Alma Mater Studiorum · Università di Bologna

ENGINEERING SCHOOL

DEI The Department of Electrical, Electronic and Information Engineering

AUTOMATION ENGINEERING'S MASTER DEGREE

MASTER THESIS
in
Robotics and Computer Vision

CONVOLUTIONAL NEURAL NETWORKS FOR BIN PICKING APPLICATIONS WITH A COLLABORATIVE ROBOT AND A 2D CAMERA

Supervisor:
Chiar.mo Prof.
Gianluca Palli

Co-Supervisor:
Graziano De Capua

Presented by:
Alberto Di Stefano

Session II Academic year 2020-2021

Abstract

In bin picking applications, robots manipulate randomized objects placed in a bin. For that, the objects must be located before picking. The procedure of localization relies heavily on data from cameras. In the literature many approaches are available by means of the traditional computer vision approaches that work very well in an industrial scenario, while for more challenging situations artificial intelligence must be adopted. So, this thesis addresses this problem by presenting a possible solution for detecting objects and estimating their poses. Many convolutional neural networks exist to detect objects but still a reliable approach for estimating the pose of casual objects does not exist, in this thesis the problem will be solved for a certain category of objects, that is objects in which the orientation is clear from a purely geometrical point of view. In addition, the problem will be solved by using a collaborative robot, so that the task can be performed in a shared environment with people without hurting them.

Contents

List of Figures

Acronyms List

CNN	Convolutional Neural Networks
YOLO	You Only Look Once
UR	Universal Robot
ROS	Robot Operating System
HMI	Human Machine Interface
RTDE	Real Time Data Exchange
SIFT	scale-invariant feature transform
GUI	Graphical User Interface
URDF	Unified Robotic Description Format

Chapter 1

Introduction

1.1 Motivations

The idea behind this thesis come from the need of more reliable computer vision system for detecting objects, in the literature many valuable techniques exist but they work well in controlled environment where light, shadow, shape is always the same. In the next pages, instead, a novel approach will be introduced to explore new possible scenarios based on artificial intelligence, by using convolutional neural networks.

1.2 Constructa Sistemi S.r.l

Constructa Sistemi S.r.l is the company in which this project has been carried out. Constructa has been operating in the factory of automation industry since 1984. They found their identity in the 90s when they got involved in the household appliances sector where, in addition to operating as a supplier of assembly lines and testing, they soon became a leader in the

production of silicone coating systems. In the years that followed, while continuing to operate in the household appliances sector, Constructa has expanded its operating space, completed projects and supplied for a vast number of varied sectors including: [3]

1. Gluing applications
2. Assembly lines
3. Automatic testing
4. Robotized handling
5. Collaborative robots
6. Artificial vision

Chapter 2

Universal Robot 10e

The project can be mainly divided into two parts, the computer vision part and the control of the robot, in this chapter the latter will be discussed.

2.1 The robot

For the application a universal robot 10e is used. The UR10e is an extraordinarily versatile collaborative industrial robot, delivering both high payload (12.5 kg) lift and long reach (1300mm) which makes it well suited for a wide range of applications in machine tending, palletizing, and packaging [19]. They have many benefits, like:

1. Flexibility, they can be moved between tasks quickly and are able to reuse programs for recurrent tasks, giving customers the flexibility to automate multiple manual tasks within one production facility even with just one cobot;

2. Safety, these cobots are able to take over strenuous tasks in danger-

ous or dull environments to minimize risk in the production process.

3. Easy programming, a user friendly and intuitive teach pendant allow operators to program a cobot by moving its arms to the desired waypoints, or simply using drag-and-drop functions on a touchscreen tablet.

To pick up the objects a tool was necessary and usually for bin picking application the best one is the vacuum because it can pick up an object even if it has in its neighborhood obstacles, but at the same time a vacuum has not a big strength and only some kind of objects can be grasped by using this kind of tools. For these reasons a gripper is used. A gripper RG6 of OnRobot, this gripper has 6 kg of payload and an opening of 150mm.

As already said a very user-friendly HMI is present for this kind of robot, but in this case every step of the project will be carried on by using ROS, in this way it is possible to control the robot (fig. 2.1), the gripper (fig. 2.2) and the camera simultaneously.

Figure 2.1: The Universal Robot 10e

Figure 2.2: OnRobot RG6 gripper

2.2 ROS

ROS stands for Robot Operating System, it is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [16].
ROS is an open-source, meta-operating system for robots. It provides the services expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers [2].
The reasons behind the choice of using ROS are a lot, like:

- ROS is general purposes, it can be used for manipulator, mobile

robots or drones;

- There are tons of available packages that are free and ready to be used;
- Multiple robots can be controlled simultaneously with it;
- Code can be written either with python or C++;
- It uses few resources and space, indeed it can also be used in micro controller like Raspberry Pi;

Given the above reasons also in the following project ROS will be adopted to control each part of the application. ROS Melodic will be used and all the script will be written in Python.

