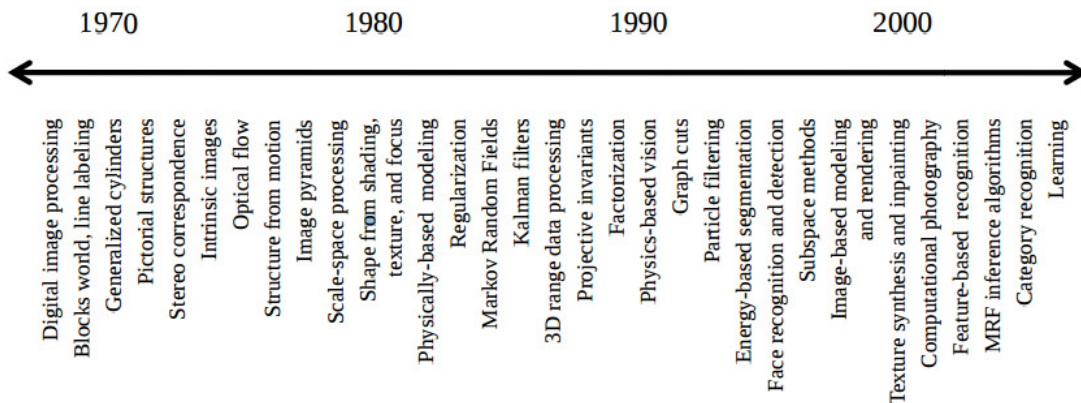


FIGURE 2.1: Timeline with some active topics of research in computer vision[34].

In the last years, computer vision has advanced quickly thanks to deep learning techniques. These algorithms are based on neural networks with multiples hidden-layers and hundreds of thousands, even millions, of artificial neurons. The pioneering article[35] was in context of the ImageNet LSVRC-2010 contest[36]. They used a deep convolutional neural network to classify 1.2 million high-resolution images into 1000 different classes. Nowadays, computer vision using deep learning outperforms to humans in image recognition, Figure 2.2.

2.3 Computer Vision Systems

Computer Vision Systems are used to capture images or videos. Most computer vision systems rely on image sensors, which detect electromagnetic radiation, which is typically in the form of either visible or infra-red light.

In this project the data retrieve for the computer vision systems can be separate in:

RGB image has three channels: red, green, and blue. RGB channels roughly follow the color receptors in the human eye, and are used on computer displays and image scanners. It is the typical image that one can get with a normal digital camera, Figure 2.3a.

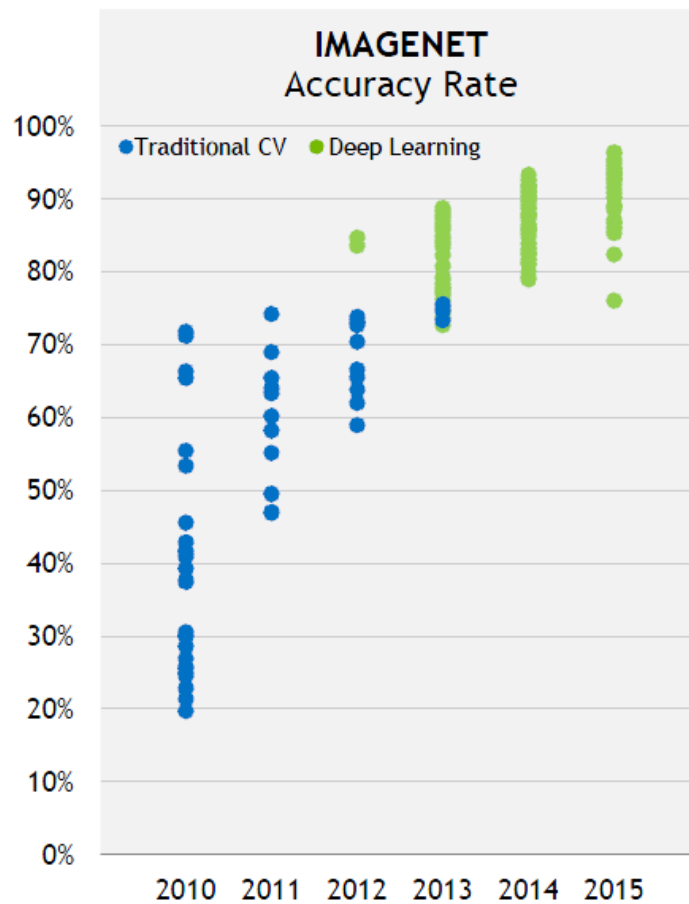
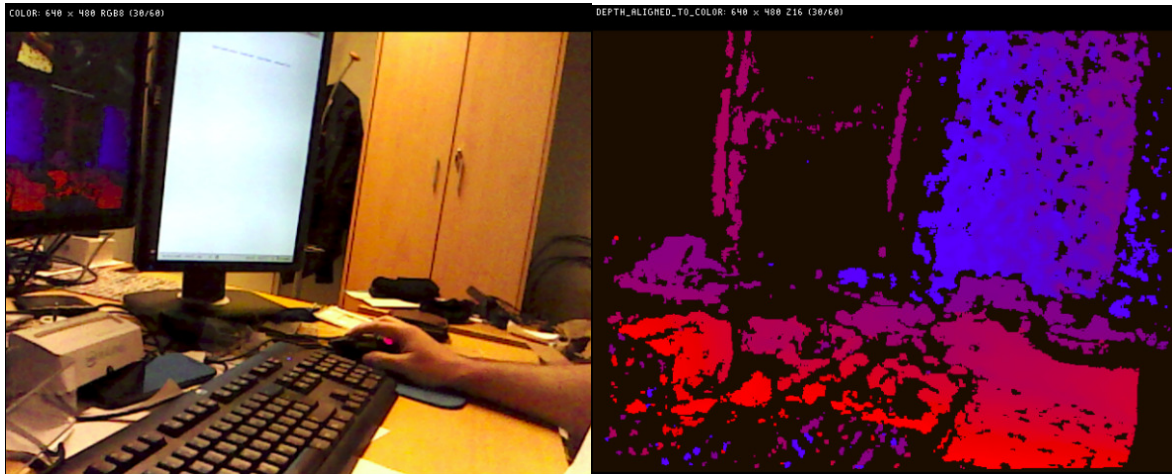


FIGURE 2.2: Evolution of Imagenet Error Rate. Source: [Nervana](#).

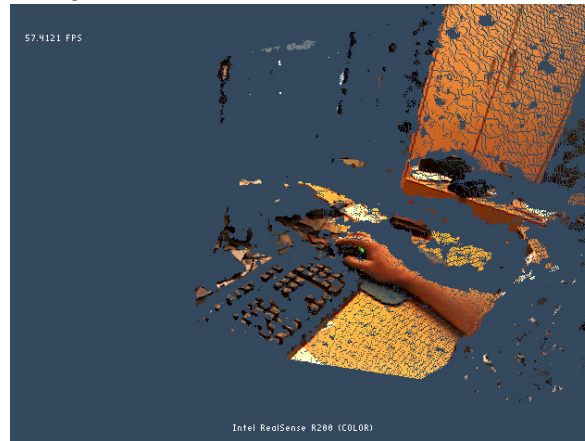
Depth field is gotten in different ways, for example, with a LiDAR one gets a point cloud from the orientation of the laser, and the measured distance or one gets an image in which each pixel has a depth value, this happens with cameras as Kinect, Figure 2.3b.

RGB-D is the match between a RGB image, and its correspondent depth field. Normally is a representation in the format of a point cloud with color, Figure 2.3c.



(A) RGB image.

(B) Depth image.



(C) Point cloud with color.

FIGURE 2.3: Data from a Intel RealSense camera.

The computer vision systems used for this project for 3D reconstruction can be divided into:

- **Time of Flight** camera (ToF camera) is a range imaging camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point on the image. *Kinect V2* and a LiDAR are examples of ToF cameras.
- **Stereo** camera is based on the disparity in the image of two or more monocular cameras.
- **Structured light projection** camera uses projected light patterns which is reflected back from the scene and is detected by the sensor. Using triangulation, is possible to

reconstruct a 3D scene.

2.3.1 Monocular Camera

The main device used in computer vision is a monocular camera. This kind of camera captures RGB Image and it can use different light sources, but it doesn't retrieve a depth image. In this project, I could use an IDS UI-2220SE-C-HQ Industrial Camera, Figure 2.4.



FIGURE 2.4: IDS UI-2220SE-C-HQ Industrial Camera.

The specification of the sensor of this camera is reference in the Table 2.1.

Sensor Technology	CCD
Manufacturer	Sony
Resolution (h x v)	768×576
Color depth (sensor)	12 bit
Color depth (camera)	12 bit
Pixel Class	PAL
Sensor Size	1/2"
Shutter	Global shutter
max. fps in Freerun Mode	71.0
Binning Modes	Mono
Subsampling Modes	-
Sensor Model	ICX415AQ
Pixel size	$8.3 \mu\text{m}$
Optical Size	$4.781 \text{ mm} \times 6.374 \text{ mm}$

TABLE 2.1: Sensor Specification of an IDS UI-2220SE-C-HQ Industrial Camera[37].

The main problem with this camera is that it isn't integrated in OpenCV. It's necessary to use the IDS SDK and retrieve the data image and then transform the data into a matrix of

OpenCV. For this purpose was created the class IDSCamera.

LISTING 2.1: IDSCamera.h

```
#include "ueye.h"
#include <opencv2/opencv.hpp>
#include <iostream>

class IDSCamera {

public:
    IDSCamera(int width, int height, int camNumber);
    ~IDSCamera();

    void GetFrame(cv::Mat& frame);

private:
    HIDS hCam;
    int pWidth, pHeight;
    char* m_pcImageMemory;
};
```

Another problem is that the available optics produce very deformed images, Figure 2.5. Also in this project was tested a common high-definition webcam *Logitech HD Pro C920*.

There is an algorithm for pose estimation for monocular cameras in OpenCV library[38] but it is only for textured objects and targets in this project are mainly texture-less, then it was rejected.

A technique that was used is the **photogrammetry**, Figure 2.7. It creates a 3D model from a set of pictures, Figure 2.6. For this technique was used the application *Autodesk 123D Catch*, that allows to capture places, people and things in 3D using your Windows Phone or Mobile device, iPhone, iPad, Android device, or any camera[39]. In this task was used a

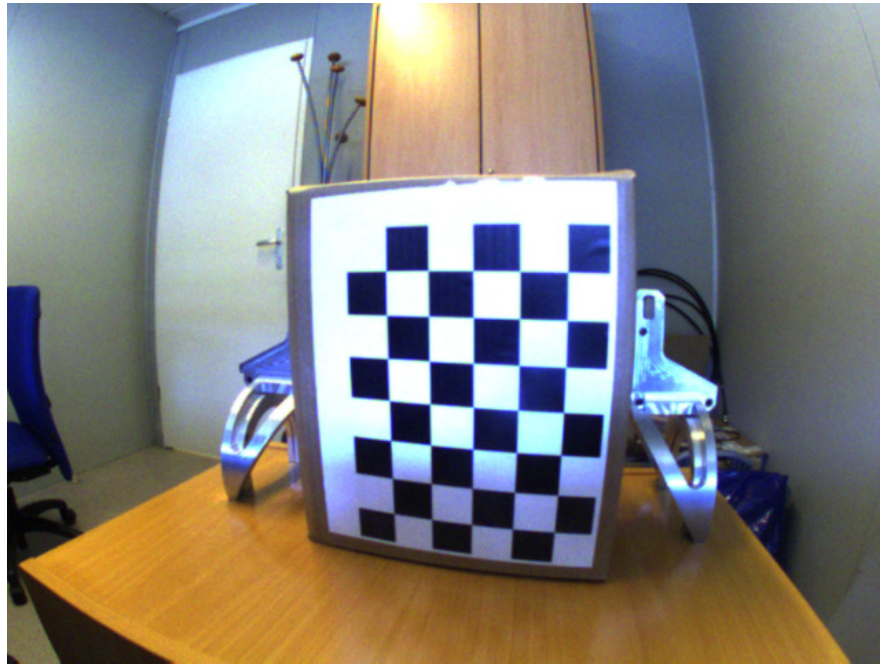


FIGURE 2.5: Deformed image produced by optics.

normal Android Phone with the app 123D Catch.



FIGURE 2.6: Set of images to do a photogrammetry.

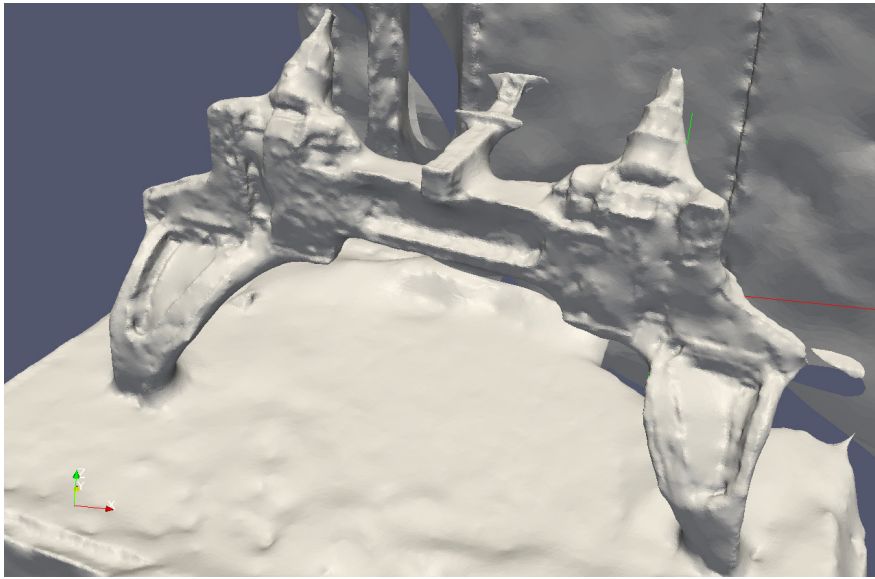


FIGURE 2.7: Photogrammetry of one of the separators.

One advantage of this technique is that only uses a monocular camera, but it is not real time and it does not provide the current position of the camera. Then for the purpose of this project is useless, and for this we have already the CAD models.

2.3.2 Stereo Camera

The main problem of a monocular camera is that it doesn't retrieve depth data. However, it can use a **stereo camera** to get the depth data of the scene. As in the laboratory there wasn't a stereo camera, it was create one with two IDS Cameras, Figure 2.8.

The main problem of this approach is that to be useful it has to do a very good calibration. If not, there is a lot of noise in the depth field. It was tried a calibration of each camera, but the results were not very useful. It was studied bought a commercial stereo camera between the options were:

- Ensenso N10 Stereo 3D camera
- MultiSense S7
- Duo 3D MLX



FIGURE 2.8: Stereo camera with two IDS camera

- Capella

But they don't provide better depth field than a ToF camera or a Structured Light camera.

2.3.3 LiDAR

LiDAR, Light Detection And Ranging, is a surveying method that measures distance to a target by illuminating that target with a laser light. Lidar is popularly used to make high-resolution maps, with applications in geodesy, geomatics, archaeology, geography, geology, geomorphology, seismology, forestry, atmospheric physics, laser guidance, airborne laser swath mapping (ALSM), and laser altimetry[40].

In this project, I used a *Velodyne HDL-32E*2.9. It's often used in autonomous navigation, because it has many advantages[41] as:

- High Range. 100 m.
- 700.000 Points per Second
- 360° Horizontal FOV.
- $\pm 20^\circ$ Vertical FOV.



FIGURE 2.9: LiDAR Velodyne HDL-32E.

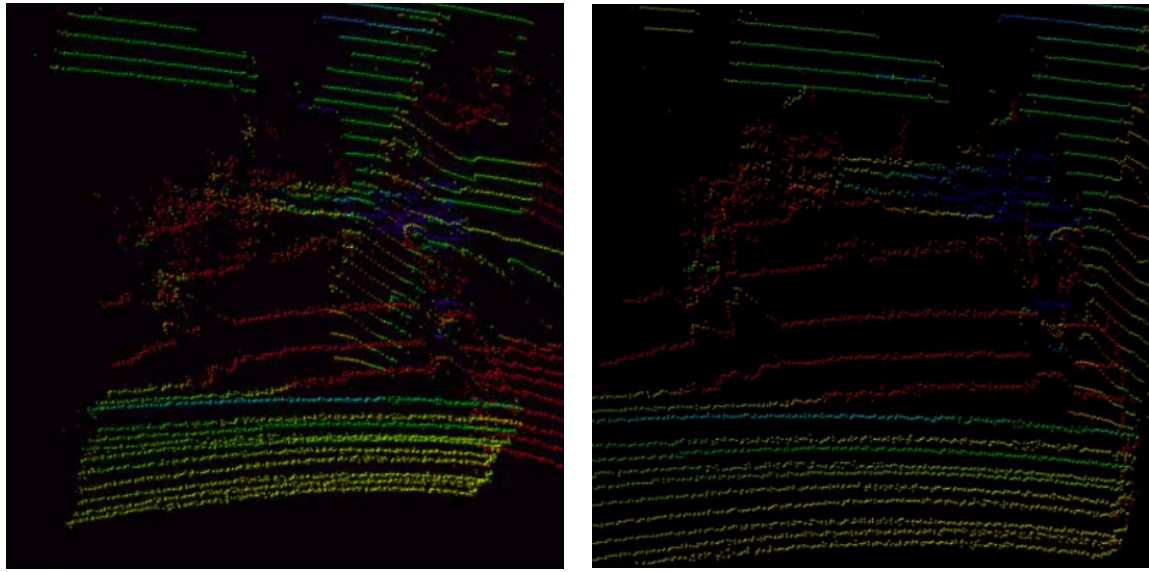
- Low Power Consumption

But for our purpose it presents many drawbacks:

- Low accuracy. ± 2 cm.
- Minimum return distance is approximately 1 meter
- Low resolution of object of interest.
- Reflections with metal or polished surfaces.

These problems are shown in the Figure 2.10. Because the generated point cloud extends into 360° there is a low density in a small target and the reflections with metal cause problems in the point cloud.

A solution for the low density of points is to use another LiDAR as *Ocular Robotics' Robot-Eye RE05*[42], this LiDAR allows to define the desired scan region, vertical and horizontal data resolution. But, in this project, I couldn't use this sensor.



(A) A view of the object of interest

(B) Another view of the object of interest.

FIGURE 2.10: We can see two main problems: a low density of points in the target and a different reflectance in the same material.

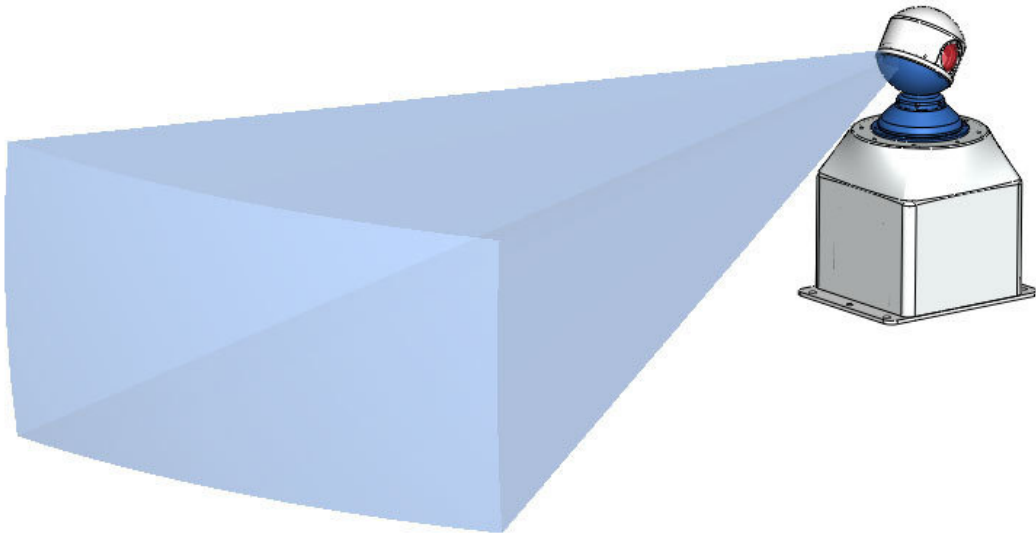


FIGURE 2.11: Mode Region Scanning of *RobotEye RE05*. This mode allows the user to define a rectangular region within the scanner's azimuth and elevation field in which to concentrate the scanning[42].

2.3.4 RGB-D Cameras

In this project RGB-D cameras are those based on **Structured Light** and on **Time of Flight**.

Structured Light is the process of projecting a known pattern (often grids or horizontal bars) on to a scene. The way that these deform when striking surfaces allows vision systems to calculate the depth and surface information of the objects in the scene, as used in structured light 3D scanners[43].

Time of Flight camera is a range imaging camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point of the image[44].

A big problem with these cameras is the presence of reflections with metal objects; this produces holes in the depth field. These holes appear mostly in parts in front of the camera, Figure 2.12. This effect cannot be avoided using other external sources of IR light, the camera has to use its own source of light.

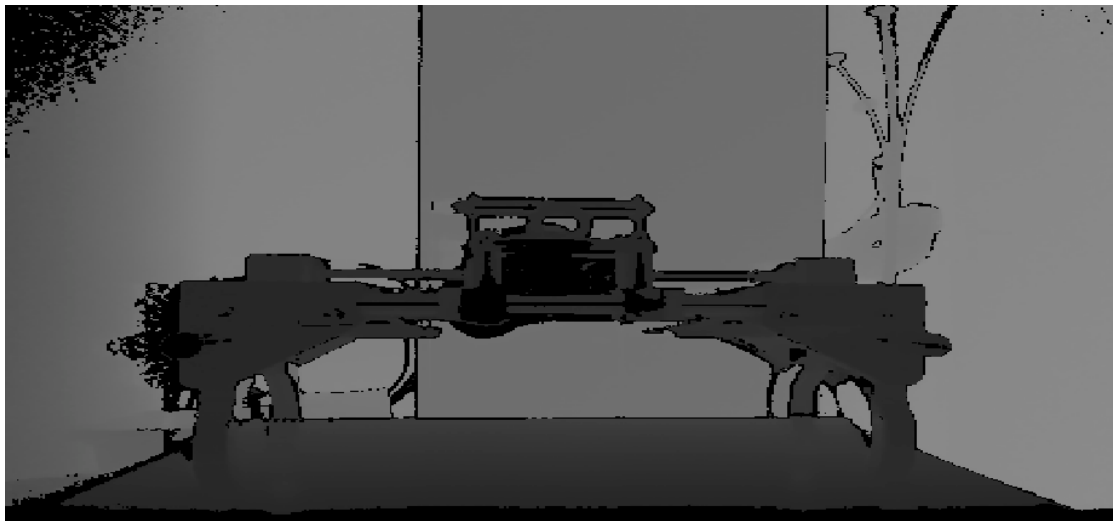


FIGURE 2.12: The black zones are null data in the depth field caused by reflections of the light of the projector.

To avoid this problem was tried to create a 3D model of the target using programs as *Kinfu*[45] (in Linux platforms) and *Kinect Fusion*[46] (in Windows platforms). The result was very bad, Figure 2.13. Because the camera lost the tracking very fast due to the appearance

and disappearance of keypoints when the camera is moved.