Since the great spread of ROS in the industry almost for each robot on the market there is already the driver for interconnecting it with ROS, so also for the Universal Robot family there is the related driver, that is the *ur_robot_driver*. This driver allows communication between the robot and ROS, so that everything can be controlled in a standard way. The protocol that is used is the Real Time Data Exchange (RTDE), this interface is available on all the robots of the universal robot series and it provides a way to synchronize external applications with the UR controller over a standard TCP/IP connection, without breaking any real-time properties of the UR controller [17]. The interface makes available a great number of ports, each one with a specific role, some are for listening from the robot to the PC and other are to send commands to the robot, by knowing the protocol and the ports to be used it is possible to control completely the robot.

The ROS driver does nothing different from what is reported above, that is exchanging information between the robot and the PC in a bidirectional way, moreover it makes available many commands and information on the ROS platform. The core of the ROS framework is the nodes, the topics and the messages. The nodes are basically executable scripts that represent a subprogram inside the ROS application, so they are the brain. The messages are a simple data structure, comprising typed fields, e.g. the pose message contains information about the position and orientation of the robot in the space and it is standardized so that every robot library sends the same type of message regarding the pose. Finally the topics are named buses over which nodes exchange messages, nodes can be subscribed to a topic or they can write on it, e.g. whenever a node write a pose message to the topic */follow_joint_trajectory/goal* another node that is subscribed to this topic will send the pose to the robot and it will start moving.

Basically, this is how ROS framework works, so every information the robot sends to the PC is modified and published on a topic to be available to the user, while other topics are free to be used by nodes to send commands to the robot.

All the parts of this application are controlled by ROS, so also for controlling the camera a driver must be used, so that, for example, it is possible to get the current image by asking it to a topic. More difficult is communication with the gripper, because it is not supported by the Universal Robot Driver; to solve the problem the gripper is controlled by using a set of digital outputs on the robot. For example, if the robot must close the gripper the ROS node will set the digital output 2 to *high* and a thread running on the robot will detect the event and close the gripper; once the action is done the output is set to low. In the same fashion the opening

procedure is done.

2.3 Hand-Eye Calibration

In robotics and mathematics, the hand eye calibration problem (also called the robot-sensor or robot-world calibration problem) is the problem of determining the transformation between a robot end-effector and a camera or between a robot base and the world coordinate system.

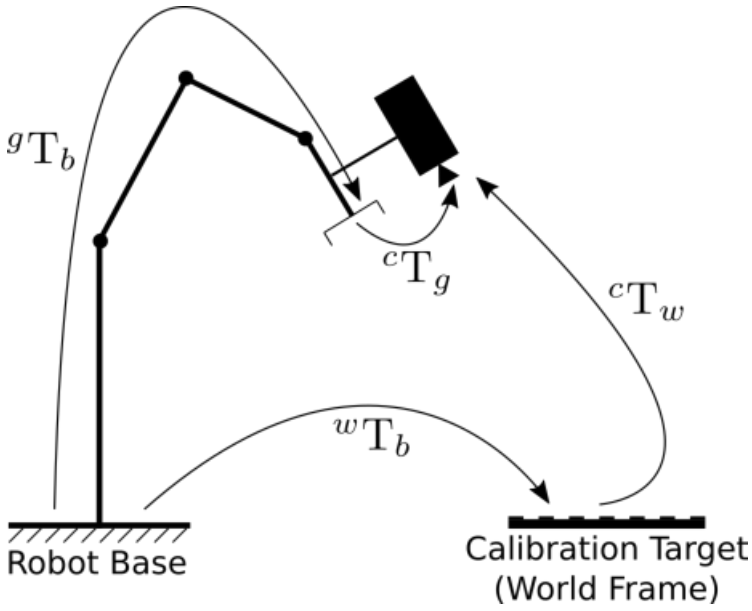


Figure 2.3: Hand eye problem

The problem of hand eye calibration takes the form $AX = ZB$, in which:

1. A is the transformation of the target frame with respect the camera frame (${}^{c}T_{w}$);
2. X is the transformation of the base with respect the target frame (${}^{w}T_{b}$);
3. Z is the transformation of the gripper with respect the camera (${}^{c}T_{g}$);
4. B is the transformation of the base with respect the gripper (${}^{g}T_{b}$).

To solve this problem at least three measurements must be taken, but to obtain good results the more samples are used the lower the error is. The idea to solve the problem is to move the robot step by step and for each positions do two operations, the first one is to register the transformation of the gripper with respect the base (${}^{g}T_{b}$) that is easily provided by the robot itself by knowing the joint configuration. The second one is to obtain the transformation between the target frame and the camera (${}^{c}T_{w}$), that is gathered by using a chessboard with a known pattern and with known size (this topic is studied in 3.1).
Once at least three measurements of these transformations are gathered the problem can be solved, but in this kind of task many errors may arise, due to the camera distortions or the errors in the robot joints positions, so more than three measurements must be taken, so that the computation of the solution become an optimization problem to minimize the error [14].

Traditionally this task is carried on by hand, there is an operator that manually moves the robot and for each new position records the configuration, since this task should be performed for each new robot a new ROS script has been written to do the procedure automatically. A chessboard 8x5 has been used for the operation, by knowing the intrinsic parameters

of the camera it is possible for each position to know the extrinsic ones based on the recognition of the pattern. This operation, that is the identification of the pattern position in the scene will be seen in 3.1.

So, the ROS node takes as input only the position above the center of the pattern and then automatically moves the robot in 30 positions recording for each one the pose of the pattern with respect the camera and the pose of the tool with respect the base. Once this operation is terminated the hand eye calibration can be performed to obtain the rigid transformation between the camera and the base; for this operation theoretically 3 positions are enough, but to deal with error it is better to use more poses, so that the problem becomes an optimization one in which the error must be minimized. Then the computation will be made with the OpenCv function *hand_eye_calibration*, once this solution is gathered it is possible to know for each point of the surface framed by the camera the position of that point with the respect the base of the robot.

In this case, the hand eye procedure returned as result:

$$ {}^{c}T_{g} = \begin{bmatrix} 0.999 & 0.0150 & 0.00647 & -0.0141 \\ -0.0150 & 0.999 & -0.00248 & -0.0643 \\ -0.00651 & 0.00238 & 0.999 & 0.0959 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1} $$

By looking at the transformation of the camera with the respect the end effector of the robot (2.1) it can be observed that the orientation is defined by an almost identity matrix, so there is no angle offset between the camera and the robot end effector, while there is a linear offset of $\simeq 9.5$

cm along the z axis, of $\simeq -6.4$ cm along y and $\simeq -1.4$ cm along x.
By knowing these information the whole hand eye problem is solved.

2.4 Control of the robot

The control of the robot is one of the main tasks for the application, because if the robot is not controlled properly the task will fail. However, the task is not complicated, but some decisions must be made; the idea is simple: the robot must go to a shot position, find the piece, grasp it and release it in a desired configuration. Three main approaches were available to perform the task and each one has been tested and has its own peculiarities and disadvantages.
The simplest one is based on one shot position and has not an intermediate step to refine the grasping, in this case the robot is in its fastest configuration, but it has some limitations. By using only one shot position and having a 2D camera it is not possible to have a 3D information about the scene, so it must use a predefined depth to grasp objects. Furthermore, without using an intermediate station to refine the pose it is not possible to have a very good placement of the object in the final configuration, because the scene has other objects and there may be occlusions or distortions. So this technique may be good for all those cases in which speed is an important requirement and it does not require a precise placement.
The second approach is with the intermediate step and with only one shot position, so still no information about depth is available, but it is possible to obtain a much clearer pose estimation. The idea is that in the main scene with the other object the robot detects the target one with a good approximation of the orientation and of the center point, but it happens that

sometimes the picking point is above or below the real center. In these cases, after the object has been picked up it might be useful to release it in a second position and take another shot to refine the grasping.
For last the more complete approach is the one with two shot positions and an intermediate station, in this case by using two photos it is possible to reconstruct better the scene and to obtain information about the depth at which the objects are, the drawback is that this operation will spend almost three times more than the first one.
These are the three main scenarios carried out by the robot, so for each one a script has been made, even if the first approach has not been widely used because in this project the final placement was important. For the first test screws will be used and the scope is to release them in vertical position with the head on a planar surface, so just a couple of millimeters of error may create several problems.

2.5 Moving The robot

Motion planning is always one of the main concern when dealing with robots, that is a sub-field of automation that deals with moving individual parts of a machine in a controlled manner. This is especially helpful in applications such as production lines, where power, efficiency, and accuracy of movement are of vital importance.
In robotics motion planning is very important, every time that a manipulator must perform an action a motion planning operation must be carried out. There are different types of target operations that can be useful:

- Position control;
- Velocity control;