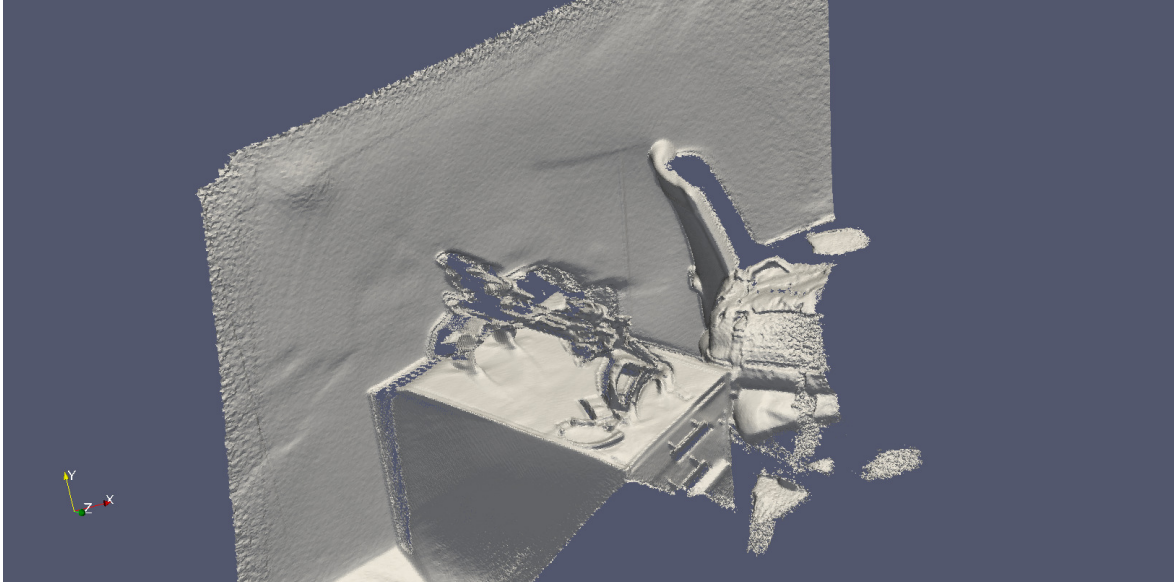


FIGURE 2.13: 3D reconstruction of the target done in real time with camera tracking by using *Kinect Fusion*.

There are other *Kinect Fusion* like algorithms as *ElasticFusion*[47].

Microsoft Kinect V2

Microsoft Kinect V2 is a time-of-flight sensor developed by Microsoft, replacing the older technology from *PrimeSense*. This camera was developed for videogames in the *XBOX One* console.

Depth CMOS:	512×424 , 30 Hz
FOV:	70×60
Range:	0.5 – 4.5 meters
Color CMOS:	1080p, 30 Hz (15 Hz in low light)
Dimensions:	$24.9\text{ cm} \times 6.6\text{ cm} \times 6.7\text{ cm}$
Data Interface:	USB 3.0

TABLE 2.2: Main specifications of Microsoft Kinect 2[48].



FIGURE 2.14: Microsoft Kinect V2[48].

As seen in the Table 2.2, the main problem with this sensor for the purpose of this project, are the dimensions of the sensor. It is too big to put it on the tool of an arm robot. A good point is that it gets a dense point cloud of the object of interest, but with a high noise, Figure 2.15.

To use Kinect V2 in Linux is mandatory to use the *libfreenect2*[49] library.

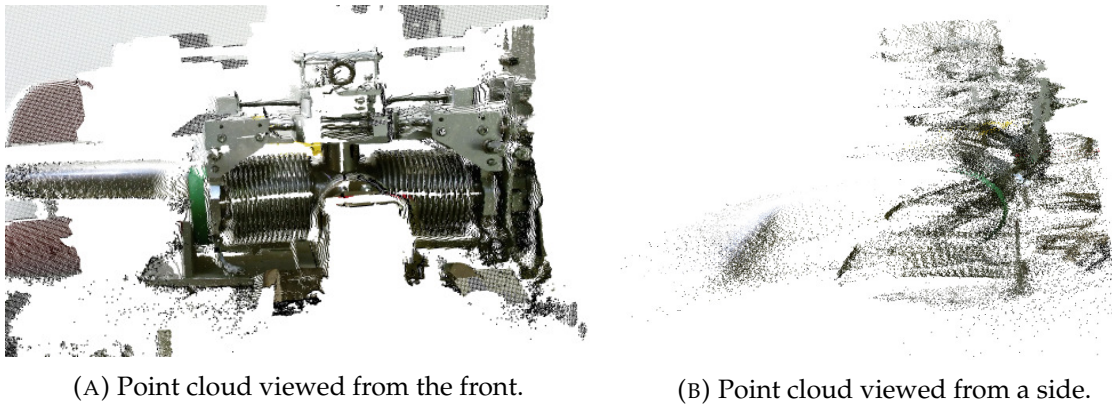


FIGURE 2.15: Point cloud of the object of interest from a Kinect V2. It has a dense point cloud, but with a high noise.

Even with these drawbacks, this camera was used in the first tests of concept of the algorithm developed in this project.

Intel RealSense

Intel RealSense is a RGB-D camera developed by Intel for applications of augmented reality and head and hand tracking. The model *R200* and *SR300* were used in this project, Figure 2.16.

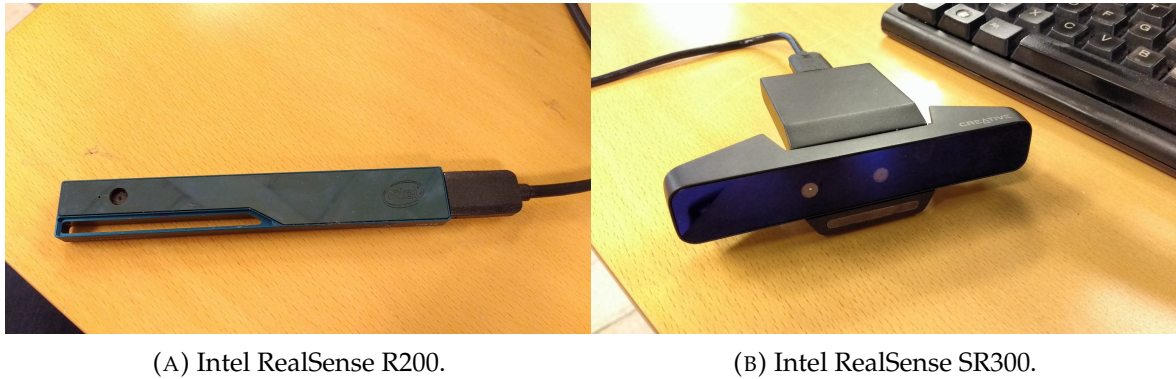


FIGURE 2.16: Intel RealSense cameras used in this project.

R200 The Intel RealSense Camera R200 is a module that implements a long range, stereo-vision 3D imaging system. The small size of the R200 module provides system integrators flexibility to design into a wide range of products.

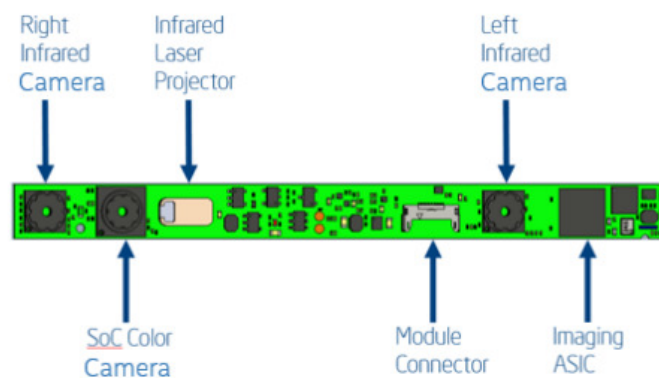


FIGURE 2.17: Diagram of Intel RealSense R200. It has a color camera, 2 IR cameras and an IR laser projector[50].

Depth Range:	from 0.4 to 2.8m
Dimensions:	101.56mm length × 9.55mm height × 3.8mm width.
Color Sensor:	Full HD RGB.
Data Interface:	USB 2.0

TABLE 2.3: Some features of Intel RealSense R200[50].

SR300 The Intel® RealSense™ Camera SR300 is a subassembly product that implements a short range, coded light 3D imaging system. The small size of the SR300 subassembly provides system integrators flexibility to design into a wide range of products.

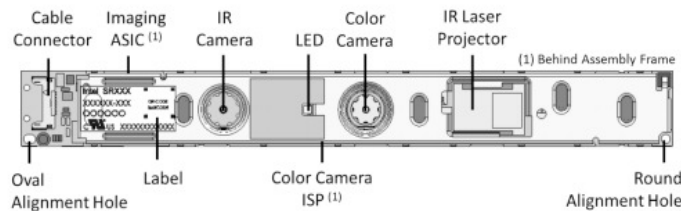


FIGURE 2.18: Diagram of Intel RealSense SR300. It has a color camera, an IR camera and an IR laser projector[51].

Depth Range:	from 0.2 to 1.5m
Dimensions:	101.56mm length × 9.55mm height × 3.8mm width.
Color Sensor:	Full HD RGB.
Data Interface:	USB 3.0

TABLE 2.4: Some features of Intel RealSense SR300[51].

The main difference between both systems is the **depth range**, both are suitable to work in the LHC tunnel. A sensor fusion could be done using both sensors and get a point cloud with a depth range from 0.2 meters to 2.5 meters. There is no problem to put them on the tool of the arm robot because they have a small size.

Orbbec Astra Pro

Orbbec Astra Pro computes depth data by using structured light technology, which recovers the 3D positions of points on an object by projecting a pattern with known features on it then performing 3D reconstruction on image correlation and triangulation bias. Orbbec Astra

Pro can easily output high-accuracy VGA (640 × 480) raw depth stream without heavy post processing such as interpolation[52].

Dimensions:	160 × 30 × 40 <i>mm</i>
Weight:	0.3 <i>kg</i>
Range:	0.4 ~ 8 <i>m</i>
Depth Image Size:	640 × 480 (VGA) 16bit @ 30FPS
RGB Image Size:	1280 × 720 @ 30FPS (UVC Support)
Field of View:	60° horiz × 49.5° vert. (73° diagonal)
Data Interface:	USB 2.0

TABLE 2.5: Orbbec Astra Pro technical specs[53].

In the Figure 2.19, it can be seen that the depth field has less noise than the depth field retrieved by Kinect, 2.15. And it retrieves a point cloud with a better separation between zones of the scenario. A drawback is the loss of certain zones with respect the Kinect, but it detects perfectly the flat zones. This is important for this project, because the objects of interest have a lot of flat parts.

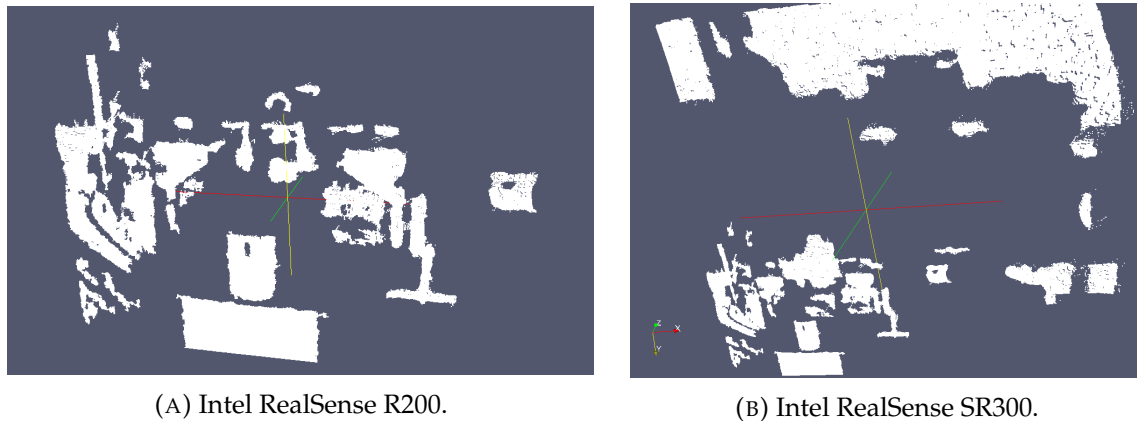


FIGURE 2.19: Intel RealSense cameras used in this project.

Another drawback with this camera is that cannot retrieve at the same time the depth field and a color image. But, how in this project the objects of interest are texture-less objects, this is not a problem.

2.4 Computer Vision Frameworks

To work with computer vision several frameworks exist, these frameworks have many applications and functions that allow us to develop computer vision applications in a simpler way. The most famous is **OpenCV**, but it hasn't functions to work with point clouds, for this purpose exists **Point Cloud Library**.

2.4.1 OpenCV

OpenCV, Open source Computer Vision, is a computer vision library, which initially was developed by Intel[54]. Currently, it has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. In the last version it supports CUDA computing[55].

OpenCV has functions to retrieve the image of many types of webcams in its own format. This makes easy builds applications with home cameras. There are many algorithms to work on images, such as feature detectors, tracking algorithms, calibration algorithms, object detection algorithms, etc.

2.4.2 Point Cloud Library

The Point Cloud Library[56] (PCL) is an open-source library of algorithms to point cloud processing tasks and 3D geometry processing, such as occur in three-dimensional computer vision. The library contains algorithms for feature estimation, surface reconstruction, registration, model fitting, and segmentation. It is written in C++ and released under the BSD license[57].

PCL has grabbers for several devices from LIDAR's to RGBD Cameras, these grabbers convert the depth image and color image retrieve from these devices to point cloud data. Also has many algorithms of clustering, features detector, object recognition, segmentation, etc.

PCL Algorithms

Within PCL there are several tutorials of different algorithms to perform matching of two point clouds: *Iterative Closest Point*[58][59], *Implicit Shape Model*[60][61], *3D Object Recognition based on Correspondence Grouping*[62] and *Hypothesis Verification for 3D Object Recognition*[63]. These algorithms were tested with negatives results, it was tested to match parts of the target of a point cloud with another point cloud of the same target object.

2.5 Algorithms

There were several state-of-the-art algorithms to get the pose position of the target that were tested:

- LSD-SLAM
- LINE-MOD
- PWP3D
- Sliding Shapes

2.5.1 LSD-SLAM

LSD-SLAM[64][65], Large-Scale Direct Monocular SLAM, is a novel direct monocular SLAM technique: Instead of using keypoints, it directly operates on image intensities, both for tracking and mapping. The camera is tracked using direct image alignment, while geometry is estimated in the form of semi-dense depth maps, obtained by filtering over many

pixelwise stereo comparisons. LSD-SLAM runs in real-time on a CPU, and even on a modern smartphone.

By using direct visual odometry methods by optimizing the geometry directly on the image intensities, this algorithm can extract a point cloud of texture-less objects, as methods based on features detection fails with this kind of objects.

The algorithm consists of three major components: tracking, depth map estimation and map optimization, Figure 2.20.

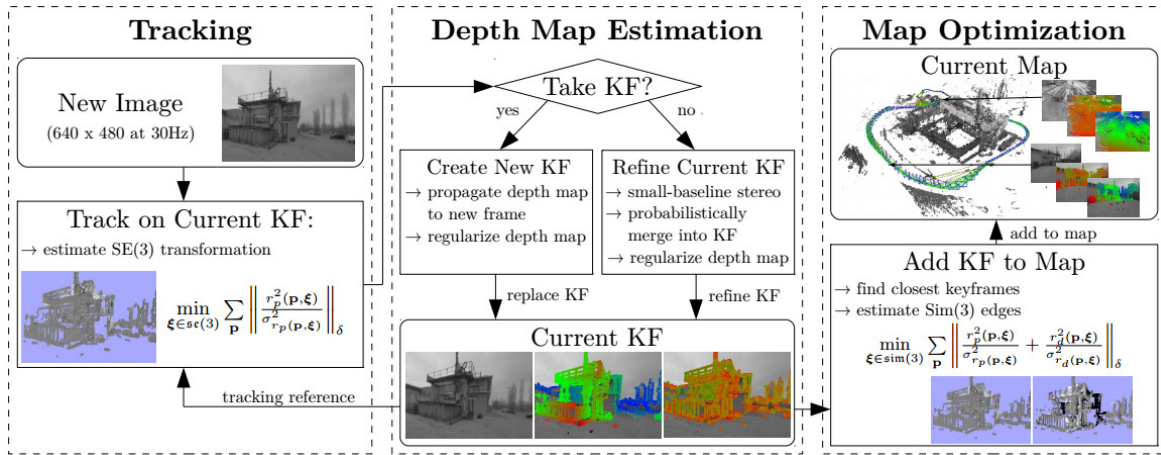


FIGURE 2.20: Overview over the complete LSD-SLAM algorithm[64].

- The **tracking** component continuously tracks new camera images.
- The **depth map estimation** component uses tracked frames to either refine or replace the current keyframe.
- Once a keyframe is replaced as tracking reference, it is incorporated into the global map by the **map optimization** component.

This algorithm can use a monocular camera, then it doesn't depend in a special camera to get the Depth Field. It can get directly a point cloud of the object of interest with the movement of a camera. In the test of this algorithm in this project was used a *Logitech C920 HD Pro*, Figure 2.21, and it was necessary to do a very good calibration, because in the first

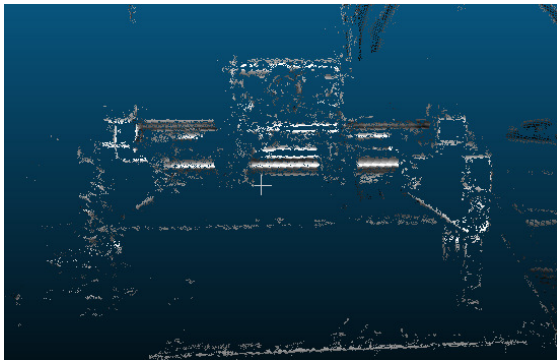
tests, the tracking was lost too quickly, and it creates a very bad point cloud. The official implementation of LSD-SLAM[66] in ROS[67] was used in the tests.



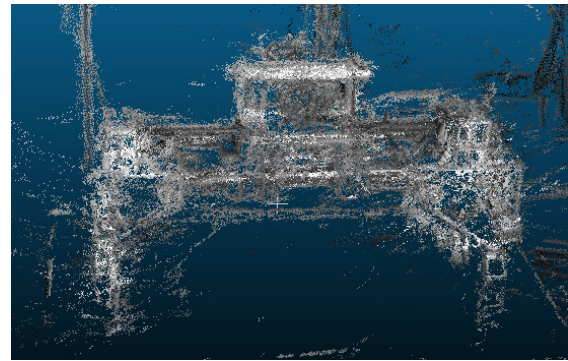
FIGURE 2.21: Logitech C920 HD Pro.

LSD-SLAM has important drawbacks with the necessary keyframes to create the point cloud:

- To get a small point cloud it needs six or more different keyframes, Figure 2.22a.
- The more frames, the more noises the point cloud has, Figure 2.22b.



(A) Point cloud created with 6 keyframes using LSD-SLAM.



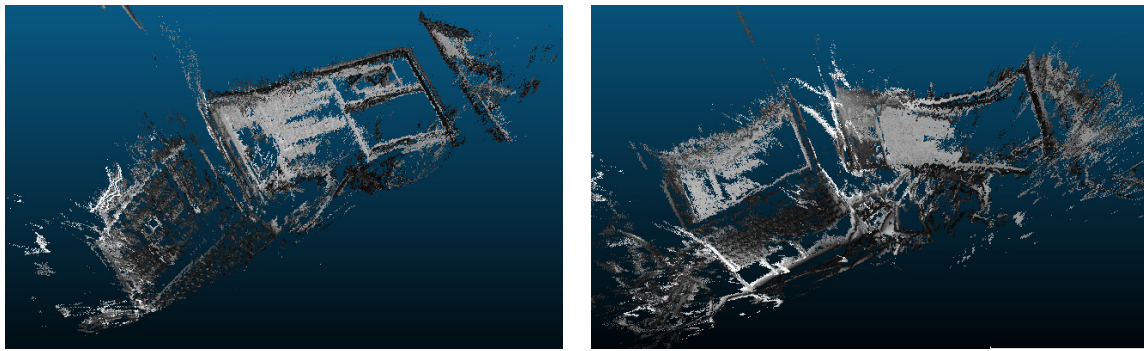
(B) Point cloud created with more than 6 keyframes using LSD-SLAM.

FIGURE 2.22: Point clouds of the object of interest using LSD-SLAM. The image on the left shows reflections.

Moreover, it has a problem with the tracking due to reflections, Figure 2.22a. The reflected light follows the camera, LSD-SLAM detects the reflections, as features and then the tracking starts to fail. If the program doesn't finish the point cloud appears very deformed.

A polarized filter was used to try to remove reflection, but it didn't work.

Sometimes the reconstruction of the scenario doesn't match with the reality and it produces deformations in the objects that are trying to reconstruct, Figure 2.23.



(A) A good point cloud of a computer.

(B) A deformed point cloud of a computer.

FIGURE 2.23: Point clouds of a computer created using LSD-SLAM.

With these important drawbacks, the point cloud created is **unreliable**. In the development of this project a new visual slam algorithm was developed by the same group of LSD-SLAM, this new algorithm is *DSO: Direct Sparse Odometry*[68][69], but it wasn't tested in this project.

Possible solutions to these drawbacks are:

- Make rectilinear camera movements. It got noiseless point clouds, but it limits too much as the size of the point cloud. And avoid no pure rotations.
- Take the fewer possible pictures.

2.5.2 LINE-MOD

LINEMOD[70] is a method for detecting 3D objects using multi-modalities. It works in real-time, under heavy clutter, does not require a time-consuming training stage, and can handle untextured objects, Figure 2.24. It is based on an efficient representation of templates which capture the different modalities.

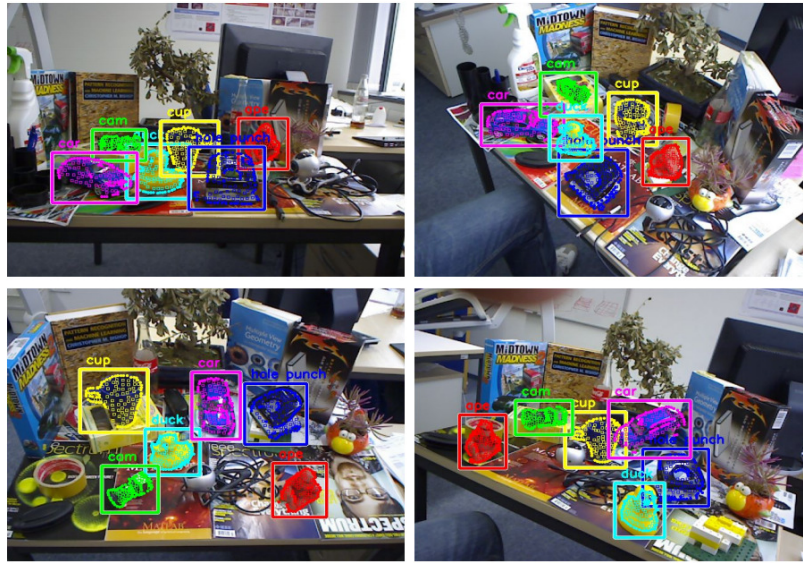


FIGURE 2.24: Texture-less 3D objects detected in real-time under different poses over heavily cluttered background using an image and its depth map[70]

This algorithm uses a multimodal template, combining color and depth information to improve detection of texture-less objects and to improve handling of cluttered background. It has an efficient implementation that enables real-time performance by quantizing and spreading the features values, precomputing response maps and linearizing the memory. For the templates LINEMOD uses 3D CAD models[71].

In this project was used the implementation of LINEMOD of *Willow Garage* using ROS[72]. The implementation is a pipeline that manages objects using *couchDB*[73] database. Thus, in order to learn objects, it needs to store the 3D models in the database first.

When the objects are stored in the database, the pipeline has two more steps:

- **Pre-processing step**, to learn object models from the database, the algorithm generates all the templates, it uses an automatic view generator. Those vary the in-plane rotations of the cameras and deals with the different scales and view points. Thousands of images with depth+mask are generated and fed to the OpenCV trainer.
- **Detection**, once learned, objects can be detected from the input point cloud.

In the tests, it was used a Kinect V2 camera using the ROS package *IAI Kinect2*[74]. The goal of this package is to give a driver, and the tools needed to receive data from the Kinect-2 sensor, in a way useful for robotics. It contains:

- **A calibration tool** for calibrating the IR sensor of the Kinect One to the RGB sensor and the depth measurements
- **A library for depth registration** with OpenCL support
- **The bridge between libfreenect2 and ROS**
- **A viewer** for the images / point clouds

The example of the tutorial of the LINEMOD pipeline in the first test was unsuccessful. The algorithm didn't detect the can of the example, this happened because the Kinect wasn't calibrated. To make this algorithm work is mandatory a very good calibration of the Kinect. For this, it is necessary to calibrate the RGB camera, the Depth camera, and the synchronization between the two cameras, and both have different resolution.