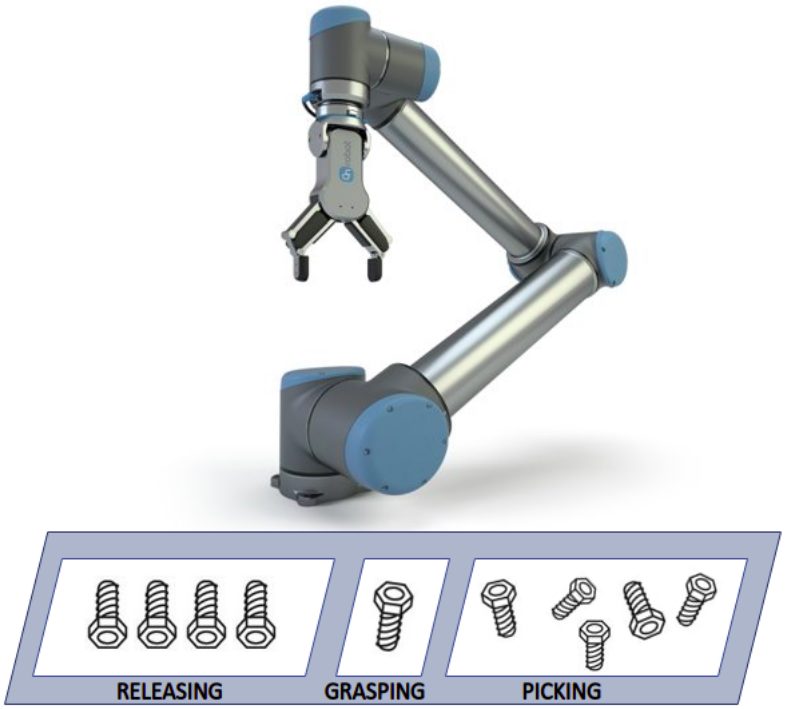


Figure 2.4: Setup of the whole application, on the right the pieces displaced randomly can be observed, in the center the intermediate station for the refine the grasping, on the left the final configuration of the pieces.

- Force control;
- Impedance control.

Depending on the task one of these actions must be controlled, if not the robot cannot move. To actuate this kind of actions some information about the robot must be available, indeed in almost every robot there are motors and encoders to know and change the joint configuration, while in the reality the most of the times it is needed to move the robot to a position in the world frame. So, every time that the robot must be moved a conversion between the joint and the world frame must be done; to do it the inverse kinematics must be known to compute the correspondence between a point in the 3D world and the robot configuration in the joint space.

Assuming a linear movement between two points is required the steps that the motion planner must do are the following ones:

1. Compute the linear path between the two points, that is, given a step (e.g. 10 mm), find all the points at distance 10 mm in the 3D world that are in the between the initial and the final point;
2. Transform each one of these points from the 3D world to the joint space, that is associate the 3D point to a vector of 6 elements (for a 6 DOF robot), each one corresponding to the value of one joint;
3. Send all points to the robot.

To pursue this task many tools exist, in particular MoveIt is very used across the world. It is a free software that runs on top of ROS and it is a

primary source of the functionality for manipulation (and mobile manipulation) on it [13].
MoveIt can handle different types of problems like trajectory generation, collision avoidance, grasping actions and other ones; in this project the trajectory generation tool has been used. This problem is not too complicated but having a software that easily do it could be very useful.
MoveIt has a graphical interface (figure 2.5) in which it is possible to define the geometry and the characteristics of the robot, to run the URDF model of the robot must be uploaded. The Unified Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot like the sensors, the motors, the scene, the kinematics and so on. By uploading this file in the GUI it is possible to add other information (e.g. constraints to avoid collision or positions in space) and once the setup is completed it is possible to download it. This file will be used during the motion planning of the robot to get the trajectories.

At run time to get the trajectory it is enough to send to MoveIt the type of trajectory, the initial point, the final one and the step with which the trajectory must be done, then it will compute the 3D points of the path and the joint configurations related to each point of the space by means of the inverse kinematics. Once everything is done the points are sent to the robot to start moving.

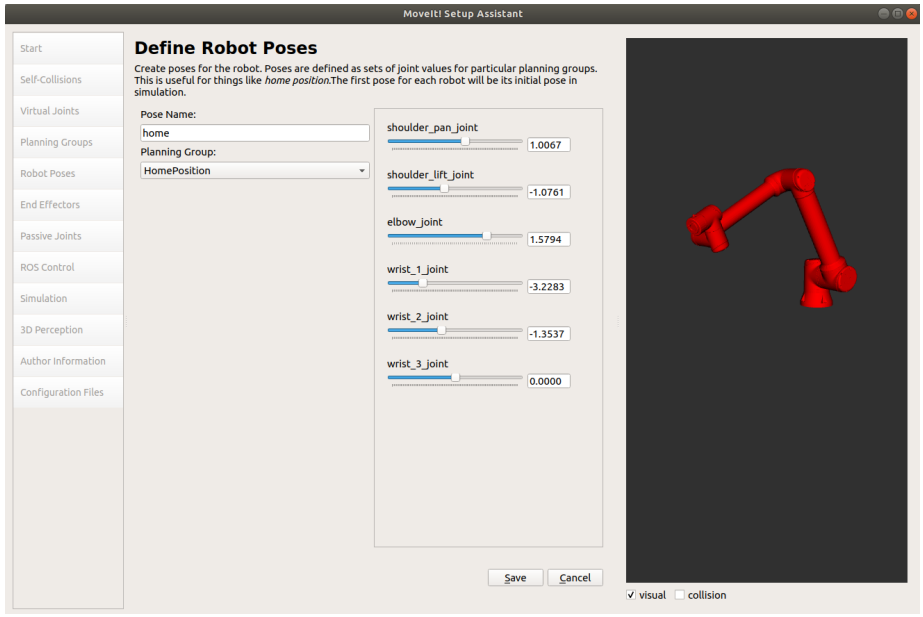


Figure 2.5: MoveIt setup assistant

Chapter 3

Computer Vision

In this chapter the computer vision part will be faced, two are the main tasks to be considered: the detection of objects and the estimation of the orientation and pose.

3.1 The Camera

The choice of the camera and the lens is very important when dealing with computer vision task, so in this section some considerations will be made. Depending on the applications 2D or 3D camera should be adopted, in particular the first one can give information along X and Y distances without any knowledge of depth, while the second one gives a complete knowledge about distances along X, Y and Z. The time consumption in gathering and elaborating the image is the biggest problem of 3D cameras, but sometimes they became fundamental, in the specific case of this task it depends, if the pieces rely on the same surface and there is not overlapping a 2D camera would be enough, while instead if pieces are place in a

Figure 3.1: 2D Camera Ueye

box randomly a 3D camera became essential.
Although the available camera for the project was a 2D camera and so this one will be used, but with a trick the depth problem will be solved too.
In particular the camera belongs to the IDS factory and the model is a UI-5480CP, it is fitted with the 5 megapixel CMOS sensor. About half an inch in size, the sensor delivers a resolution of 2560 x 1920 pixels as well as rolling and global start shutter features. The sensor is extraordinarily sensitive and is a real megapixel CCD replacement. The various shutter modes provide ideal parameters for every application to produce sharp, low-noise images [8].

Another important parameters is the choice of the lens, by looking at the distance and the area that must be framed; in this case a Kowa LV1214 has been used with 2/3" and focal length 12mm. With this setup the best trade off between the height of the robot and the maximum size of the area to place the objects could be found. [10]

Figure 3.2: Kowa lens 12mm

3.2 Camera calibration

Every time a camera is used for position estimation task is very important that it has been calibrated opportunely, if not the task is not feasible.
Camera calibration is the process whereby all parameters defining the camera model are estimated for a specific camera device.
The pinhole camera model is represented by the Perspective Projection Matrix (PPM), which in turn can be decomposed into 3 independent tokens: intrinsic parameter matrix (A), rotation matrix (R) and translation vector (T). Depending on the application, either the PPM only or also its independent components (A,R,T) need to be estimated.

Many camera calibration algorithms do exist. The basic process, though, relies always on setting up a linear system of equations given a set of known 3D-2D correspondences, so as to then solve for the unknown cam-

era parameters.

To obtain the required correspondences specific physical objects (referred to as calibration targets) having easily detectable features (such as e.g. chessboard or dot patterns) are typically deployed.

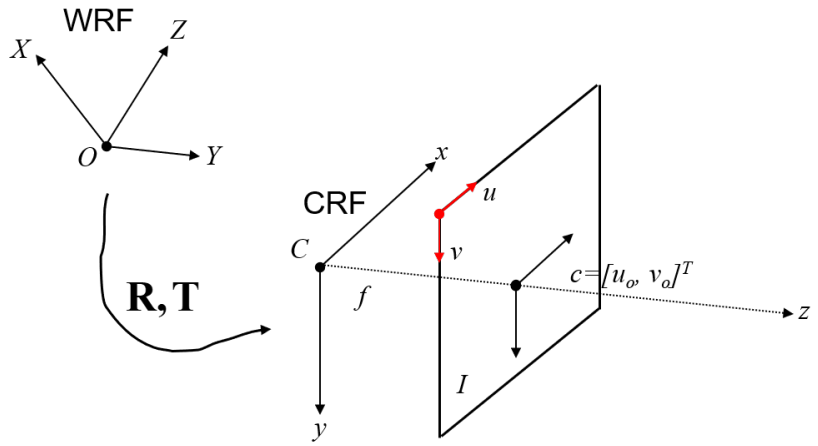


Figure 3.3: Camera representation

The general form of the Perspective Projection Matrix (PPM or P) can be thought of as encoding the position of the camera with respect to the world into G, the perspective projection carried out by a pinhole camera into the canonical $PPM[I|0]$ and, finally, the actual characteristics of the sensing device into A. Matrix A, which models the characteristics of the image sensing device, is called "Intrinsic Parameter Matrix".

$$A = \begin{bmatrix} a_u & 0 & u_0 \\ 0 & a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

In 3.4 the intrinsic matrix A is represented, where:

- f is the focal length of the pinhole system
- $k_u = \frac{1}{\delta u}$, $k_v = \frac{1}{\delta v}$ are, respectively, the inverse of the horizontal and vertical pixel size
- u_0, v_0 are the coordinates of the piercing point wrt the top-left corner

Intrinsic parameters are 5 but can be reduced in number by setting $a_u = f \cdot k_u$, $a_v = f \cdot k_v$ such quantities representing, respectively, the focal length expressed in horizontal and vertical pixel size.

The number of intrinsic parameters estimated by OpenCV is thus 4. The pixel size is usually provided in the camera data sheet: if it is known, the metric focal length can be recovered.

Matrix G, which encodes the position and orientation of the camera with respect to the World Reference Frame (WRF), is called Extrinsic Parameter Matrix.