After the calibration the can was detected and got its position and rotation very well, but it was very **unreliable against occlusions even small ones**. If a person catches the can with their fingers the program doesn't detect it.

When it was tried with pieces in the collimator because of the reflections, the algorithm detected these as occlusions. Then the algorithm is unable to detect pieces of the object of interest.

The *LineMOD* algorithm as well as *tod*[75] and *tabletop*[76] proved unusable for this task.

2.5.3 PWP3D

PWP3D[77] is a probabilistic framework for simultaneous 2D segmentation and 2D–3D pose tracking, using a known 3D model (of arbitrary shape) of the segmented object. This technique is region-based; at each frame it maximizes the discrimination between statistical foreground and background models, by adjusting the pose parameters iteratively. It uses posterior membership probabilities for foreground and background pixels, rather than pixel likelihoods, and during periods of stable tracking, it allows adaptation of the statistical foreground and background models.



FIGURE 2.25: Example of PWP3D detecting a soda can with motion blur[77].

When it was analyzed, the code it was found that it needs a template frame of the piece to start to find the position of the piece. This template is difficult to get. And it wasn't found any method to get it. So this algorithm was discarded.

2.5.4 Sliding Shapes

Sliding Shapes[78] uses depth maps for object detection and designs a 3D detector to overcome the major difficulties for recognition, namely the variations of texture, illumination, shape, viewpoint, clutter, occlusion, self-occlusion and sensor noises. The algorithm takes a collection of 3D CAD models and renders each CAD model from hundreds of viewpoints to obtain synthetic depth maps. For each depth rendering, the algorithm extracts features from the 3D point cloud and trains a Support Vector Machine (SVM) classifier. Finally, the

algorithm slides a 3D detection window in 3D space.

During training the algorithm learns an ensemble of linear Exemplar-SVM classifiers, each of which is trained with a rendered depth map from CG model as the single positive and many negatives from labeled depth maps. During testing, it takes a depth image with the gravity direction as input. The learned SVMs are used to classify a sliding window in 3D and output 3D bounding boxes with detection scores. Four types of 3D features have been designed, and several methods have been proposed to handle clutter, occlusion and missing depth.

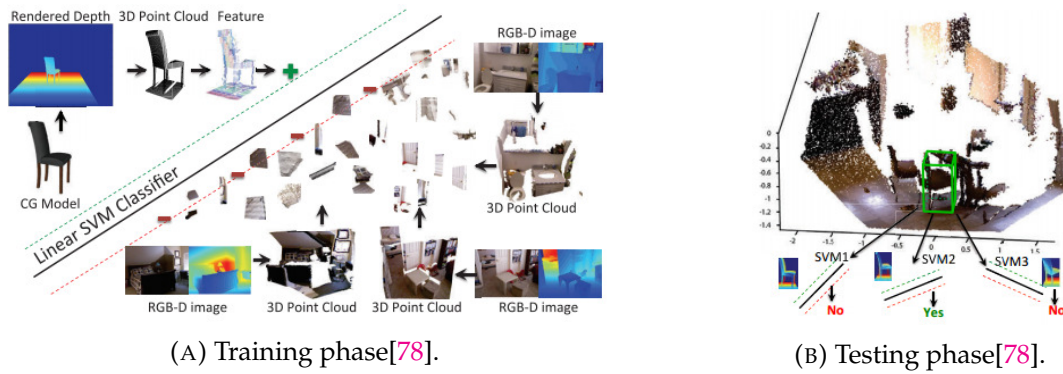


FIGURE 2.26: Graphical explanation of *Sliding Shapes*.

This algorithm was really promising. But, it was unable to make it works the code of *Sliding Shapes*. In the demo, the render throws an error, and too much time was spent trying to find the cause. The program it's in Matlab and the bad function is in a mex file (c++ for Matlab), and from Matlab it is difficult to debug that. Another problem with *Sliding Shapes* is that it is a 3D object detector that runs sliding windows in 3D to classify directly each 3D window. However, the algorithm uses handcrafted features, and the algorithm uses many exemplar classifiers so it is very slow. With these drawbacks the algorithm was discarded.

2.6 Deep Learning Algorithms

Modern deep learning provides a very powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions

of increasing complexity. Most tasks that consist of mapping an input vector to an output vector, and that are easy for a person to do rapidly, can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples[79].

Deep learning is a field in which appears many daily publications and it is hard to stay updated. But, I found good tools and services to stay updated:

Arxiv Sanity, <http://www.arxiv-sanity.com/> This service, that was created by Andrej Karpathy², lets you search and sort through 20,000+ arXiv papers on Machine Learning over the last three years in the same pretty format.

GitXiv, <http://www.gitxiv.com/> GitXiv is a space to share collaborative open computer science projects. GitXiv offers a collaboratively curated feed of projects. Each project is conveniently presented as arXiv + Github + Links + Discussion.

Twitter, <https://twitter.com> Twitter is a social networking service. In this social network is easy to follow experts in Deep Learning. They share their new publications and comment between them new interesting papers and conferences, and it is easy "tweet" with them. So this social network is an excellent tool to stay update of advance in the field.

2.6.1 Paper of Amazon Picking Challenge

The most interesting paper[80] related with my project that uses deep learning is in the context of *Amazon Picking Challenge*[81]. The challenge of this competition is to develop a pick-and-place that recognizes and locates objects amid cluttered environments, self-occlusions, sensor noise, and a large variety of objects. The authors present an approach in which they

²Andrej Karpathy is an important research scientist working on Deep Learning. <http://cs.stanford.edu/people/karpathy/>

segment and label multiple views of a scene with a fully convolutional neural network, and then fit pre-scanned 3D object models to the resulting segmentation to get the 6D object pose.

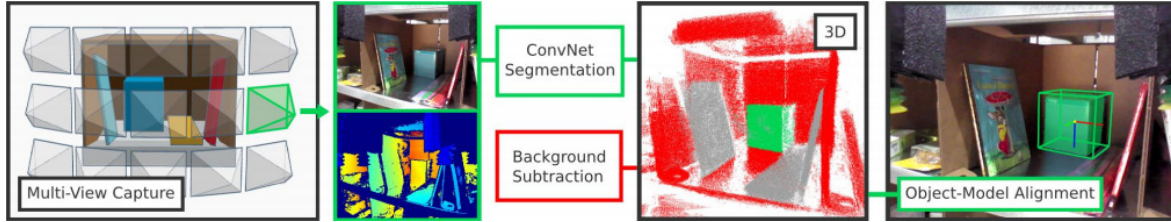


FIGURE 2.27: System purpose in [80].

The proposed algorithm, Figure 2.27, to estimate the 6D pose is divided in two phases. First, it segments the RGB-D point clouds captured from multiple views into different objects using a deep convolutional neural network. Second, it aligns pre-scanned 3D models of the identified objects to the segmented point clouds to estimate the 6D pose of each object. To make the 3D model-fitting, they use the iterative closest point algorithm.

But they have problems handling objects with missing depth, which is the same problem I have in my project with metallic objects. Then the point cloud is noisy and sparse, and their pose estimation algorithm performs poorly.

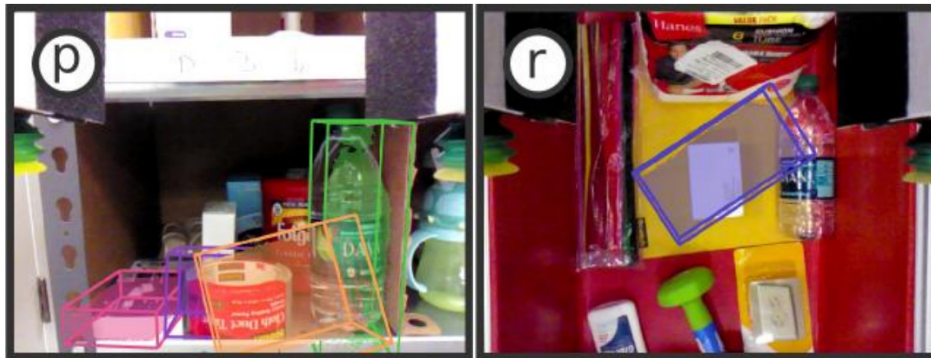


FIGURE 2.28: Problems with objects which generate holes and noise in the point cloud as the plastic bottle. [80].

Another problem, Figure 2.28, is that the FCN segmentation for objects under heavy occlusion or clutter is likely to be incomplete resulting in poor pose estimation, or undetected. This happens with more frequency with poor illumination. So for applications in the LHC

tunnel, which has unstructured and harsh environments, presenting several constraints like variable luminosity, difficult accessibility and light reflections, is not a so good algorithm. Even so, I think this paper creates a new baseline in the state-of-the-art.

The main teaching of this paper is that external constraints limit the set of states in which the system can be, which can lead to opportunities form simplifications and robustness in the perception system.

2.6.2 Deep Sliding Shapes

Another interesting paper uses a 3D ConvNet formulation that takes a 3D volumetric scene from a RGB-D image as input and outputs 3D object bounding boxes[82]. This work is based on *Sliding Shapes*[78].

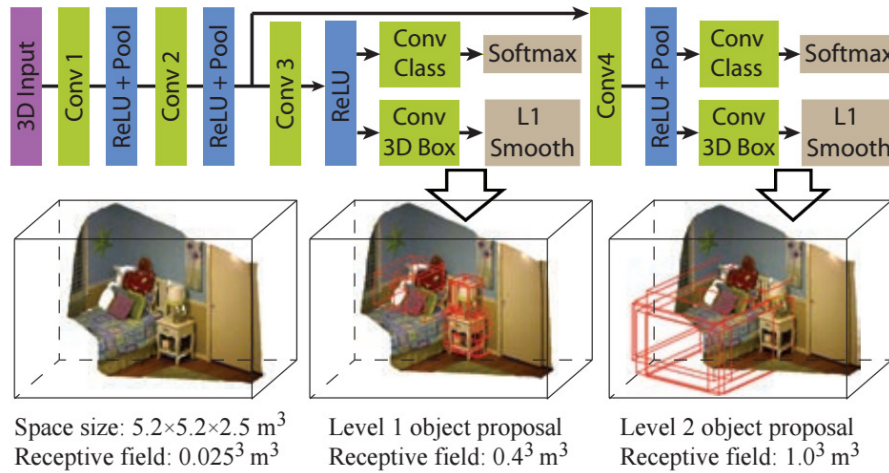


FIGURE 2.29: 3D Amodal Region Proposal Network.[82].

The authors introduce a complete 3D formulation to learn object proposals and classifiers using 3D convolutional neural networks (ConvNets). They propose in this paper the first 3D Region Proposal Network (RPN), that takes a 3D volumetric scene as input and outputs 3D object proposals. It is designed to generate proposals for whole objects at two different scales for objects of different sizes. They also propose the first joint Object Recognition Network (ORN) to use a 2D ConvNet to extract image features from color, and a 3D

ConvNet to extract geometric features from depth. This network is also the first to regress 3D bounding boxes for objects directly from 3D proposals.

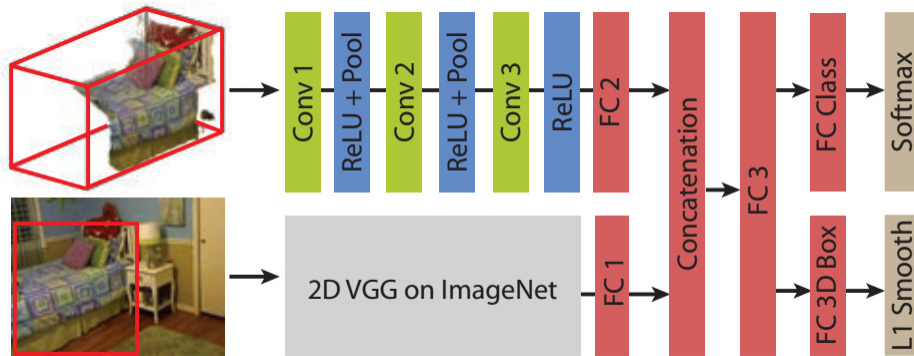


FIGURE 2.30: Joint Object Recognition Network[82].

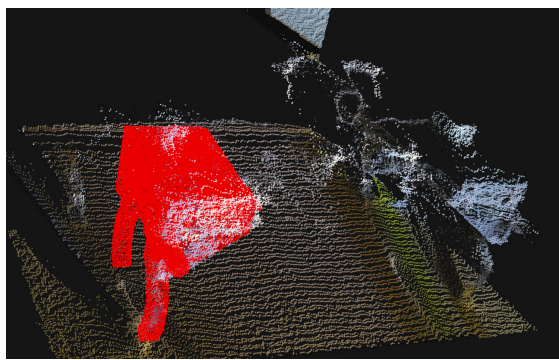
This algorithm is useful for scenarios similar to a room and objects present in it, but not for the LHC tunnel, with shape of tube and objects very near between them. Additionally, in the paper the authors don't say anything about test with metallic or transparent objects.

Chapter 3

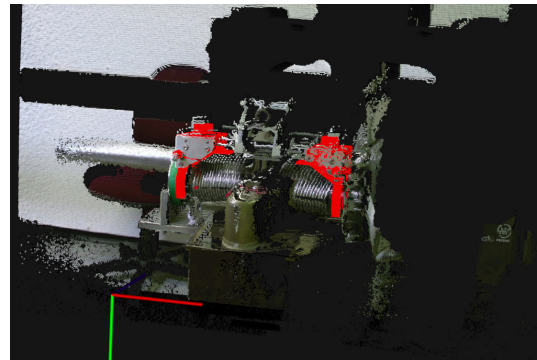
Pose Estimation System Proposed

3.1 Introduction

The estimation of the position and the rotation of the object has been solved by means of an algorithm specialized in each target object. The algorithm proposed uses mainly Point Cloud Library, therefore, the input will be a point cloud. In the first demonstration of concept was used point clouds obtained by the Kinect V2. But then the Orbbec Astra Pro camera was used. The Orbbec camera is mainly used for its size and weight, which can be placed on the robot arm tool. It also detects objects with less noise and detects well flat surfaces, which make up many objects in the LHC. Even so, the Kinect V2 camera gave good results in demonstrating the concept of the algorithm, Figure 3.1.



(A) Detecting a leg of the target.



(B) Detecting two legs of the target.

FIGURE 3.1: Prove of concept using a Kinect V2.

At first, it wanted to use a general algorithm that allowed to estimate the position, but

after a study of the State of the Art did not find any algorithm that worked properly. Therefore, it was thought to use a specialized algorithm for each object to be detected, but all based on the same idea and procedure.

The general idea of the algorithm is through the use of clustering algorithms and segmentation to find a robust point versus changes on the position of the camera. As this point is robust, it will always be the same point, even if the camera is in a different position. Once obtained the point can be known the rest of the points of the object overlapping a Point Cloud obtained from a CAD of the object. The transformation of the CAD mesh to a point cloud has been done with the CloudCompare[83] tool. To match this point cloud with the detected point in the point cloud of the camera, the center of the CAD point cloud is changed, and it is located at the same point that is to be detected. This operation is also done with CloudCompare. Finally, only the point cloud of the CAD should be moved to the distance XYZ of the detected point. In this way, we can know the position of the object and reconstruct it in 3D, Figure 3.2.



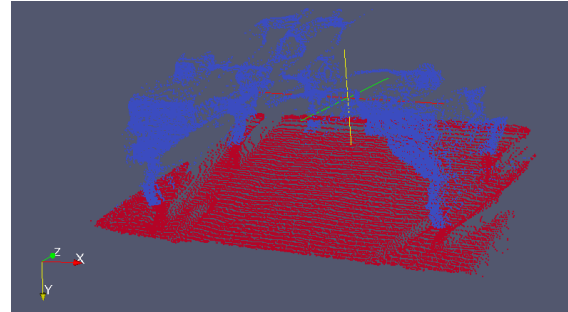
FIGURE 3.2: Final result of the algorithm.

In order to make a rapid development of the algorithm, the Paraview[84] tool with integrated Point Cloud Library[85] was used. Paraview is a powerful visualizer of CAD meshes that incorporates a wide variety of tools of post-processing, in this version includes PCL. Once the operations to be performed using Paraview are known, they are written in C ++

code using PCL.



(A) Clustering using Paraview.



(B) Segmentation using Paraview.

FIGURE 3.3: Different operations using Paraview with PCL.

3.2 Development

The RGBDTracking class has been created for position estimation in a simple way within the CERNTauro framework. The class header is as follows:

LISTING 3.1: header

```
#include <pcl/io/pcd_io.h>
#include <pcl/common/geometry.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/features/normal_3d.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/segmentation/extract_clusters.h>
#include <pcl/common/transforms.h>
#include <pcl/features/moment_of_inertia_estimation.h>
#include <pcl/visualization/cloud_viewer.h>

#include <pcl/registration/icp.h>

#include <unordered_map>
```



```

template<typename PointT>
class RGBDTracking {

private:

    typename pcl::PointCloud<PointT>::Ptr _cloud;

    float cosTheta(Eigen::Vector3f vector1, Eigen::Vector3f vector2);

    void Model_Rotation(pcl::PointCloud<pcl::PointXYZ>::Ptr model, float angle
        , Eigen::Vector3f axis,
            pcl::PointCloud<pcl::PointXYZ>::Ptr &transformed_cloud
        );

    void Model_Translation(pcl::PointCloud<pcl::PointXYZ>::Ptr model, pcl::
        PointXYZ mainCorner,
            pcl::PointCloud<pcl::PointXYZ>::Ptr &transformed_cloud);

    void Model_Translation(pcl::PointCloud<pcl::PointXYZ>::Ptr model, pcl::
        PointXYZ mainCorner, pcl::PointXYZ secondCorner,
            pcl::PointCloud<pcl::PointXYZ>::Ptr &
                transformed_cloud);

    void Visualization(pcl::PointCloud<pcl::PointXYZ>::Ptr &leftLeg, pcl::
        PointCloud<pcl::PointXYZ>::Ptr &rightLeg,
            pcl::PointCloud<pcl::PointXYZ>::Ptr &center);

    void Visualization(pcl::PointCloud<pcl::PointXYZ>::Ptr &leftLeg, pcl::
        PointCloud<pcl::PointXYZ>::Ptr &rightLeg);

    void Visualization(pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud);

    void Visualization();

    void Segmentation(pcl::PointCloud<pcl::PointXYZ>::Ptr &input, float
        threshold, bool setNegative,

```



```

        pcl::PointCloud<pcl::PointXYZ>::Ptr &output, Eigen::
            Vector3f axis, double epsAngle);

void Clustering(pcl::PointCloud<pcl::PointXYZ>::Ptr &input, float
    tolerance, std::vector<pcl::PointIndices> &cluster_indices);

void DepthFilter(float distance, pcl::PointCloud<pcl::PointXYZ>::Ptr cloud
    );

void HeightFilter(float distance, pcl::PointCloud<pcl::PointXYZ>::Ptr
    cloud);

std::unordered_map <std::string, pcl::PointXYZ> FindCorners(pcl::
    PointCloud<pcl::PointXYZ>::Ptr cloud);

std::unordered_map <std::string, pcl::PointXYZ> FindBottomCorners(pcl::
    PointCloud<pcl::PointXYZ>::Ptr cloud);

std::unordered_map <std::string, pcl::PointXYZ> InertiaMoments(pcl::
    PointCloud<pcl::PointXYZ>::Ptr cloud, bool visualize);

void ExtractSlice(pcl::PointCloud<pcl::PointXYZ>::Ptr input, float
    threshold,
                                Eigen::Vector3f *axis, pcl::
                                PointCloud<pcl::PointXYZ>::Ptr
                                &output);

void ExtractLegs(pcl::PointCloud<pcl::PointXYZ>::Ptr input, pcl::PointXYZ
    device_Center,
    pcl::PointCloud<pcl::PointXYZ>::Ptr &leftLeg, pcl::
    PointCloud<pcl::PointXYZ>::Ptr &rightLeg,
    std::vector<pcl::PointCloud<pcl::PointXYZ>::Ptr> &
    others_Parts);

ulong ExtractCenter(std::vector<pcl::PointCloud<pcl::PointXYZ>::Ptr>
    other_Parts, pcl::PointXYZ leftPoint, pcl::PointXYZ rightPoint,
    pcl::PointCloud<pcl::PointXYZ>::Ptr leftCenter, pcl::
    PointCloud<pcl::PointXYZ>::Ptr rightCenter);

```

```

public:

    RGBDTracking(typename pcl::PointCloud<PointT>::Ptr &cloud): _cloud(cloud)
    {}

    void Run();

    void Run2();

    void RunCollimator();

    void RunCollimator2();

};

```

A template, *template<typename PointT>*, has been used for the input data because the point cloud can come from a Kinect V2, in this case it would be the type of point with color, or another camera like the Orbbec, only the points.

The user simply instantiates the class by passing the point cloud obtained by the RGBD Camera and choosing the method of the object that wants to detect, *Run2()* for the separator and *RunCollimator2()* for the collimator. The *Run()* and *RunCollimator()* are deprecated methods.

LISTING 3.2: example.cpp

```

#include <RGBDCamera/RGBDTracking.hpp>

int main(int argc, char *argv[]) {
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::
        PointXYZ>());

    if (argc > 1) {
        pcl::io::loadPCDFile(argv[1], *cloud);
    } else {

```

```

        std::cout << "You must introduce a Point Cloud" << endl;
        return 1;
    }

    RGBDTracking<pcl::PointXYZ> test (cloud);

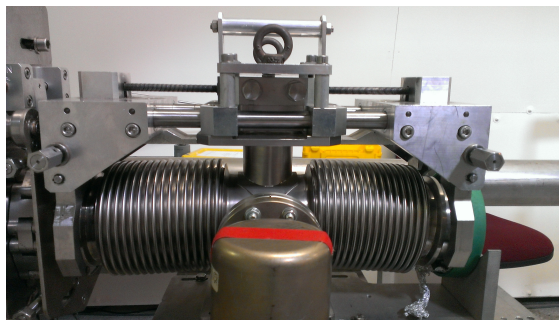
    test.RunCollimator2();

    return 0;
}

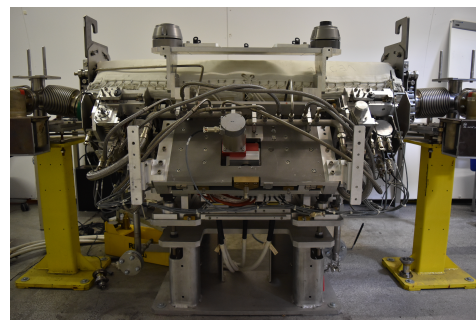
```

This program works by passing the name of a cloud of points stored in the hard disk and tries to estimate the position and rotation of the collimator present in the cloud. The output will be a dot cloud display with the reconstructed collimator in red, Figure 3.2.

The objects to be detected are the separator^{3.4a} and the collimator^{3.4b}, once the algorithms for both are exposed; a general algorithm will be defined to be used with other future objects.



(A) Photo of the separator.



(B) Photo of the collimator.

FIGURE 3.4: Objects to detect.

3.2.1 Separator

The separator is a key piece for detaching the collimator for inspections. It is a piece formed by three moving parts, so they will have relative positions different from each other. Therefore, obtaining the position of this piece can be decomposed to obtaining the position of the

two legs and the central part.

Since the input point cloud, Figure 3.5, contains elements that do not belong to the target, they must be removed. To do this, first, a filtering of points will be made, eliminating the most distant ones, since the object to be estimated, the position will be close to the camera, Figure 3.6.