- As a rotation matrix (3x3=9 entries) has indeed only 3 independent parameters (DOF), which correspond to the rotation angles around the axis of the RF, the total number of extrinsic parameter is 6 (3 translation parameters, 3 rotation parameters)
- Hence, the general form of the PPM can be thought of as encoding the position of the camera with respect to the world into G, the perspective projection carried out by a pinhole camera into the canonical $PPM[I|0]$ and, finally, the actual characteristics of the sensing device into A.

Camera calibration approaches can be split into two main categories:

- Those relying on a single image featuring several (at least 2) planes containing a known pattern.

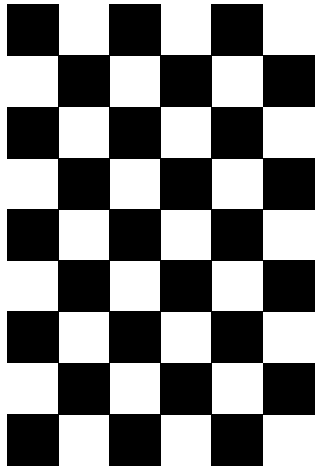


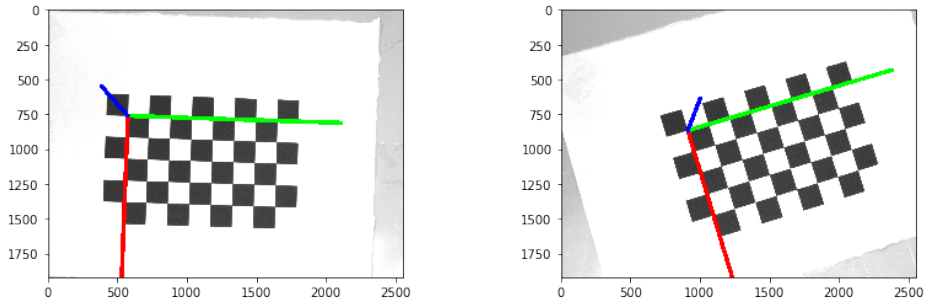
Figure 3.4: Chessboard used for camera calibration

- Those relying on several (at least 3) different images of one given planar pattern. (We are using this kind of approach!)

For solve the calibration problem the Zhang's method is used, in which a chessboard must be printed with a predefined number of corner (8x5 in this case) and size of each square (10mm). Once this operation is done the camera must take at least 3 photos of the pattern, but for minimizing the error several ones are taken, and for each image it searches for the upper left corner and knowing the real distance between each corner it's able to estimate the distortion in the image.

Once the calibration is terminated the output parameter are returned, in particular:

- Root Mean Square = 0.7584757



(a) (b)

Figure 3.5: Pattern used by camera calibration

- Camera Matrix = $\begin{bmatrix} 5.8221e+03 & 0.0000e+00 & 1.2842e+03 \\ 0.0000e+00 & 5.8267e+03 & 9.8479e+02 \\ 0.0000e+00 & 0.0000e+00 & 1.0000e+00 \end{bmatrix}$
- Distortion coefficients = $[-1.5848e-02 \; -2.8086e+00 \; 2.1425e-04 \; 3.8741e-03 \; 4.5146e+01]$

Then for each image also the rigid transformation between the camera and the pattern are provided, for example for the image 3.5a the output of the calibration problem is:

- Translation vector = $\begin{bmatrix} -46.234 & -14.420 & 377.778 \end{bmatrix}$ mm
- Rotation vector = $\begin{bmatrix} -0.0639 & -0.0294 & 0.0314 \end{bmatrix}$

And in particular this information will be used for the hand eye calibration of section 2.3.

3.3 Object detection

Object detection has been one of the major problem in computer vision, many different solutions have been taken in considerations. Nowadays two ways are mainly used: convolutional neural networks or classical computer vision algorithms.
The latter one will now be taken in consideration. This approach is based on finding features in the model image to be compared with the features of the sample images to find matches, the most famous one is SIFT. The features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images with efficient algorithms. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition. The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test. Following are the major stages of computation used to generate the set of image features:

1. Scale-space extrema detection: The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.

2. Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based

on measures of their stability.

3. Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

4. Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination. [12]

For image matching and recognition, SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors.
The keypoint descriptors (fig. 3.6) are highly distinctive, which allows a single feature to find its correct match with good probability in a large database of features. However, in a cluttered image, many features from the background will not have any correct match in the database, giving rise to many false matches in addition to the correct ones. [12]
The biggest advantage of SIFT is that few time is required to implement it, just one image is enough to obtain good results. The main problem instead is that it is very sensitive to the light change, shadow and occlusion; the time consumption is another issue that must be taken in consideration using this kind of approaches. SIFT has been tested for this specific task with not very good results, the problem is that when many equal objects