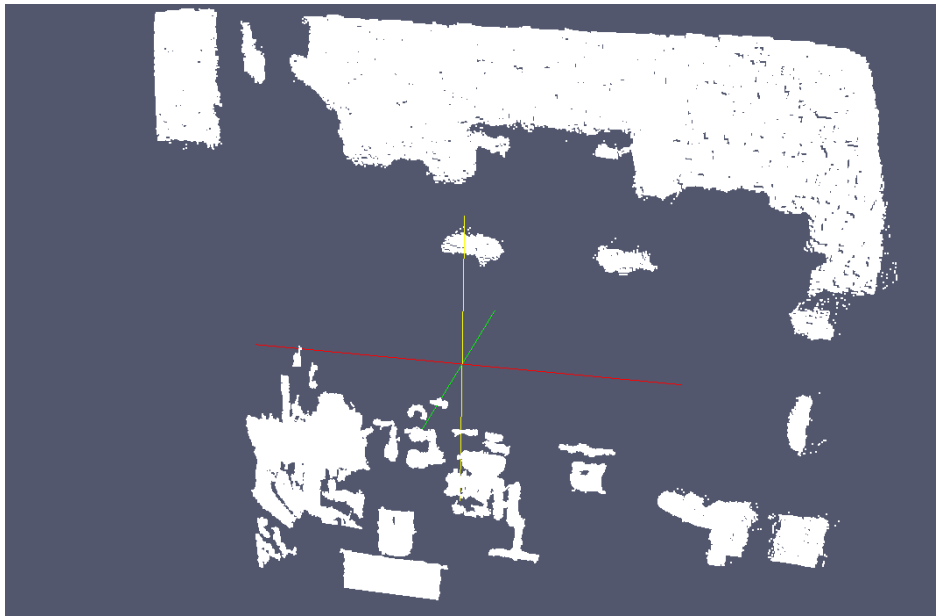


FIGURE 3.5: Input point cloud.

For this filtering, the *DepthFilter* 3.2.1 method has been created in which the point cloud is passed by reference and the distance from which points have to be removed.

LISTING 3.3: DepthFilter

```
template<typename PointT>
void RGBDTracking<PointT>::DepthFilter(float distance, pcl::PointCloud<pcl::
    PointXYZ>::Ptr cloud) {

    pcl::PointCloud<pcl::PointXYZ>::Ptr shrinkCloud (new pcl::PointCloud<pcl::
        PointXYZ>);
    uint pointsNumber = 0;
```

```

for (pcl::PointCloud<pcl::PointXYZ>::iterator it = cloud->begin(); it !=
cloud->end(); ++it){
    if (it->z < distance){
        shrinkCloud->points.push_back (*it);
        pointsNumber++;
    }
}

shrinkCloud->width = pointsNumber;
shrinkCloud->height = 1;
shrinkCloud->is_dense = true;

*cloud = *shrinkCloud;
}

```

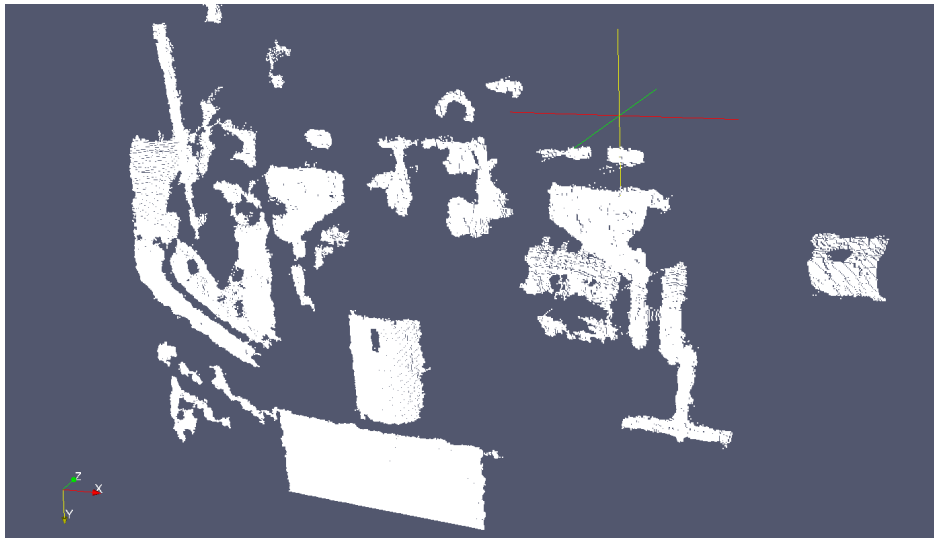


FIGURE 3.6: Point cloud with the most distant points filtered.

As the separator has the flat front of the legs will have to extract a planar segment of the cloud of points containing it. To do this, the *ExtractSlice* 3.2.1 method has been created. This iterating method looks for the plane whose normal is more parallel to the axis of the camera.

LISTING 3.4: ExtractSlice

```

template<typename PointT>

```

```

void RGBDTracking<PointT>::ExtractSlice(pcl::PointCloud<pcl::PointXYZ>::Ptr
    input, float threshold,
                                           Eigen::Vector3f *&axis, pcl::
                                           PointCloud<pcl::PointXYZ>::Ptr &
                                           output){

    for (int i = 20; i < 100; i++){
        Segmentation(input, threshold, false, output, *axis, i*0.01);
        std::unordered_map <std::string, pcl::PointXYZ> device_Inertia =
            InertiaMoments(output, false);

        Eigen::Vector3f normSurf(device_Inertia["z_axis"].x, device_Inertia["
            z_axis"].y, device_Inertia["z_axis"].z);
        Eigen::Vector3f zAxis(0,0,1);

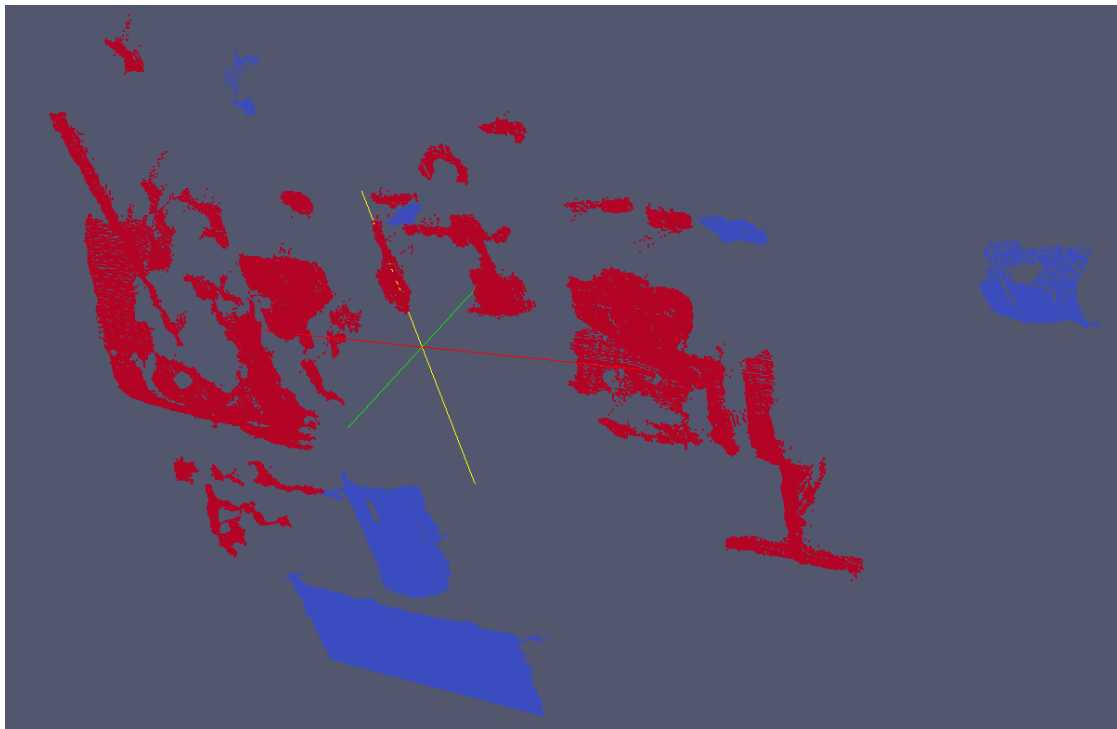
        float costheta = cosTheta(normSurf, zAxis);

        if (std::abs(costheta) > 0.95){
            cout << "costheta: " << costheta << endl;
            break;
        }
    }
}

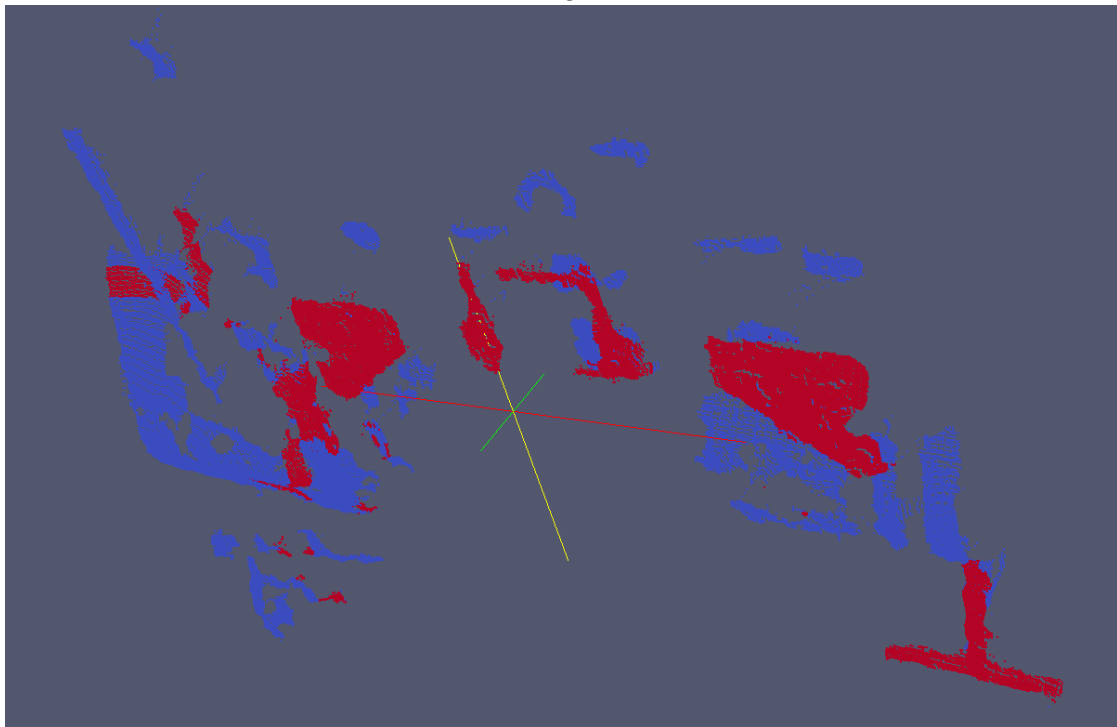
```

This method contains two methods, *InertiaMoments* and *Segmentation*. *InertiaMoments* extracts in an *unordered_map* the axis of inertia of the point cloud and its center of masses. *Segmentation* extracts a plane from the point cloud with a certain thickness, with a certain normal and a certain angle of the normal, this angle is the iterated.

First a thick segmentation, Figure 3.7a, will be done to eliminate elements that are closer to, and far from the desired area and on the obtained one will become ten times thinner, 3.7b.



(A) Thick Segmentation.



(B) Tight Segmentation.

FIGURE 3.7: Planar segmentation. The red zone is the extracted plane.

With the obtained surface, it is tried to look for the legs of the separator for it has been created the method *ExtractLegs* 3.2.1. This method performs a clustering, Figure 3.8, and classifies the two largest clusters as the separator's legs, differentiating the left from the right.

LISTING 3.5: ExtractLegs

```
template<typename PointT>
void RGBDTracking<PointT>::ExtractLegs (pcl::PointCloud<pcl::PointXYZ>::Ptr
    input,
                                     pcl::PointXYZ device_Center,
                                     pcl::PointCloud<pcl::PointXYZ>::Ptr &
                                     leftLeg,
                                     pcl::PointCloud<pcl::PointXYZ>::Ptr &
                                     rightLeg,
                                     std::vector<pcl::PointCloud<pcl::
                                     PointXYZ>::Ptr> &others_Parts){
    std::vector<pcl::PointIndices> cluster_indices_separator;

    Clustering(input, 0.005, cluster_indices_separator);

    ulong sizeLeg1 = 0;
    pcl::PointCloud<pcl::PointXYZ>::Ptr leg1 (new pcl::PointCloud<pcl::
        PointXYZ>);
    ulong sizeLeg2 = 0;
    pcl::PointCloud<pcl::PointXYZ>::Ptr leg2 (new pcl::PointCloud<pcl::
        PointXYZ>);

    ulong idLeg1 = 0;
    ulong idLeg2 = 0;

    //std::vector<pcl::PointCloud<pcl::PointXYZ>::Ptr> clusters;

    for (std::vector<pcl::PointIndices>::const_iterator it =
        cluster_indices_separator.begin ();
        it != cluster_indices_separator.end (); ++it)
    {
```



```

pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cluster (new pcl::PointCloud
    <pcl::PointXYZ>);
for (std::vector<int>::const_iterator pit = it->indices.begin ();
    pit != it->indices.end (); ++pit) {

    cloud_cluster->points.push_back (input->points[*pit]);
}

others_Parts.push_back(cloud_cluster);

cloud_cluster->width = (uint32_t)cloud_cluster->points.size();
cloud_cluster->height = 1;
cloud_cluster->is_dense = true;

std::unordered_map <std::string, pcl::PointXYZ> cluster_Inertia =
    InertiaMoments(cloud_cluster, false);

/*float distance = cluster_Inertia["center"].y - device_Center.y;*/

if (cloud_cluster->points.size () >= sizeLeg2 && cloud_cluster->points
    .size () >= sizeLeg1 /*&& distance < 0*/){

    sizeLeg2 = cloud_cluster->points.size();
    *leg2 = *cloud_cluster;
    idLeg2 = it - cluster_indices_separator.begin();
}

else if (cloud_cluster->points.size () >= sizeLeg1 /*&& distance < 0*/
    && std::abs(cloud_cluster->points[0].x - leg2->points[0].x) >
    0.2){
    sizeLeg1 = cloud_cluster->points.size();
    *leg1 = *cloud_cluster;
    idLeg1 = it - cluster_indices_separator.begin();
}
}

others_Parts.erase(others_Parts.begin() + idLeg1);
others_Parts.erase(others_Parts.begin() + idLeg2);

```

```

    if (leg1->points[0].x < leg2->points[0].x) {
        *leftLeg = *leg1;
        *rightLeg = *leg2;
    } else {
        *leftLeg = *leg2;
        *rightLeg = *leg1;
    }
}

```

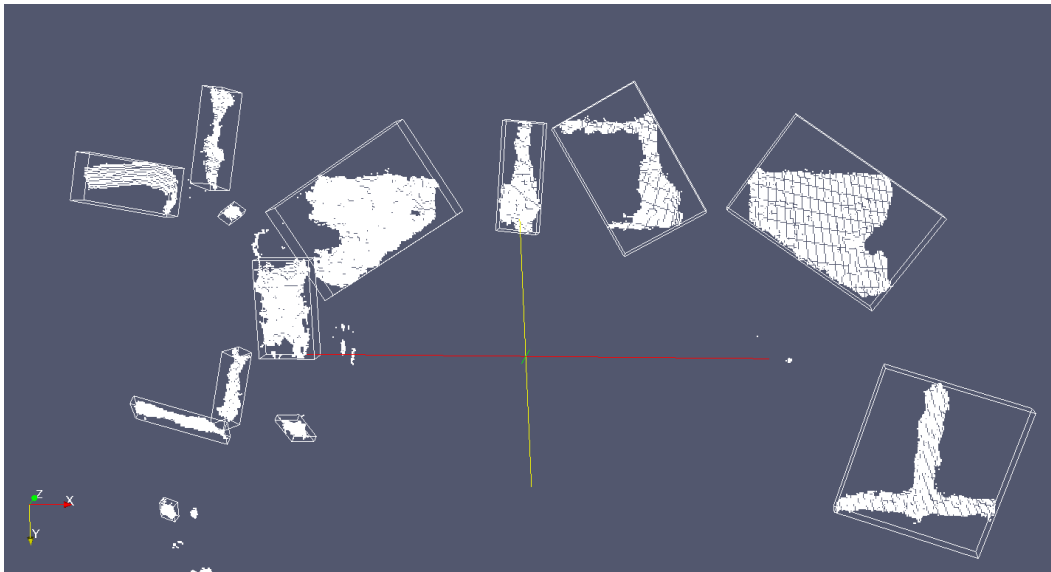


FIGURE 3.8: Clustering of the surface.

In the clusters of the legs, we look for the upper corners of the legs. To do this, the *FindCorners* 3.2.1 method has been created. The method looks for the points of the point cloud that are more to the left and to the right within a range. The left and right points are stored in an *unordered_map*.

LISTING 3.6: FindCorners

```

template<typename PointT>
std::unordered_map <std::string, pcl::PointXYZ> RGBDTracking<PointT>::
    FindBottomCorners(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud) {

    pcl::PointXYZ yCorner(cloud->points[0].x, cloud->points[0].y, cloud->
        points[0].z);
}

```

```

    for (size_t i = 1; i < cloud->points.size(); i++) {
        if (yCorner.y < cloud->points[i].y) {
            yCorner.x = cloud->points[i].x;
            yCorner.y = cloud->points[i].y;
            yCorner.z = cloud->points[i].z;
        }
    }

    pcl::PointXYZ leftCorner(yCorner);
    for (size_t i = 1; i < cloud->points.size(); i++) {
        if ((leftCorner.x > cloud->points[i].x) && (cloud->points[i].y > (
            yCorner.y - 0.02))) {
            leftCorner.x = cloud->points[i].x;
            leftCorner.y = cloud->points[i].y;
            leftCorner.z = cloud->points[i].z;
        }
    }

    pcl::PointXYZ rightCorner(yCorner);
    for (size_t i = 1; i < cloud->points.size(); i++) {
        if ((rightCorner.x < cloud->points[i].x) && (cloud->points[i].y > (
            yCorner.y - 0.02))) {
            rightCorner.x = cloud->points[i].x;
            rightCorner.y = cloud->points[i].y;
            rightCorner.z = cloud->points[i].z;
        }
    }

    std::unordered_map<std::string, pcl::PointXYZ> corners;
    corners["left"] = leftCorner;
    corners["right"] = rightCorner;

    return corners;
}

```

The remaining clusters are stored in the same point cloud, and the central part is extracted using the *ExtractCenter* method. To extract the central part, a clustering with the rest of clusters is made and the clusters that are between the corners of the legs are looked for.

Most of the time, two clusters are obtained, and the algorithm uses the one on the right, which as with the legs searches the top corners with *FindCorners*. If a single cluster is obtained the same thing is done.

LISTING 3.7: ExtractCenter

```
template<typename PointT>
ulong RGBDTracking<PointT>::ExtractCenter(std::vector<pcl::PointCloud<pcl::
    PointXYZ>::Ptr> other_Parts, pcl::PointXYZ leftPoint, pcl::PointXYZ
    rightPoint,
                                     pcl::PointCloud<pcl::PointXYZ>::Ptr
                                     leftCenter, pcl::PointCloud<pcl::
                                     PointXYZ>::Ptr rightCenter){

    std::vector<pcl::PointCloud<pcl::PointXYZ>::Ptr> center;

    for (std::vector<pcl::PointCloud<pcl::PointXYZ>::Ptr>::iterator it =
        other_Parts.begin(); it != other_Parts.end(); ++it){
        if (it->get()->points[0].x > leftPoint.x &&
            it->get()->points[0].x < rightPoint.x) {
            center.push_back(*it);
        }
    }

    if (center.size() == 1){
        *rightCenter = *center[0];
    }
    else {
        if (center[0]->points[0].x > center[1]->points[0].x){

            *leftCenter = *center[1];
            *rightCenter = *center[0];
        }
        else {
            *leftCenter = *center[0];
            *rightCenter = *center[1];
        }
    }
}
```

```

    }

    return center.size();
}

```

For the matching of the points obtained with the CAD models, the point clouds, Figure 3.9, created from these CAD models are used.

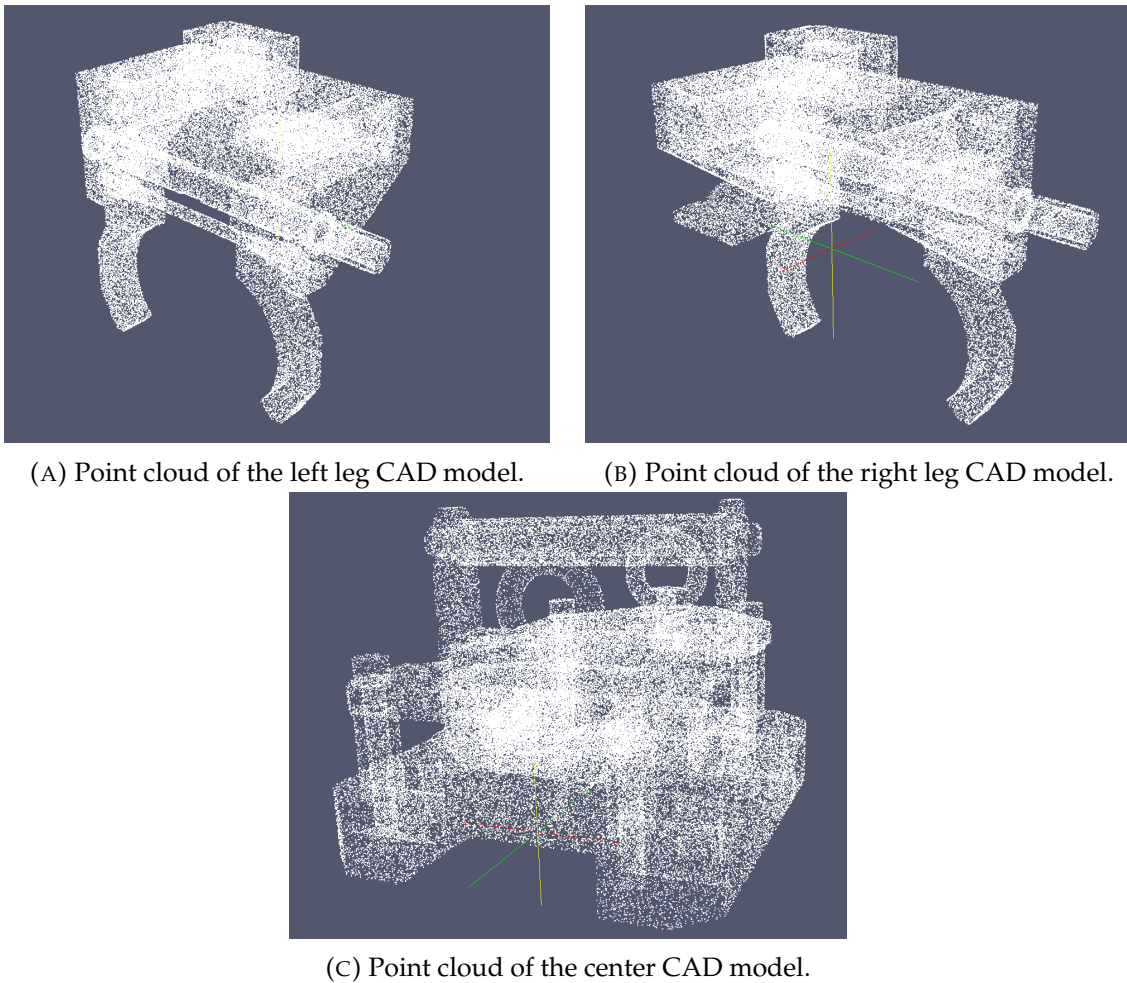


FIGURE 3.9: Point clouds of the CAD models.

To perform matching, the *Model_Translation*3.2.1 method is used. This method translates the CADs to their respective points found. The left upper corner on left leg and upper right corner on left leg and center. It also rotates the object using the angle formed between the line joining the corners and the X axis of the point cloud.

LISTING 3.8: ModelTranslation

```

template<typename PointT>
void RGBDTracking<PointT>::Model_Translation(pcl::PointCloud<pcl::PointXYZ>::
    Ptr model, pcl::PointXYZ mainCorner, pcl::PointXYZ secondCorner,
                                           pcl::PointCloud<pcl::PointXYZ>::
                                           Ptr &transformed_cloud){

    Eigen::Affine3f transform = Eigen::Affine3f::Identity();

    // Define a translation
    transform.translation() << mainCorner.x, mainCorner.y, mainCorner.z;

    // Rotation
    Eigen::Vector3f direction(secondCorner.x - mainCorner.x, secondCorner.y -
        mainCorner.y, secondCorner.z - mainCorner.z);
    Eigen::Vector3f xAxis(1, 0, 0);
    float cosAngle = 0;
    if (direction.x() > 0) cosAngle = cosTheta(xAxis, direction);
    else {
        xAxis.x() = -1;
        cosAngle = cosTheta(xAxis, direction.normalized());
    }

    cout << "angle: " << acos(cosAngle) << endl; cout << "Axis: " <<  xAxis.
        cross(direction.normalized()) << endl;

    transform.rotate(Eigen::AngleAxisf((const float)acos(cosAngle), xAxis.
        cross(direction).normalized()));

    cout << "Main Axis: " << xAxis.cross(direction).normalized() << endl;

    // You can either apply transform_1 or transform_2; they are the same
    pcl::transformPointCloud (*model, *transformed_cloud, transform);
}

```

In this way, it is possible to estimate the position and rotation of the parts of the separator and to perform its reconstruction.

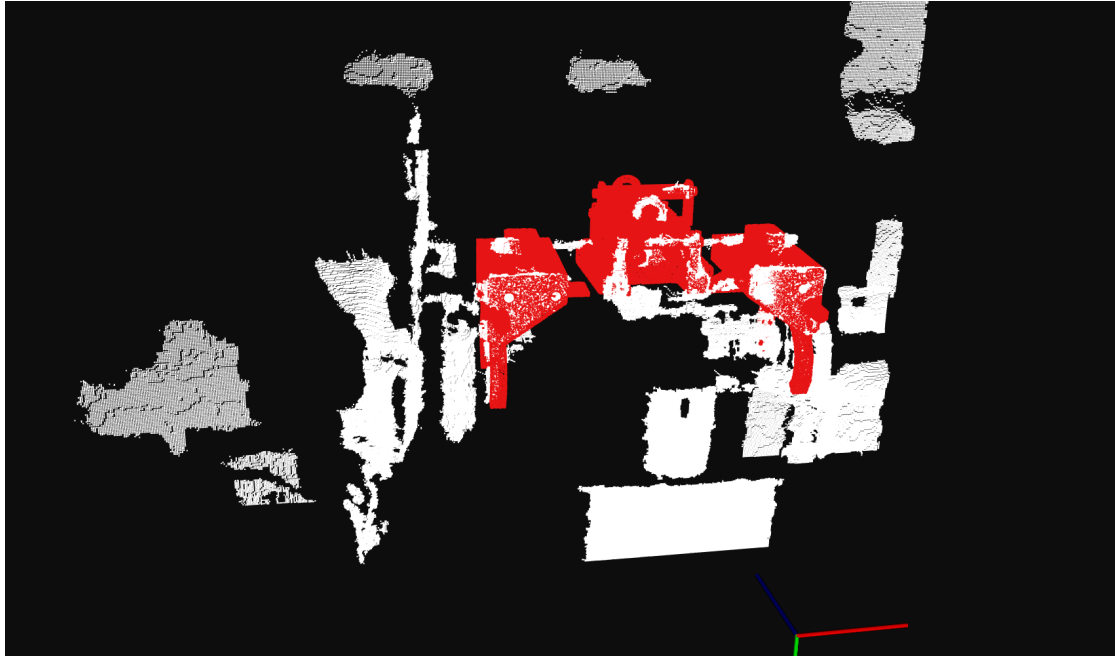


FIGURE 3.10: Reconstruction of the separator.

3.2.2 Collimator

The algorithm for detecting the position of the collimator is very similar to the developed by the separator. In this case only a single object, the collimator, has to be detected. The problem is that the object is much larger than the separator, so the point cloud will only show a part of the collimator, Figure 3.11.

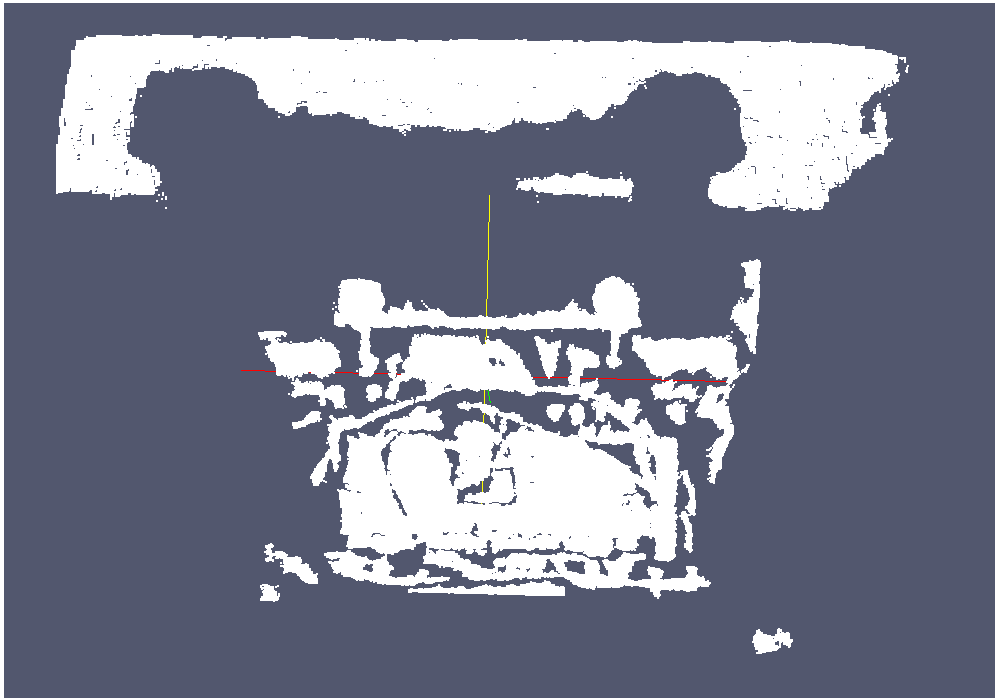


FIGURE 3.11: Input point cloud of the collimator.

First, the points farthest from the point cloud will be removed, just as the separator will be used the *DepthFilter* method, with a limit of 1.3 meters, Figure 3.12.

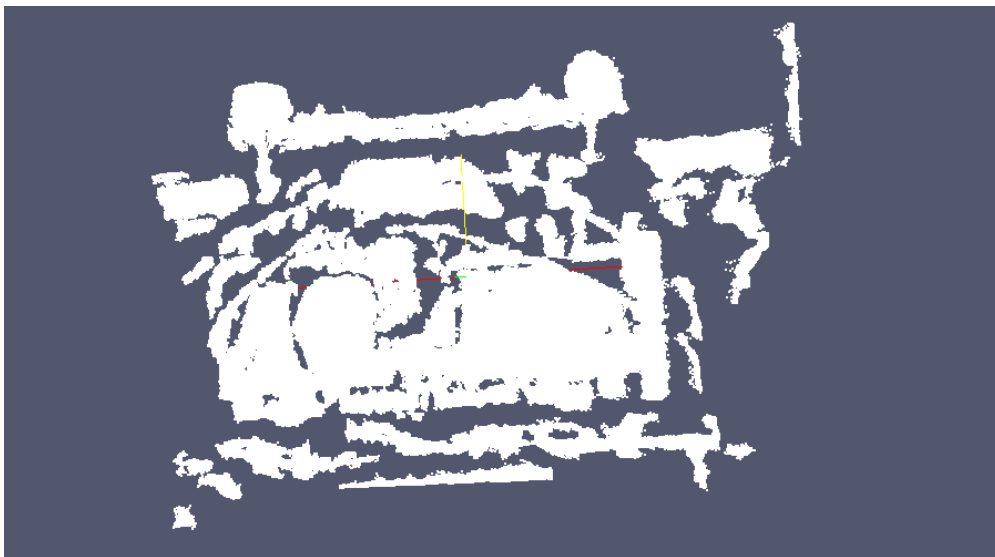


FIGURE 3.12: Filtered point cloud of the collimator.

If laterally the cloud of points is observed, Figure 3.13, it has three planes, two perpendicular to the axis of the camera and another oblique.

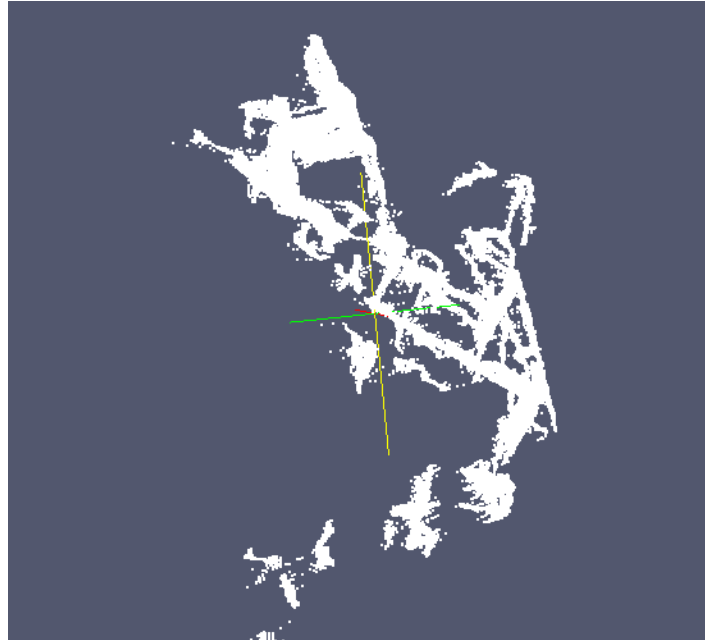


FIGURE 3.13: Lateral view of the point cloud of the collimator.

The upper plane is the desired plane, first it will have to detect the oblique plane, find its upper corners, then eliminate all the points of the cloud point below those corners and finally extract the desired plane.

To find the oblique plane will be done just like the separator, using the *Segmentation* method. However, it will have to look for a normal close to 45 degrees with respect to the axis of the camera, Figure 3.14.

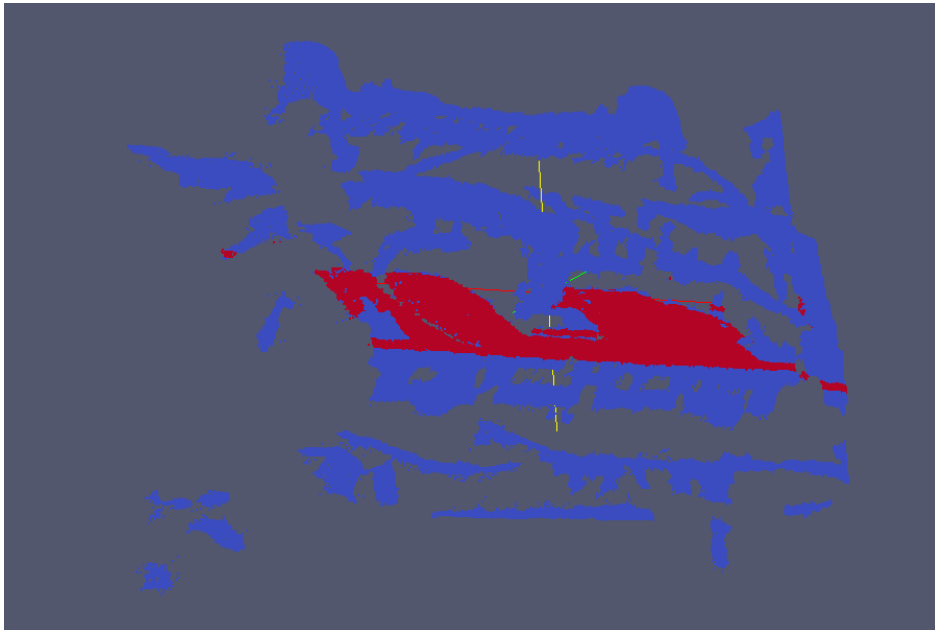


FIGURE 3.14: Extraction of the oblique plane.

As the extracted plane contains zones belonging to other parts of the collimator, a clustering will be carried out, and it will remain with the larger cluster which is the one corresponding to the searched plane, Figure 3.15.

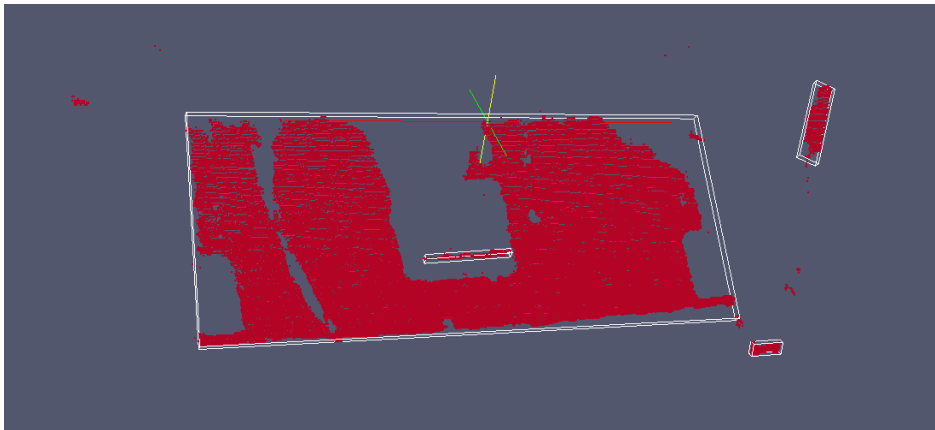


FIGURE 3.15: Clustering of the oblique plane.

The corners of the plane will be searched, using the *FindCorners* method, and a filtering of points will be done, eliminating those which are below the height of the corners; this is done with the method *HeightFilter* 3.2.2.

LISTING 3.9: HeightFilter

```

template<typename PointT>
void RGBDTracking<PointT>::HeightFilter(float distance, pcl::PointCloud<pcl::
    PointXYZ>::Ptr cloud){

    pcl::PointCloud<pcl::PointXYZ>::Ptr shrinkCloud (new pcl::PointCloud<pcl::
        PointXYZ>);
    uint pointsNumber = 0;

    for (pcl::PointCloud<pcl::PointXYZ>::iterator it = cloud->begin(); it !=
        cloud->end(); ++it){
        if (it->y < distance){
            shrinkCloud->points.push_back (*it);
            pointsNumber++;
        }
    }

    shrinkCloud->width = pointsNumber;
    shrinkCloud->height = 1;
    shrinkCloud->is_dense = true;

    *cloud = *shrinkCloud;
}

```

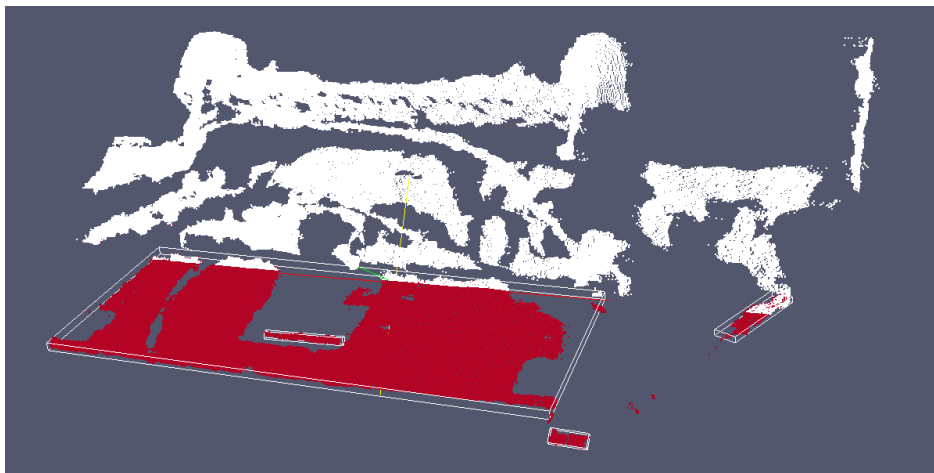
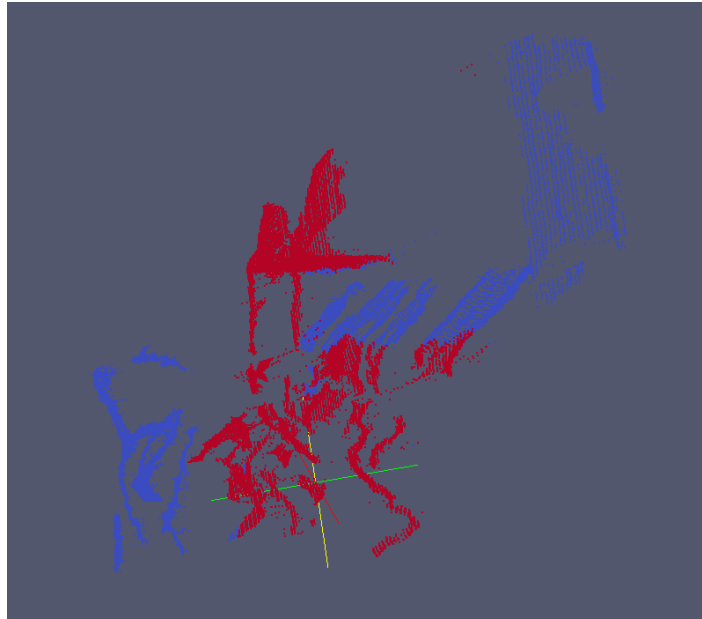
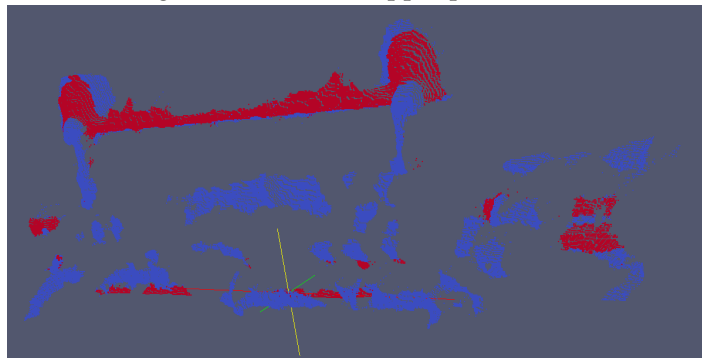


FIGURE 3.16: The red dots belong to the oblique plane previously detected and the white dots are those that are saved after using the *HeightFilter*.

With the interesting part will be done, as in the separator, first, a thick segmentation, Figure 3.17a, to remove non-interesting elements, and then a tighter, Figure 3.17b, to extract the desired plane. In both tasks, the *ExtractSlice* method will be used.



(A) Thick segmentation of the upper part of the collimator.



(B) Tight segmentation of the upper part of the collimator.

FIGURE 3.17: Segmentation of the upper part of the collimator.

In the last extracted plane, the points that are above are looked for, and in these ones is looked for that is more to the left and more to the right. Using the *FindCorners* method.

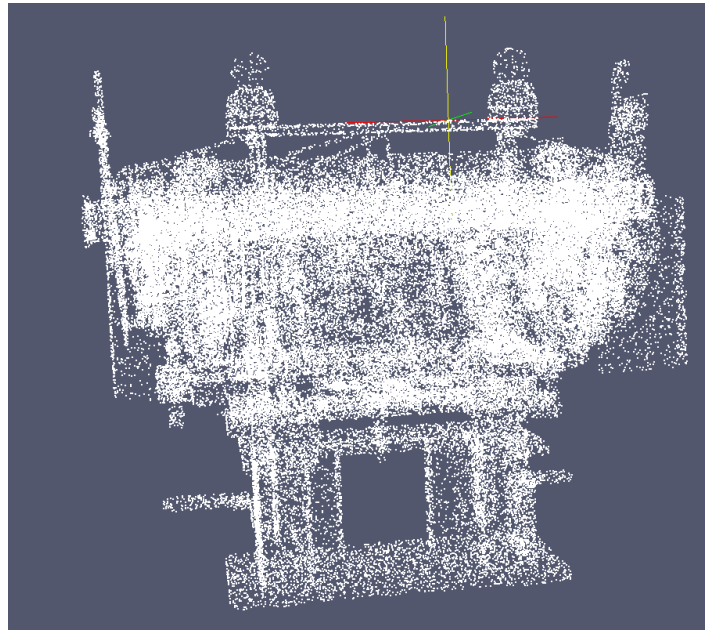


FIGURE 3.18: Point cloud of the collimator CAD model

Finally the point cloud generated from the CAD model, Figure 3.18, is transferred to the position of the corners previously found, obtaining the final reconstruction of the collimator in the LHC tunnel, Figure 3.19.



FIGURE 3.19: Reconstruction of the collimator.

3.2.3 General Objects

Throughout the development of the algorithms for the detection of the position of the separator and the collimator, we observe that they share many similar steps. This is because both developments are based on the same idea of looking for robust points from which to rebuild the object.

To locate these points, a plane of the object that has robust corners before changes of the camera must be located. This is usually given at the top of the objects. Therefore, a series of point filtrations will have to be performed in order to locate that plane.

Several methods have been developed for the filtering: *DepthFilter*, *HeightFilter*, *Clustering*. In addition, to obtaining planes, *ExtractSlice*, which also allow filtering apart from locating the desired plane.

Once the desired plane is found, the top corners are searched, using the *FindCorners* method. A priori the CAD model of the object is taken and its center is changed to the point that is searched with the algorithm. This model then moves to the position found with *FindCorners* and rotates to fit both the left and right points. Thus finally one has the three-dimensional reconstruction of the object.

Chapter 4

Implementation and Validation

4.1 Introduction

Throughout the project, we have worked with different cameras RGBD, as has been exposed in the state of the art. Although the Orbbec Astra Pro camera finally has been selected, for the final implementation in the robot, Figure 4.1, a class has been created to work with any camera without knowing the code.

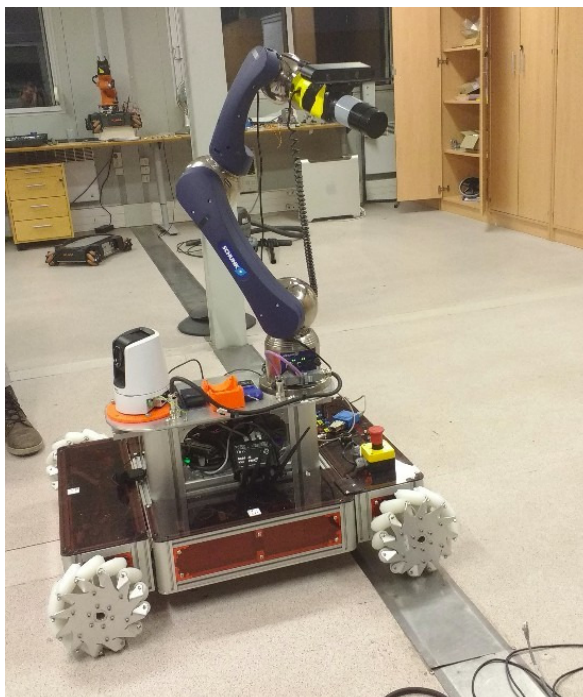


FIGURE 4.1: Mobile platform with robot arm being developed at CERN. On the arm tool is the Orbbec camera.

The tests have been done in a simulation of the LHC tunnel, so the conditions are similar to the real ones.

4.2 RGBD Camera Class

The RGBD cameras used for this project are very varied, so each one will have its own code. There are cameras that use the *openni2* library, such as Kinect V2 or Orbbec, others use *RealSense SDK*, such as Intel RealSense R200, some of these return the point cloud with real colors (Intel Realsense or Kinect V2) and others only return the point cloud without colors (Orbbec).

How to manage all these cameras in a transparent way for the user? It has chosen to use the factory pattern[86], Figure 4.2. Each camera class inherits from a RGBDCamera interface class. This class is also a template class; it can instantiate it to accept point cloud with color (PointXYZRGBA) and no color (PointXYZ).