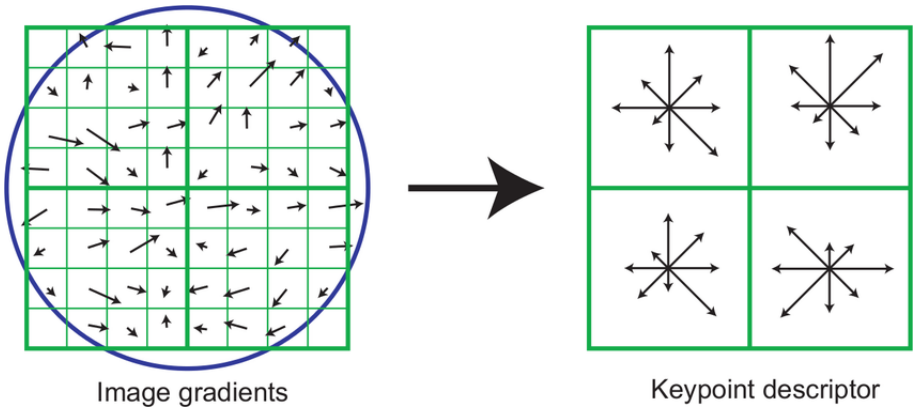


Figure 3.6: Structure of SIFT descriptor

are in the scene it is difficult for the algorithm to find correspondences with the geometry of the object.

The second typology of methods is the ones based on artificial intelligence, that nowadays is becoming the most spread. An artificial neural network (ANN) is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The ”signal” at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the in-

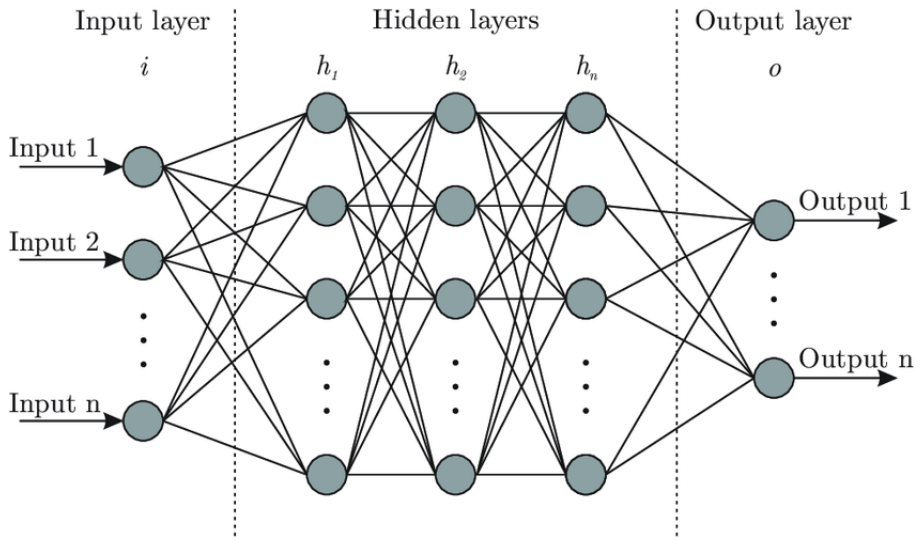


Figure 3.7: Structure of an artificial neural network

put layer), to the last layer (the output layer), possibly after traversing the layers multiple times (fig. 3.7).

Using ANN, image classification problems become difficult because 2-dimensional images need to be converted to 1-dimensional vectors. This increases the number of trainable parameters exponentially. Increasing trainable parameters takes storage and processing capability. For this reason in image classification convolutional neural networks (CNN) are more used, the main difference is that only the last layer of a CNN is fully connected whereas in ANN, each neuron is connected to every other neurons. Once the network has been created it must be trained, this operation is very time consuming because a very large amount of images must be gathered and very performing hardware should be used. The training phase is divided in two parts, the forward propagation, in which an image is used as input obtaining a certain output, and the backward propagation in which

the error between the true output and the estimated one is back propagated to refine the value of each layer.
Once this operation has been completed the test phase can begin to obtain some indexes about the performances of the net.

SIFT

- Easy and fast to be implemented;
- Not robust to light changes or shadow;
- Fast computation;
- Poor generalization;
- Not robust to non linear transformation;
- More relevant for identification tasks.

CNN

- Big image set is required;
- Lots of processing power and time is needed for implement the net;
- More relevant for classification and categorization tasks, has very good generalization abilities;
- Currently very popular model for image and video tasks.

3.4 YOLO

YOLO [15] is a convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. It has been the first net in which the detection and the classification problem has been treated in a unique

regression problem.
YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. For this reason it is extremely fast, one fast version works at 155 fps with a mean average precision (mAP=52.7%) on the database VOC07, while the base version works at 45 fps with mAP=63.4% on the same train set.
The network architecture is inspired by the GoogLeNet [18] model for image classification. The network has 24 convolutional layers followed by 2 fully connected layers (fig. 3.9). Instead of the inception modules used by GoogLeNet, a 1×1 reduction layers followed by 3×3 convolutional layers are used, similar to Lin et al [18]. The full network is shown in figure 3.8.