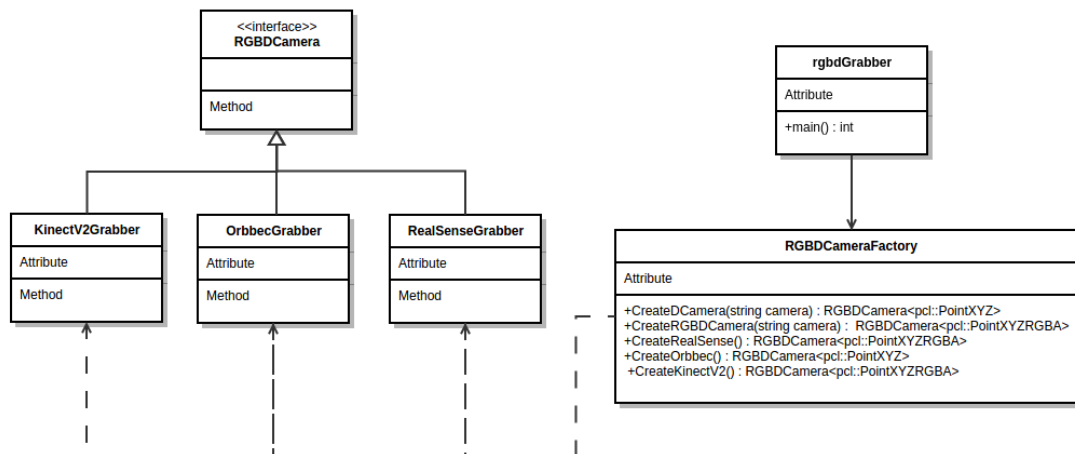


FIGURE 4.2: UML implementing the factory pattern for the management of RGBD cameras.

The interface class 4.2 implements methods that allow the management of the cameras, from their initialization, *InitScan*, to get the point cloud, *getScan*, and send the data by the

communications protocol implemented in the framework CERNTAURO, *PublishPointCloudMMAP*.

LISTING 4.1: RGBDCamera.h

```
template<typename PointT>
class RGBDCamera{

public:

    typedef pcl::PointCloud<PointT> Cloud;
    typedef typename Cloud::ConstPtr CloudConstPtr;

    RGBDCamera () {}
    ~RGBDCamera () {}

    virtual void InitScan() = 0;
    virtual void stopScan() = 0;
    virtual pcl::PointCloud<PointT> getScan() = 0;
    virtual void InitVis(std::string viewerName) = 0;
    virtual void visualization(CloudConstPtr cloud) = 0;
    virtual void PublishPointCloudMMAP(char * mmap_filename) = 0;

protected:

    pcl::visualization::CloudViewer *_viewer;
    CloudConstPtr _cloud;

};
```

The *RGBDCameraFactory*[4.2](#) class implements methods to access the cameras directly (*CreateRealsense*, *CreateOrbbec*, *CreateKinectV2*) or by giving it the name of the camera (*CreateDCamera*, *CreateRGBDCamera*).

LISTING 4.2: RGBDCameraFactory.hpp

```
class RGBDCameraFactory{

public:
```

```

static RGBDCamera<pcl::PointXYZ> * CreateDCamera(std::string camera){
    if (camera == "Orbbec"){
        return new OrbbecGrabber();
    }
}

static RGBDCamera<pcl::PointXYZRGBA> * CreateRGBDCamera(std::string camera
){
    if (camera == "KinectV2"){
        return new KinectV2Grabber();
    }

    if (camera == "RealSense"){
        return new RealSenseGrabber();
    }
}

static RGBDCamera<pcl::PointXYZRGBA> * CreateRealSense() {return new
    RealSenseGrabber(); }

static RGBDCamera<pcl::PointXYZ> * CreateOrbbec() {return new
    OrbbecGrabber(); }

static RGBDCamera<pcl::PointXYZRGBA> * CreateKinectV2() { return new
    KinectV2Grabber(); }

};

```

All this allows a simple management of the cameras by any CERNTAURO developer, as shown in the following example [4.2](#).

LISTING 4.3: rgbdGrabber.cpp

```

#include <RGBDCamera/RGBDCamera.h>
#include <pcl/io/pcd_io.h>
#include <RGBDCamera/RGBDCameraFactory.hpp>
#include <iostream>

int main(int argc, char *argv[]){

    RGBDCamera<pcl::PointXYZRGBA> * camera = RGBDCameraFactory::
        CreateRealSense();
}

```

```

pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud (new pcl::PointCloud<pcl::
    PointXYZRGBA>);

camera->InitVis("camera Viewer");
camera->InitScan();

for (;;) {

    *cloud = camera->getScan();
    camera->visualization(cloud);
    //if (getchar() == ' ') break;
}

if (argc > 1) {
    pcl::io::savePCDFileASCII(argv[1], *cloud);
}

camera->stopScan();
}

```

To switch to the kinectV2 camera just replace *CreateRealSense* with *CreateKinectV2*.

4.2.1 HDL32E Grabber

To handle the Lidar HDL32E was created a class 4.2.1 for its handling, as it is very similar to the class of RGBD cameras, would be easily implementable within the factory pattern.

LISTING 4.4: HDL32E.h

```

#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/io/hdl_grabber.h>
#include <pcl/visualization/point_cloud_color_handlers.h>
#include <pcl/visualization/cloud_viewer.h>

class HDL32E {

```

```

public:

    // Type definitions
    typedef pcl::PointCloud<pcl::PointXYZI> Cloud;
    typedef typename Cloud::ConstPtr CloudConstPtr;

    // Constructor
    HDL32E (pcl::HDLGrabber& grabber,
            pcl::visualization::PointCloudColorHandler<pcl::PointXYZI> &
            handler) :
        cloud_viewer_ (new pcl::visualization::PCLVisualizer ("PCL HDL
            Cloud")),
        grabber_ (grabber),
        handler_ (handler)
    {
    }

    // Initialize the Velodyne scan
    void initScan();
    // Stop the Velodyne scan
    void stopScan();
    // Get the point cloud of a scan frame
    pcl::PointCloud<pcl::PointXYZI> getScan ();
    // Initialize visualizator
    void InitVis();
    // Visualize the point cloud frame
    void visualization(Cloud::ConstPtr cloud);

private:

    // Private Variable:
    boost::shared_ptr<pcl::visualization::PCLVisualizer> cloud_viewer_;
    pcl::Grabber& grabber_;
    boost::mutex cloud_mutex_;
    CloudConstPtr cloud_;
    pcl::visualization::PointCloudColorHandler<pcl::PointXYZI> &handler_;
    boost::signals2::connection cloud_connection_;

```

```

// Private Methods:
void cloud_callback (const CloudConstPtr& cloud);
};

```

4.3 Tests and Validation

In order to test and validate the development of the algorithms, more than 100 collimation point cloud catches and another 100 capture of the separator were captured. Getting a good dataset for the development of this project. The datasets were made by varying the position of the camera and its inclinations with respect to the target. So each point cloud is different from the others.

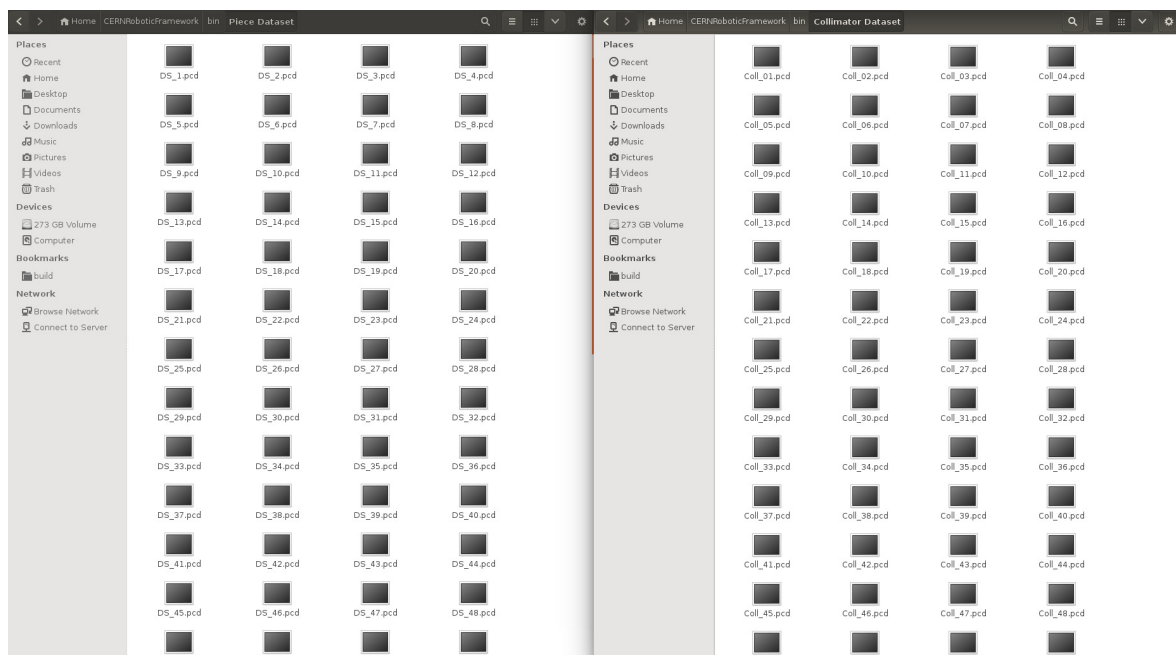
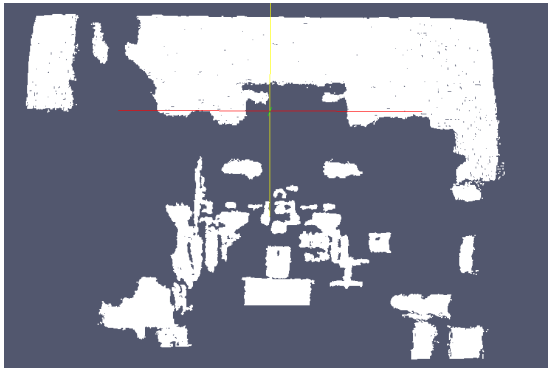
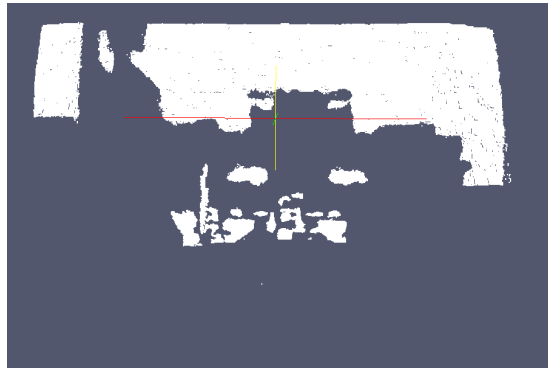


FIGURE 4.3: Point cloud datasets of the separator and collimator.

Examining the dataset it was discovered that the Orbbec camera sometimes makes the point cloud wrong, which results in a great loss of information, making position detection impossible, Figure 4.4.



(A) Good point cloud in the dataset.



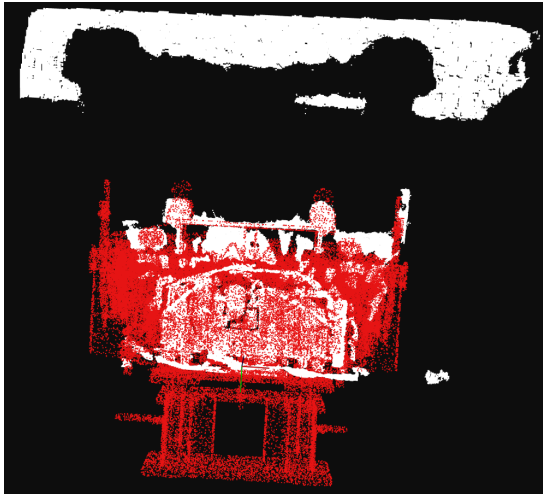
(B) Bad point cloud in the dataset.

FIGURE 4.4: There are bad point clouds in the dataset, because a problem in the Orbbec camera. These point clouds are useless.

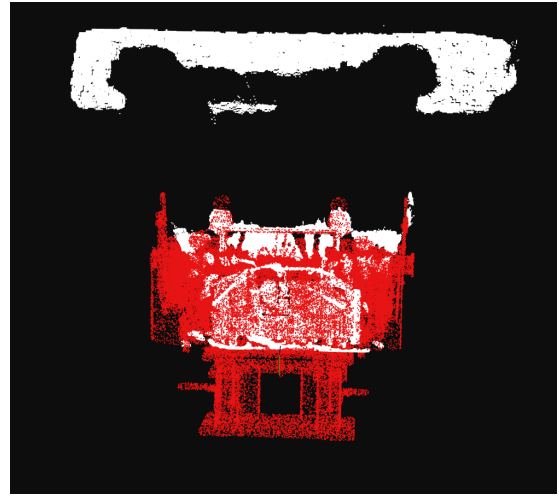
It has been proven that the developed algorithm works on a high percentage of the two datasets. With position errors less than $1cm$ in the case of the separated and between 2 and $3cm$ in the case of the collimator. This shows that the error depends on the distance to which the camera is located, in the case of the collimator being larger, the camera was situated a little further, although not too much because of the limited dimensions of the LHC tunnel.



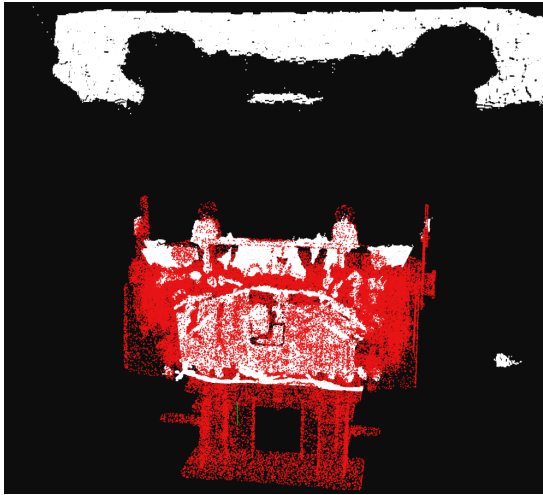
FIGURE 4.5: The dimensions of the LHC tunnel are limited. The datasets were made in a model of a segment of LHC with similar dimensions.



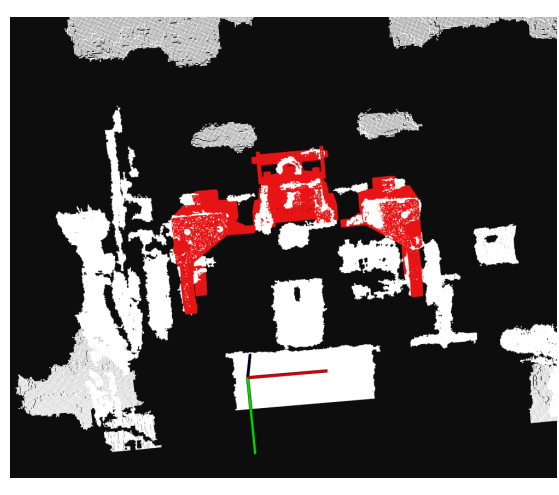
(A) Reconstruction of the collimator.



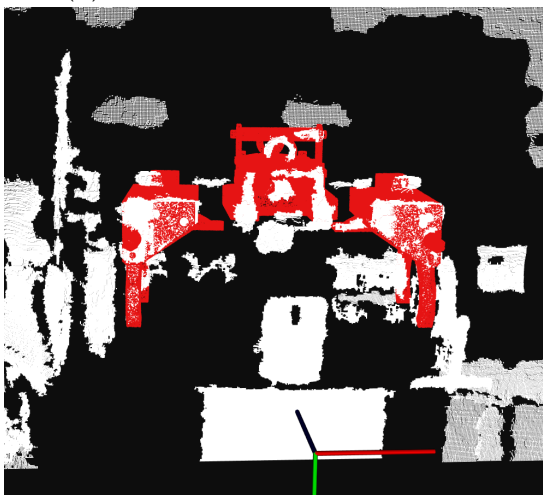
(B) Reconstruction of the collimator.



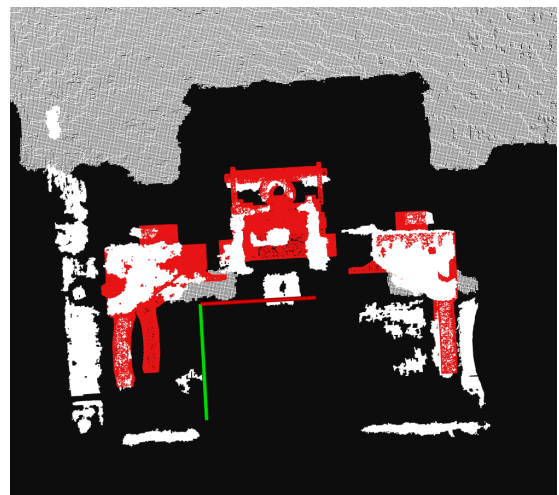
(C) Reconstruction of the collimator.



(D) Reconstruction of the separator.



(E) Reconstruction of the separator.



(F) Reconstruction of the separator.

FIGURE 4.6: Reconstruction of different point clouds of the dataset.

During the tests, it has been verified that the processing time of the algorithm is between 5 and 10 seconds. This depends on the segmentation operation, which must sometimes iterate over and over again until it finds the right plane. Times are quite acceptable, since the algorithm must be run only once, because the global position is known.

4.3.1 Robustness of the Algorithm

The robustness of the algorithm depends greatly on the point cloud captured by the camera. Therefore, the quality of the point cloud will have to be improved. One method is to capture various point clouds in different positions and merge them into a single point cloud.

During the realization of this project, I was asked to create a program to perform a three-dimensional mapping of an area using the RGBD camera of the robot. Each time the user takes a point cloud, the algorithm uses the homogeneous transformation matrix of the robot arm tool to transform the coordinates of the point cloud taken to the coordinates with respect to the base of the robot arm. When a new point cloud is taken, the coordinates of its points are transformed and added to the previous point clouds. This creates a single point cloud from several.

LISTING 4.5: CloudMapping.hpp

```
#include <pcl/common/geometry.h>
#include <pcl/common/transforms.h>
#include <pcl/point_cloud.h>
#include <pcl/io/pcd_io.h>

#include <IPC/MMAP.hpp>
#include <RGBDCamera/RGBDPackets.hpp>

/**
 * @brief Class CloudMapping. It makes a merge of multiple clouds to do a map.
 * @tparam PointT is a type of point
 */
```



```

template<typename PointT>
class CloudMapping {

private:
    /**
     * The point cloud that contains the previous point clouds.
     */
    typename pcl::PointCloud<PointT>::Ptr _total_Cloud;

public:
    /**
     * Class constructor. Get the first point cloud.
     * @param cloud First point cloud
     * @param homMatrix Homogenous Matrix of the Tool with respect to the Base
     * @return
     */
    CloudMapping(typename pcl::PointCloud<PointT>::Ptr cloud, Eigen::Matrix4f
        homMatrix);

    /**
     * Method to add a new point cloud.
     * @param cloud New point cloud
     * @param homMatrix Homogenous Matrix of the Tool with respect to the Base
     */
    void addCloud(typename pcl::PointCloud<PointT>::Ptr cloud, Eigen::Matrix4f
        homMatrix);

    /**
     * Method to publish the Total Cloud
     * @param mmap_filename the MMAP filename
     */
    void PublishDepthMMAP(char * mmap_filename);

    /**
     * Method to publish the total point cloud in a pcd file.
     * @param fileName
     */
    void WriteTotalCloud(std::string fileName);

```

```

};

template<typename PointT>
CloudMapping<PointT>::CloudMapping(typename pcl::PointCloud<PointT>::Ptr cloud
    , Eigen::Matrix4f homMatrix) {

    std::cout << homMatrix << std::endl;
    Eigen::Matrix4f changeBase;
    changeBase << 0 , -1, 0 , 0 ,
                  1 , 0 , 0 , 0 ,
                  0 , 0 , 1 , 0 ,
                  0 , 0 , 0 , 1;

    pcl::transformPointCloud (*cloud, *cloud, changeBase);
    pcl::transformPointCloud (*cloud, *cloud, homMatrix);
    _total_Cloud = cloud;
}

template<typename PointT>
void CloudMapping<PointT>::addCloud(typename pcl::PointCloud<PointT>::Ptr
    cloud, Eigen::Matrix4f homMatrix){

    //typename pcl::PointCloud<PointT>::Ptr transformed_cloud (new typename
        pcl::PointCloud<PointT>::Ptr);

    std::cout << homMatrix << std::endl;
    Eigen::Matrix4f changeBase;
    changeBase << 0 , -1, 0 , 0 ,
                  1 , 0 , 0 , 0 ,
                  0 , 0 , 1 , 0 ,
                  0 , 0 , 0 , 1;

    pcl::transformPointCloud (*cloud, *cloud, changeBase);
    pcl::transformPointCloud (*cloud, *cloud, homMatrix);
    *_total_Cloud += *cloud;
}

template<typename PointT>
void CloudMapping<PointT>::PublishDepthMMap(char * mmap_filename) {

    uint size = _total_Cloud->size();

```

```

    MMAP mmap = MMAP::CreateWriter(mmap_filename, size * sizeof(Packets::
        DepthPointPacket));
    Packets::PacketHeader header;
    header.type = Packets::DEPTH_POINT_PACKET;
    header.lenght = size * sizeof(Packets::DepthPointPacket);
    Packets::DepthPointPacket packet [size];

    for (uint i = 0; i < size; i++){
        packet[i].x = (short)std::round(_total_Cloud->points[i].x * 1000);
        packet[i].y = (short)std::round(_total_Cloud->points[i].y * 1000);
        packet[i].z = (short)std::round(_total_Cloud->points[i].z * 1000);
    }

    mmap.write(packet[0], header);
}

template<typename PointT>
void CloudMapping<PointT>::WriteTotalCloud(std::string fileName) {

    pcl::io::savePCDFFileASCII(fileName, *_total_Cloud);
}

```

Using this class [4.3.1](#), it can do a scan of the target and creating a point cloud with more points and more detail making the detection of the position and its reconstruction more robust. In addition, it could solve the problem that was raised in the state of the art with the reflections of the metallic pieces.

4.3.2 Accuracy of the Algorithm

As seen in the tests on large pieces, such as the collimator, the error of the estimated position increases, approaching levels that would cause problems in teleoperation.

One way to reduce this error is to make a second estimation of the position. Once the first estimation is made, the camera in the robot arm can be approached to a predetermined part, this could be done automatically. Using the algorithm to detect that part of the large piece, it is detected with minor errors. And since the position of that part is known a priori

with respect to the rest, this allows to reduce the error of the global piece.

4.3.3 Degree of Autonomy of the Algorithm

The degree of autonomy of the algorithm is a bit low. Since it is necessary to place the camera/robot in front of the target and select the object to be detected. A good way to increase autonomy would be to implement a detection system, allowing to detect automatically what is captured on the image. Currently, the user must indicate which piece has been captured in the point cloud for the algorithm to work.

In the last years, many Deep Learning models have been developed for the image recognition[87][35][88][89][90] and object detection[91][92]. Programs like YOLO[93], which works in real time, can be implemented and when it detects objects for which it has been trained, then it can be placed in front of the object and execute the algorithm developed in this project autonomously without the direct action of the user.

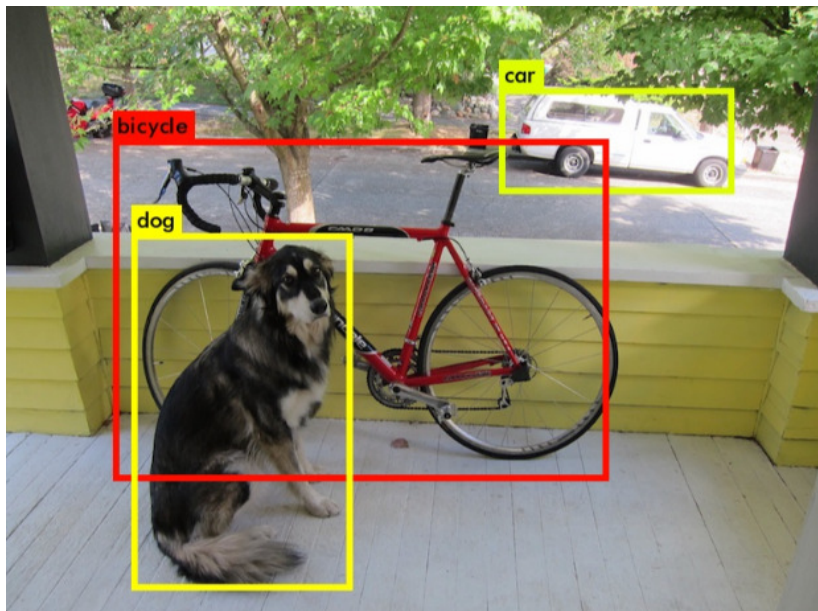


FIGURE 4.7: Object detection using YOLOv2[93].

Other deep learning models allow the segmentation of images[94][96][95]. Therefore, in addition to detecting the object, it would be possible to delete all the points of the point

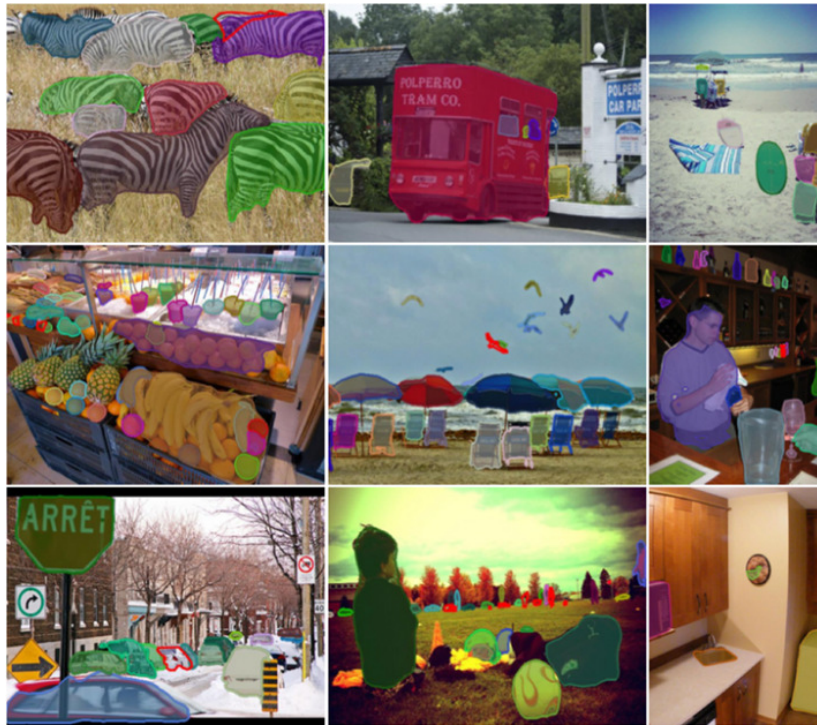


FIGURE 4.8: Object segmentation using *DeepMask*[94] and *SharpMask*[95].

cloud that do not belong to the object to be detected, thus improving the degree of autonomy and robustness of the algorithm.

4.4 Integration with the CERNTAURO Framework

CERN Telemannipulation semi-AutonomousUnit for Robot Operations, CERNTAURO. CERNTAURO is a software library that is being developed at CERN for the management of fully teleoperated or semi-autonomous robots. This project is integrated within CERNTAURO for tasks of semi-autonomous teleoperation, Figure 4.9.

Communication with other CERNTAURO programs is done through TCP communication. This allows sending data to other computers in the robot and to the remote control center of the robot. The remote user controls the operation of the cameras and when executing the position detection system designed for this project, by sending commands via TCP.

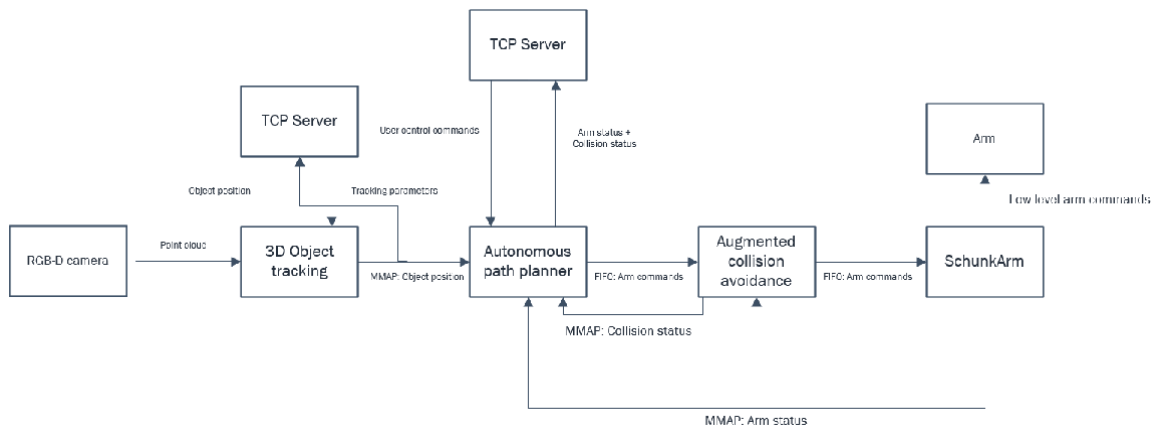


FIGURE 4.9: The developed in this project corresponds to RGBD Cameras and 3D Object Tracking.

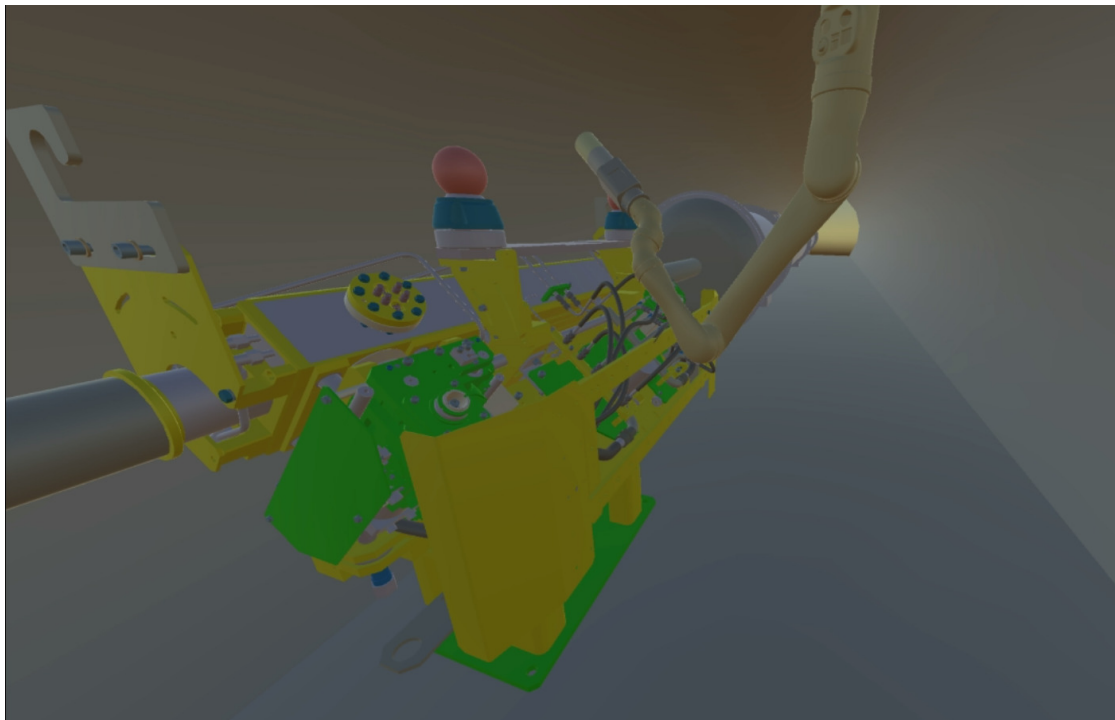
4.5 Integration with Virtual Reality

During the development of this project I was asked to create in Unity3D[97] a simulation for Virtual Reality of a segment of the LHC. In the interactive simulation had to be present the different robots that are used in the CERN, these could be moved through the keyboard, Figure 4.10.

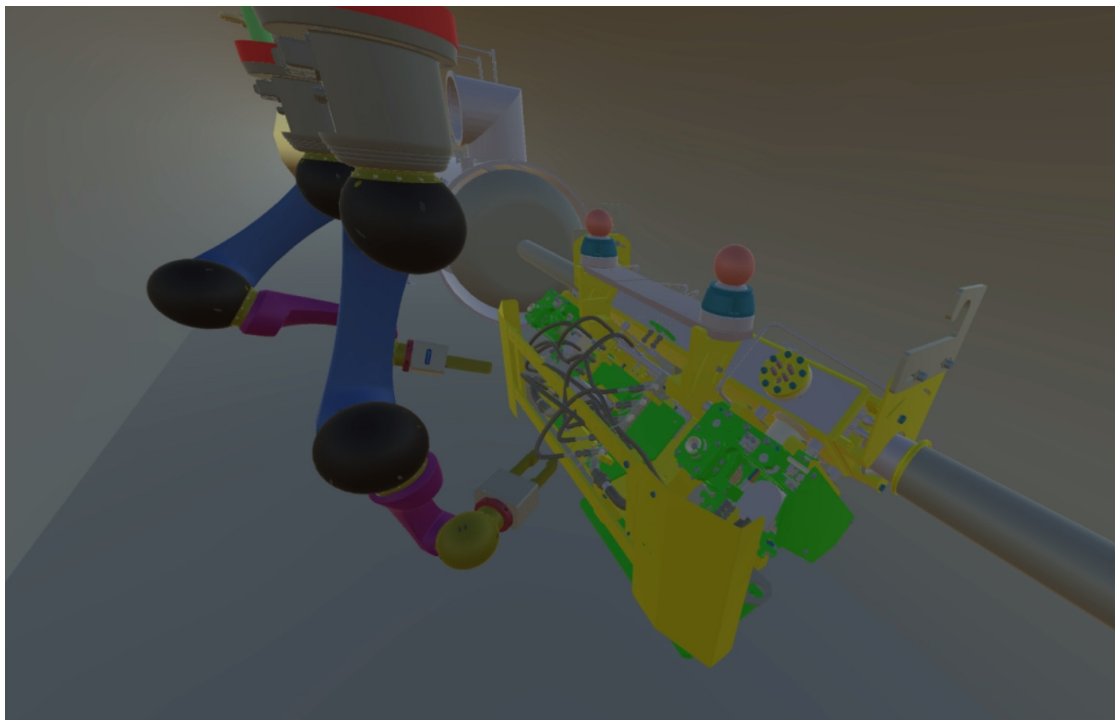
Virtual reality headsets have acquired a good degree of maturity in recent years, companies like Facebook are betting heavily on this technology[98].

This virtual reality environment in Unity simulating a segment of the LHC can be combined with the algorithm developed during this project. Once the algorithm has detected the position of the target, it could be placed in the simulation in Unity the object in that position. In addition, the same CAD models are used, so the integration would be simple when using the same values.

It can also add to the simulation in Unity the point cloud captured by the camera, thus adding real information to the user. This 3D reconstruction of the environment in Unity would allow the user to teleoperate the robot more naturally. The user could move as if he were in the place and having a three-dimensional view could handle the robot from different points of view and achieve a better performance in teleoperation.



(A) Collimator and Kinova Robot.



(B) Collimator and two Schunkarms.

FIGURE 4.10: Segment of LHC tunnel in Virtual Reality using Unity.

Chapter 5

Conclusions and Future Development

5.1 Conclusions

The main objectives of the project have been achieved by developing algorithms for the detection of pieces of great importance in the operation of the LHC. A proof of concept was performed with the separator placed on a table and demonstrated the robustness of the algorithm by placing the part in a model of the LHC tunnel. The algorithm was extended to larger objects such as the collimator. And finally a generalization of the algorithm has been exposed to be able to detect other objects of the LHC.

One of the advantages of the algorithm is its reduced processing time, a few seconds. It is an important part of the semi-autonomous teleoperation within the CERNTAURO framework. In addition, it allows the three-dimensional reconstruction of the target; this reconstruction is important for detecting collisions and performing path planning in areas that the cameras detect incompletely. Finally, its integration with virtual reality systems would increase the teleoperation skill of the user.

5.2 Future Development

The main objective to the future is to demonstrate the position detection for general objects and improve the robustness and the accuracy, and increase the degree of autonomy of the proposed algorithm. In addition, the integration of this software in a virtual reality system has to be completed.

Chapter 6

Planning and Scope of the Project

6.1 Work Breakdown Structure

The work breakdown structure (WBS) is shown in Figure 6.1.

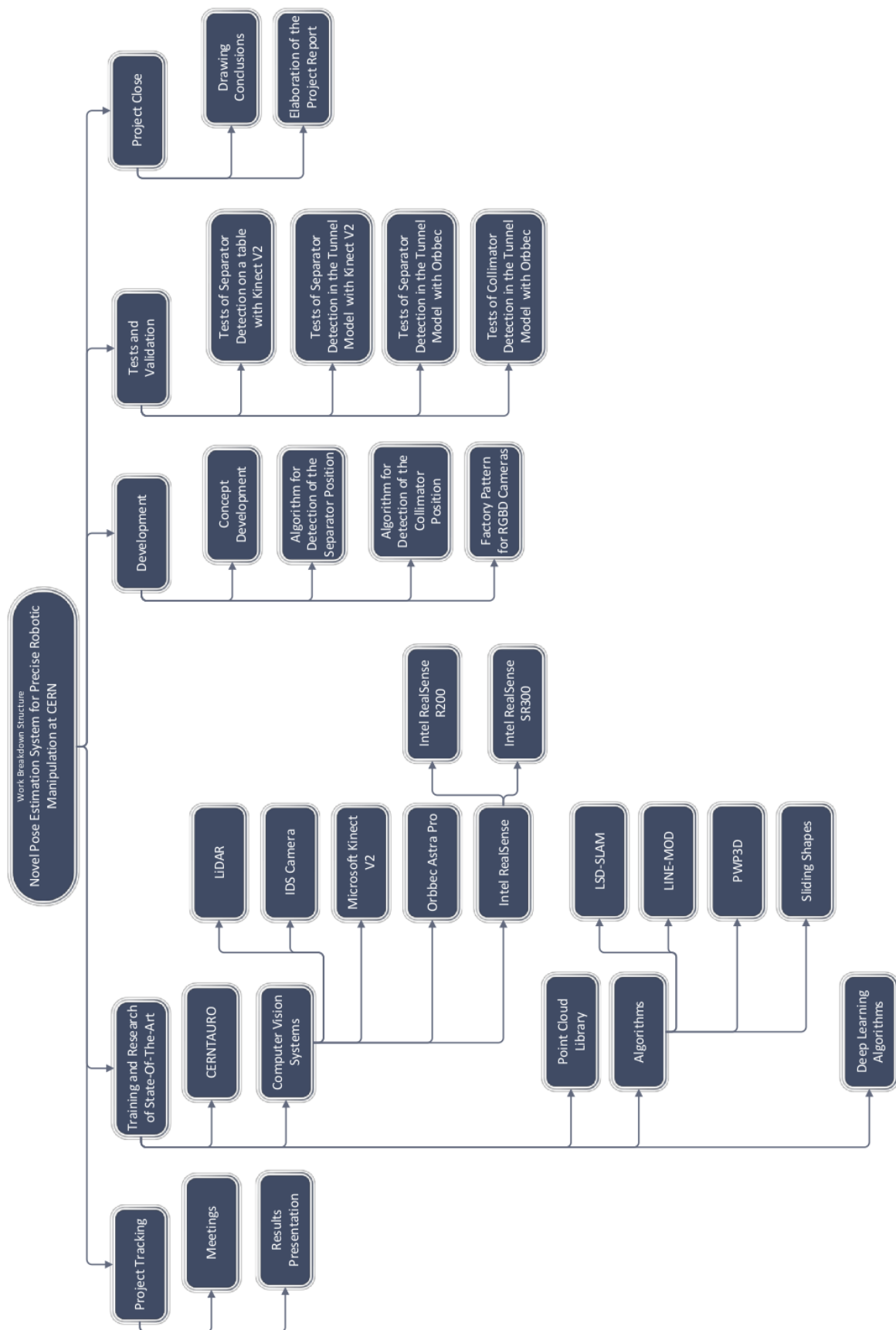


FIGURE 6.1: Work Breakdown Structure.

6.2 Gantt Diagram

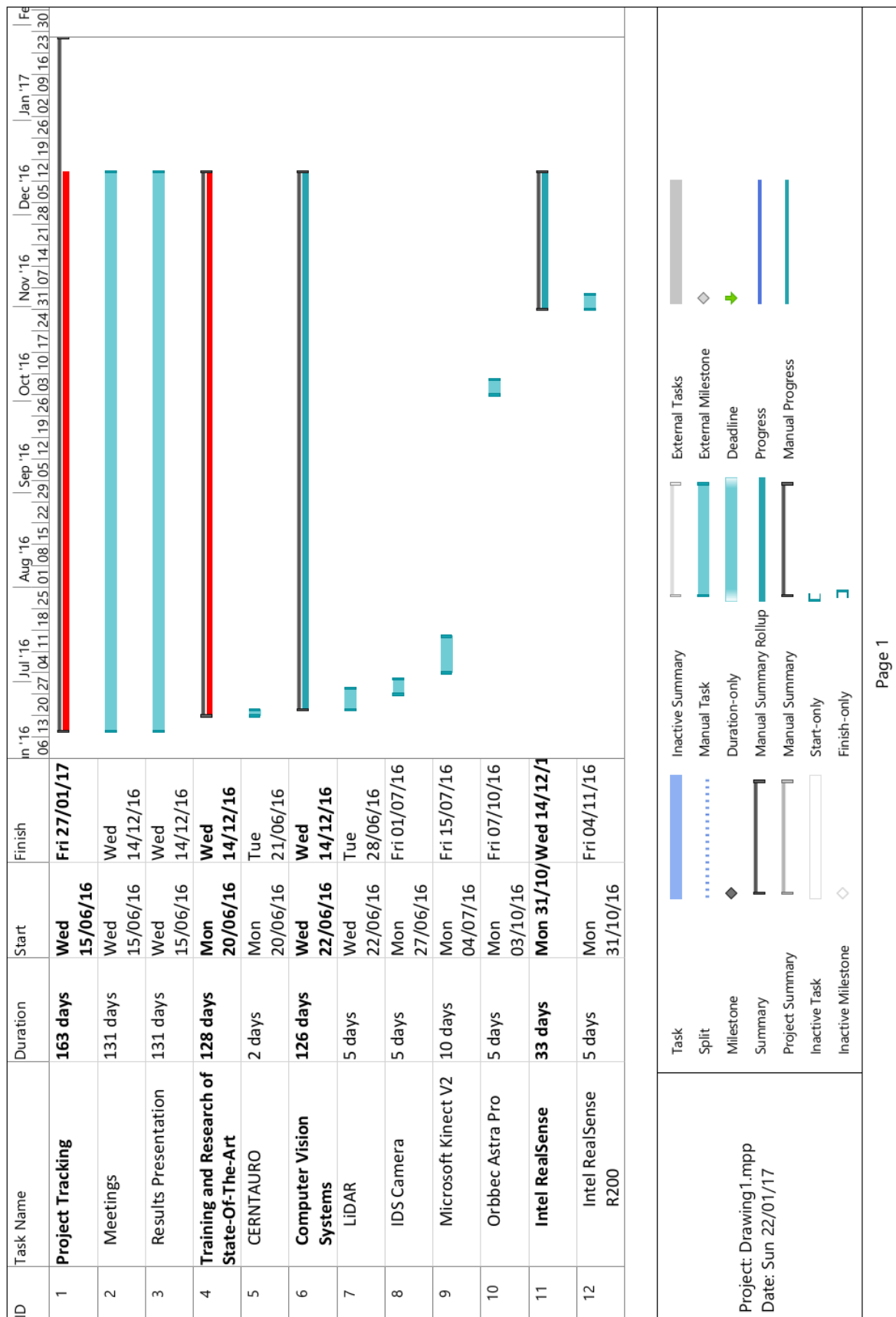


FIGURE 6.2: Work Breakdown Structure.

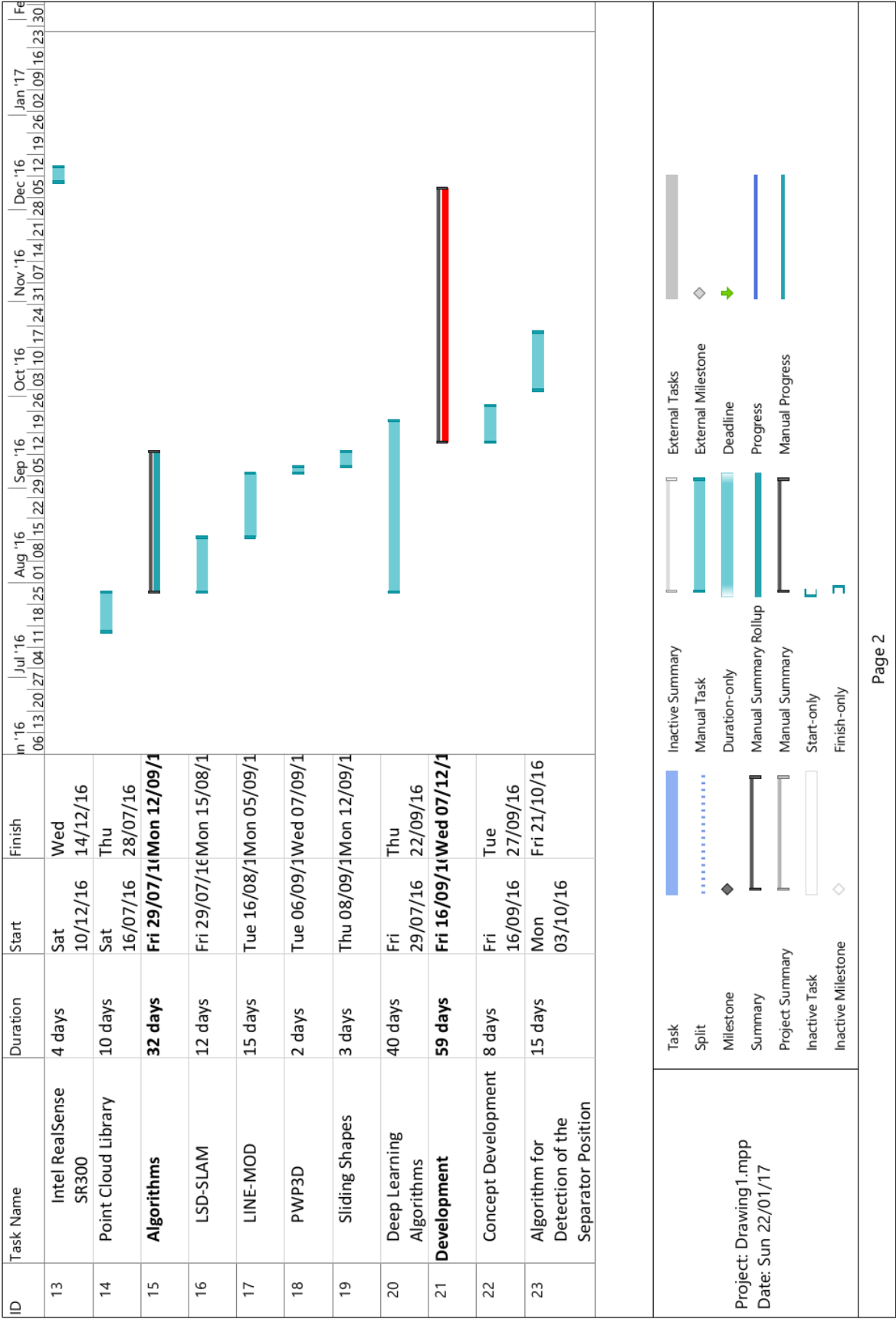


FIGURE 6.3: Work Breakdown Structure.

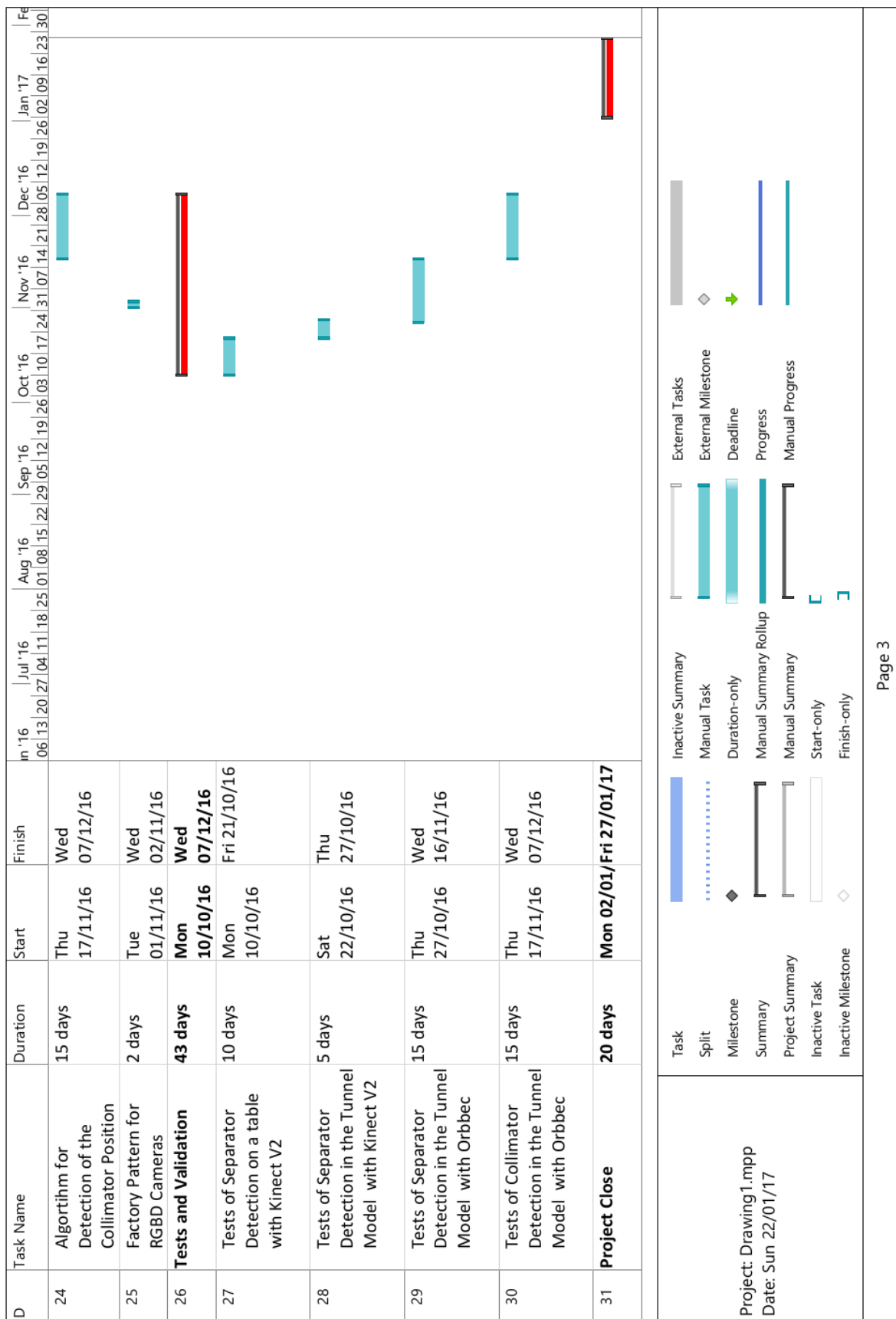


FIGURE 6.4: Work Breakdown Structure.

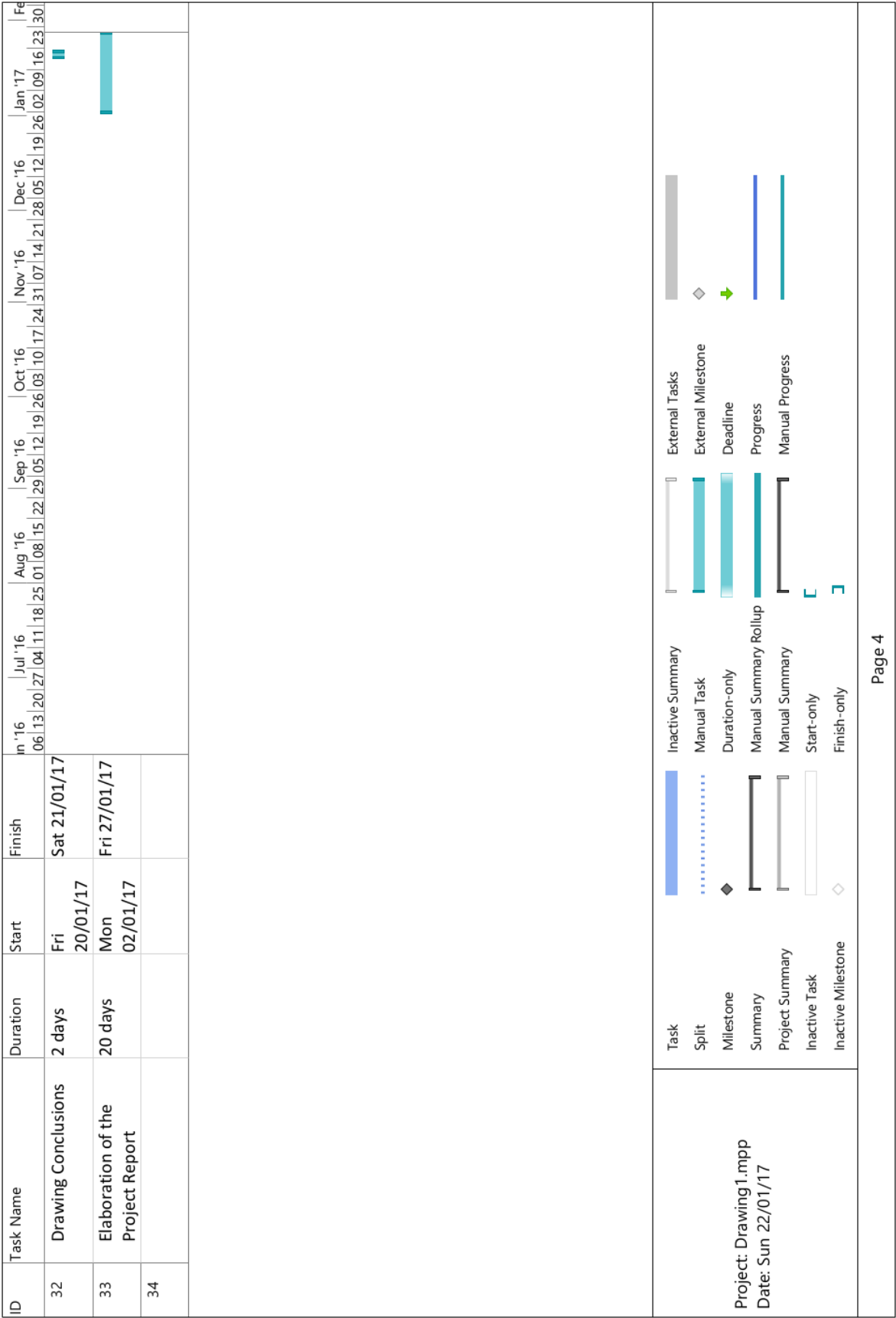


FIGURE 6.5: Work Breakdown Structure.

Bibliography

- [1] CERN. *About CERN*. [Online]. URL: <http://home.web.cern.ch/about>.
- [2] CERN. *The Large Hadron Collider*. [Online]. URL: <http://home.web.cern.ch/topics/large-hadron-collider>.
- [3] CERN. *The Linear Accelerator 2*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/linear-accelerator-2>.
- [4] CERN. *The Proton Synchrotron Booster*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/proton-synchrotron-booster>.
- [5] CERN. *The Proton Synchrotron*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/proton-synchrotron>.
- [6] CERN. *The Super Proton Synchrotron*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/super-proton-synchrotron>.
- [7] CERN. *The discovery of 'heavy light'*. [Online]. URL: <http://cern-discoveries.web.cern.ch/cern-discoveries/courier/heavylight/heavylight.html>.
- [8] CERN. *ALICE*. [Online]. URL: <http://home.web.cern.ch/about/experiments/alice>.
- [9] CERN. *ATLAS*. [Online]. URL: <http://home.web.cern.ch/about/experiments/atlas>.
- [10] CERN. *CMS*. [Online]. URL: <http://home.web.cern.ch/about/experiments/cms>.
- [11] CERN. *LHCb*. [Online]. URL: <http://home.web.cern.ch/about/experiments/lhcb>.

-
- [12] CERN. *Pulling together: Superconducting electromagnets* This content is archived on the CERN Document Server. [Online]. URL: <http://home.web.cern.ch/about/engineering/pulling-together-superconducting-electromagnets>.
- [13] D Manglunki. "The CERN Control Centre: Setting Standards for the 21st Century". In: *RPPB01, ICALEPCS 7* (). URL: <https://accelconf.web.cern.ch/accelconf/ica07/PAPERS/RPPB01.PDF>.
- [14] CERN. *The Antiproton Decelerator*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/antiproton-decelerator>.
- [15] CERN. *ISOLDE*. [Online]. URL: <http://home.web.cern.ch/about/experiments/isolde>.
- [16] CERN. *CERN Neutrinos to Gran Sasso*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/cern-neutrinos-gran-sasso>.
- [17] CERN. *The Compact Linear Collider*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/compact-linear-collider>.
- [18] CERN. *nTOF*. [Online]. URL: <http://home.web.cern.ch/about/experiments/ntof>.
- [19] CERN. *Linear Accelerator 3*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/linear-accelerator-3>.
- [20] CERN. *The Low Energy Ion Ring*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/low-energy-ion-ring>.
- [21] CERN. *Long shutdown 1: Exciting times ahead*. [Online]. URL: <http://home.web.cern.ch/about/updates/2013/02/long-shutdown-1-exciting-times-ahead>.
- [22] CERN. *Linear Accelerator 4*. [Online]. URL: <http://home.web.cern.ch/about/accelerators/linear-accelerator-4>.
- [23] ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the {ATLAS} detector at the {LHC}". In: *Physics Letters B* 716.1 (2012), pp. 1 –29. ISSN: 0370-2693. DOI: <http://dx.doi.org/10.1016/j>.

- physletb.2012.08.020. URL: <http://www.sciencedirect.com/science/article/pii/S037026931200857X>.
- [24] S. Weinberg. *The Quantum Theory of Fields*. The Quantum Theory of Fields 3 Volume Hardback Set v. 2. Cambridge University Press, 1995. ISBN: 9780521550024. URL: <https://books.google.es/books?id=sn9QvU5dmBQC>.
- [25] CERN. *The Totem Experiment*. [Online]. URL: <http://totem.web.cern.ch/Totem/>.
- [26] CERN. *LHCf*. [Online]. URL: <http://home.web.cern.ch/about/experiments/lhcf>.
- [27] CERN. *The Monopole & Exotics Detector at the LHC*. [Online]. URL: <http://moedal.web.cern.ch/>.
- [28] CERN. *Radiofrequency Cavities*. URL: <http://home.web.cern.ch/about/engineering/radiofrequency-cavities>.
- [29] J Uythoven et al. "Results from the LHC Beam Dump Reliability Run". In: LHC-PROJECT-Report-1120. CERN-LHC-PROJECT-Report-1120 (2008), 4 p. URL: <https://cds.cern.ch/record/1123695>.
- [30] RW Assmann et al. "The final collimation system for the LHC". In: *Proceedings of EPAC 2006*. 2006.
- [31] CERN. *Introduction to the LHC Collimation System*. URL: http://lhcollimation-project.web.cern.ch/lhcollimation-project/lhc_collimation_introduction.php.
- [32] CERN. *Radiation at CERN*. URL: <http://voisins.web.cern.ch/en/radiation>.
- [33] CERN. *Sievert*. URL: <http://goldbook.iupac.org/S05658.html>.
- [34] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer-Verlag London, 2011. ISBN: 978-1-84882-935-0. DOI: 10.1007/978-1-84882-935-0.

- [35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* 25. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [36] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009. URL: http://www.image-net.org/papers/imagenet_cvpr09.pdf.
- [37] IDS UI-2220SE-C-HQ Rev.3 Camera Specification. [Online]. URL: <https://www.1stvision.com/cameras/IDS/dataman/UI-2220SE-C-HQ%20Rev.3.pdf>.
- [38] *Real Time pose estimation of a textured object*. [Online].
- [39] Autodesk. *Autodesk 123D Catch*. [Online]. URL: <http://www.123dapp.com/catch/>.
- [40] Wikipedia. *LiDAR*. URL: <https://en.wikipedia.org/wiki/Lidar>.
- [41] Velodyne. *Velodyne LiDAR HDL-32E*. URL: <http://velodynelidar.com/hdl-32e.html>.
- [42] Ocular Robotics. *Ocular Robotics' RobotEye RE05 3D LIDAR*. URL: <http://www.ocularrobotics.com/products/lidar/re05/>.
- [43] Wikipedia. *Structured light*. [Online]. URL: https://en.wikipedia.org/wiki/Structured_light.
- [44] Wikipedia. *Time-of-flight camera*. [Online]. URL: https://en.wikipedia.org/wiki/Time-of-flight_camera.
- [45] Michele Pirovano. "KinFu – an open source implementation of Kinect Fusion". In: 2012. URL: <https://homes.di.unimi.it/borghese/Teaching/IntelligentSystems/Documents/PirovanoMichele-VisualReconstructionReport.pdf>.
- [46] Otmar Hilliges David Molyneaux Richard Newcombe Pushmeet Kohli Jamie Shotton Steve Hodges Dustin Freeman Andrew Davison Andrew Fitzgibbon Shahram Izadi David Kim. "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera". In: *ACM*, 2011, pp. 559–568. ISBN: 978-1-4503-0716-1. URL: <https://www.microsoft.com/en-us/research/publication/>

- kinectfusion-real-time-3d-reconstruction-and-interaction-using-a-moving-depth-camera/.
- [47] Thomas Whelan et al. "ElasticFusion: Dense SLAM Without A Pose Graph". In: *Proceedings of Robotics: Science and Systems*. Rome, Italy, 2015. DOI: 10.15607/RSS.2015.XI.001.
- [48] Microsoft. *Kinect hardware*. [Online]. URL: <https://developer.microsoft.com/en-us/windows/kinect/hardware>.
- [49] Lingzhu Xiang et al. *libfreenect2: Release 0.2*. Apr. 2016. DOI: 10.5281/zenodo.50641. URL: <https://doi.org/10.5281/zenodo.50641>.
- [50] Intel. *Intel RealSense Camera R200 Datasheet*. [Online]. URL: <https://software.intel.com/sites/default/files/managed/d7/a9/realsense-camera-r200-product-datasheet.pdf>.
- [51] Intel. *Intel RealSense Camera SR300 Datasheet*. [Online]. URL: <https://software.intel.com/sites/default/files/managed/0c/ec/realsense-sr300-product-datasheet-rev-1-0.pdf>.
- [52] Orbbec. *Orbbec. How our 3D camera works*. [Online]. URL: <https://orbbec3d.com/products/>.
- [53] Orbbec. *Orbbec Astra Pro*. [Online]. URL: <https://orbbec3d.com/product-astra-pro/>.
- [54] G.R. Bradski and V. Pisarevsky. "Intel's Computer Vision Library: applications in calibration, stereo segmentation, tracking, gesture, face and object recognition". In: *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition*, (2000), pp. 796–797. DOI: 10.1109/CVPR.2000.854964.
- [55] *OpenCV web page*. [Online]. URL: <http://opencv.org/>.
- [56] Willow. *Point Cloud Library*. [Online]. URL: <http://pointclouds.org/>.
- [57] Wikipedia. *Point Cloud Library*. [Online]. URL: https://en.wikipedia.org/wiki/Point_Cloud_Library.

- [58] Zhengyou Zhang. "Iterative point matching for registration of free-form curves and surfaces". In: *International Journal of Computer Vision* 13.2 (1994), pp. 119–152. ISSN: 1573-1405. DOI: 10.1007/BF01427149. URL: <http://dx.doi.org/10.1007/BF01427149>.
- [59] Point Cloud Library. *How to use iterative closest point*. URL: http://pointclouds.org/documentation/tutorials/iterative_closest_point.php.
- [60] Jan Knopp et al. "Hough Transform and 3D SURF for Robust Three Dimensional Classification". In: *Proceedings of the 11th European Conference on Computer Vision: Part VI. ECCV'10*. Heraklion, Crete, Greece: Springer-Verlag, 2010, pp. 589–602. ISBN: 3-642-15566-9, 978-3-642-15566-6. URL: <http://dl.acm.org/citation.cfm?id=1888212.1888257>.
- [61] Point Cloud Library. *Implicit Shape Model*. URL: http://pointclouds.org/documentation/tutorials/implicit_shape_model.php.
- [62] Point Cloud Library. *3D Object Recognition based on Correspondence Grouping*. URL: http://pointclouds.org/documentation/tutorials/correspondence_grouping.php.
- [63] Point Cloud Library. *Hypothesis Verification for 3D Object Recognition*. URL: http://pointclouds.org/documentation/tutorials/global_hypothesis_verification.php.
- [64] Jakob Engel, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-Scale Direct Monocular SLAM". In: *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 834–849. ISBN: 978-3-319-10605-2. DOI: 10.1007/978-3-319-10605-2_54. URL: http://dx.doi.org/10.1007/978-3-319-10605-2_54.
- [65] D. Caruso, J. Engel, and D. Cremers. "Large-Scale Direct SLAM for Omnidirectional Cameras". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2015. DOI: 10.1109/IROS.2015.7353366. URL: http://vision.in.tum.de/_media/spezial/bib/caruso2015_omni_lsdsлам.pdf.

-
- [66] TUM-vision. *LSD-SLAM, Implementation in ROS [Online]*. URL: https://github.com/tum-vision/lsd_slam.
- [67] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5. URL: <http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf>.
- [68] J. Engel, V. Koltun, and D. Cremers. "Direct Sparse Odometry". In: *arXiv:1607.02565*. 2016. URL: http://vision.in.tum.de/_media/spezial/bib/engel2016dso.pdf.
- [69] J. Engel, V. Usenko, and D. Cremers. "A Photometrically Calibrated Benchmark For Monocular Visual Odometry". In: *arXiv:1607.02555*. 2016. URL: http://vision.in.tum.de/_media/spezial/bib/engel2016monodataset.pdf.
- [70] S. Hinterstoisser S. Holzer et al. "Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Cluttered Scenes". In: (2011). URL: <http://campar.in.tum.de/pub/hinterstoisser2011linemod/hinterstoisser2011linemod.pdf>.
- [71] S. Hinterstoisser and S Holzer. *Object Detection – LINEMOD [Online]*. URL: http://www.pointclouds.org/assets/cvpr2012/PCL_Object_Detection_LINEMOD.pdf.
- [72] Willow Garage. *Object Recognition Using Linemod [Online]*. URL: https://wg-perception.github.io/ork_tutorials/tutorial03/tutorial.html.
- [73] Apache. *CouchDB [Online]*. URL: https://wg-perception.github.io/ork_tutorials/tutorial03/tutorial.html.
- [74] Thiemo Wiedemeyer. *IAI Kinect2. Github. [Online]*. URL: https://github.com/code-iai/iai_kinect2.
- [75] Willow Garage. *Texture Object Detection, TOD. [Online]*. URL: <http://wg-perception.github.io/tod/index.html#tod>.
- [76] Willow Garage. *Tabletop Object Recognition. [Online]*. URL: <http://wg-perception.github.io/tabletop/index.html#tabletop>.

- [77] Victor Prisacariu and Ian Reid. “PWP3D: Real-time segmentation and tracking of 3D objects”. In: *Proceedings of the 20th British Machine Vision Conference*. 2009. URL: http://www.robots.ox.ac.uk/~lav/Papers/prisacariu_reid_bmvc2009/prisacariu_reid_bmvc2009.pdf.
- [78] Shuran Song and Jianxiong Xiao. “Sliding Shapes for 3D Object Detection in Depth Images”. In: *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 634–651. ISBN: 978-3-319-10599-4. DOI: [10.1007/978-3-319-10599-4_41](https://doi.org/10.1007/978-3-319-10599-4_41). URL: http://dx.doi.org/10.1007/978-3-319-10599-4_41.
- [79] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Deep Learning”. Book in preparation for MIT Press. 2016. URL: <http://www.deeplearningbook.org>.
- [80] Andy Zeng et al. *Multi-view Self-supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge*. 2016. arXiv: [1609.09475](https://arxiv.org/abs/1609.09475).
- [81] *Official website of amazon picking challenge*. [Online]. URL: <https://www.amazonrobotics.com>.
- [82] Shuran Song and Jianxiong Xiao. “Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images”. In: 2016. URL: <http://dss.cs.princeton.edu/>.
- [83] DanielGM. *CloudCompare*. URL: <http://www.danielgm.net/cc/>.
- [84] Kitware. *Paraview*. URL: <http://www.paraview.org/>.
- [85] Kitware. *Interactive Visualization of Google Project Tango Data with ParaView*. URL: <https://blog.kitware.com/interactive-visualization-of-google-project-tango-data-with-paraview/>.
- [86] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.
- [87] Quoc Le et al. “Building high-level features using large scale unsupervised learning”. In: *International Conference in Machine Learning*. 2012.
- [88] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). URL: <http://arxiv.org/abs/1409.4842>.

-
- [89] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). URL: <http://arxiv.org/abs/1502.03167>.
- [90] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). URL: <http://arxiv.org/abs/1512.00567>.
- [91] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). URL: <http://arxiv.org/abs/1506.02640>.
- [92] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems* 28. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: <http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf>.
- [93] Joseph Redmon and Ali Farhadi. *YOLO: You Only Look Once*. URL: <http://pjreddie.com/darknet/yolo/>.
- [94] Pedro H. O. Pinheiro, Ronan Collobert, and Piotr Dollár. "Learning to Segment Object Candidates". In: *CoRR* abs/1506.06204 (2015). URL: <http://arxiv.org/abs/1506.06204>.
- [95] Pedro H. O. Pinheiro et al. "Learning to Refine Object Segments". In: *CoRR* abs/1603.08695 (2016). URL: <http://arxiv.org/abs/1603.08695>.
- [96] Piotr Dollar. *Segmenting and refining images with SharpMask*. URL: <https://code.facebook.com/posts/561187904071636/segmenting-and-refining-images-with-sharpmask/>.
- [97] Unity. *Unity 3D*. URL: <https://unity3d.com/>.
- [98] Jonathan Vanian (Fortune). *Facebook CEO Mark Zuckerberg To Spend Billions More On Virtual Reality*. URL: <http://fortune.com/2017/01/18/facebook-mark-zuckerberg-virtual-reality-billions/>.

Appendix A

Budget

The project costs are divided between material costs and human resources costs.

CERN is a large scientific institution, so there are various costs linked to each person. The budget for this project will remain simple, only for human resources the salary received for six months will be indicated in the Table A.1. The salary was given in Swiss Francs (1.500 CHF), which in Euros is 1.398,16€

Trainee Salary	Number of Months	Total Cost of Human Resources
1398,16€	6	8.388,96€

TABLE A.1: Costs of Human Resources.

Within the material cost we differentiate between the cost of the material purchased and general expenses.

The costs of material purchased, correspond to the acquisition of a computer equipment and cameras necessary to carry out this project. The breakdown of these costs is given in the Table A.2.

Hardware	Company	Price	Units	Total
Precision Tower serie 3000 (3420, 3620)	Dell	950,60€	1	950,60€
Mouse	Dell	15,19€	1	15,19€
Keyboard KB216	Dell	15,79€	1	15,79€
Monitor 20" E2016	Dell	95,20€	1	95,20€
Kinect V2	Microsoft	160,00€	1	160€
Orbbec Astra Pro	Orbbec	150,00€	1	150,00€
RealSense Robotic Development Kit	Intel	349,00€	1	349,00€
RealSense SR300	Intel	159,00€	1	149,00€
			TOTAL:	1.929,78€

TABLE A.2: Costs of Material.

We will understand as general expenses, all expenses generated due to the development of the project, except for the human resources costs, and costs of the material purchased, including expenses such as electricity consumption, office supplies and the use of consumables. To represent these costs, an approximation of these costs are made with a coefficient of 15% with respect to the costs of Human Resources and the material acquired, giving a total result of 1.547,81€.

The total budget of the project remaining as shown in the Table [A.3](#).

Type of Cost	Cost
Cost of Human Resources	8.388,96€
Costs of Material	1.929,78€
General Expenses	1.547,81€
Total Cost of the Project:	11.866,55€

TABLE A.3: Total Cost of the Project.