The separate components of object detection are unified into a single neural network. The network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means the network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real time speeds while maintaining high average precision. The system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object, formally we define confidence as $Pr(Object) \cdot IOU$.
If no object exists in that cell, the confidence scores should be zero. Oth-

Name	Filters	Output Dimension
conv 1	7 x 7 x 64, stride=2	224 x 224 x 64
Conv 1	7 x 7 x 64, stride=2	224 x 224 x 64
Max Pool 1	2 x 2, stride=2	112 x 112 x 64
Conv 2	3 x 3 x 192	112 x 112 x 192
Max Pool 2	2 x 2, stride=2	56 x 56 x 192
Conv 3	1 x 1 x 128	56 x 56 x 128
Conv 4	3 x 3 x 256	56 x 56 x 256
Conv 5	1 x 1 x 256	56 x 56 x 256
Conv 6	1 x 1 x 512	56 x 56 x 512
Max Pool 3	2 x 2, stride=2	28 x 28 x 512
Conv 7	1 x 1 x 256	28 x 28 x 256
Conv 8	3 x 3 x 512	28 x 28 x 512
Conv 9	1 x 1 x 256	28 x 28 x 256
Conv 10	3 x 3 x 512	28 x 28 x 512
Conv 11	1 x 1 x 256	28 x 28 x 256
Conv 12	3 x 3 x 512	28 x 28 x 512
Conv 13	1 x 1 x 256	28 x 28 x 256
Conv 14	3 x 3 x 512	28 x 28 x 512
Conv 15	1 x 1 x 512	28 x 28 x 512
Conv 16	3 x 3 x 1024	28 x 28 x 1024
Max Pool 4	2 x 2, stride=2	14 x 14 x 1024
Conv 17	1 x 1 x 512	14 x 14 x 512
Conv 18	3 x 3 x 1024	14 x 14 x 1024
Conv 19	1 x 1 x 512	14 x 14 x 512
Conv 20	3 x 3 x 1024	14 x 14 x 1024
Conv 21	3 x 3 x 1024	14 x 14 x 1024
Conv 22	3 x 3 x 1024, stride=2	7 x 7 x 1024
Conv 23	3 x 3 x 1024	7 x 7 x 1024
Conv 24	3 x 3 x 1024	7 x 7 x 1024
FC 1	-	4096
FC 2	-	7 x 7 x 30 (1470)

Figure 3.8: Layers of YOLO

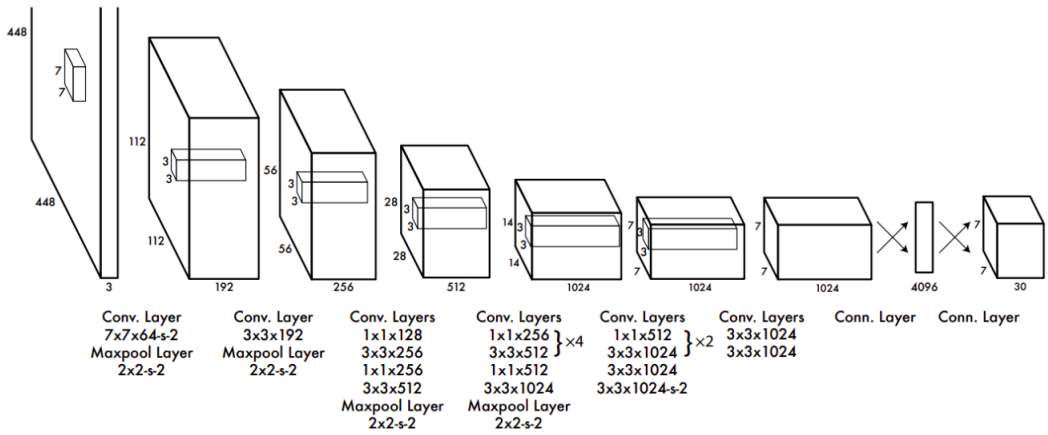


Figure 3.9: Structure of YOLO

erwise the confidence score should be equal to the intersection over union (IOU) between the predicted box and the ground truth.
Each bounding box consists of 5 value: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. W and H represent the width and height of the rectangle. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, *Pr(Class|Object)*. These probabilities are conditioned on the grid cell containing an object. Only one set of class probabilities is predicted per grid cell, regardless of the number of boxes B.

At test time the conditional class probabilities is multiplied by the in-

dividual box confidence predictions,

$$Pr(Class_i|Object) \cdot Pr(Object) \cdot IOU_{pred}^{truth} = Pr(Class_i) \cdot IOU_{pred}^{truth} \tag{3.2}$$

which gives class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object [15].
Many different ways are available for running YOLO, for coherency with the rest of the project OpenCV will be used through the DNN (Deep Neural Networks) module.

3.5 Training e test

Build the image set in the proper way is fundamental to obtain reliable results, indeed many problems could arise from a bad image set. One of the major problems that might arise is a lack of accuracy, the net is not able to detect objects, but at the same time avoiding overfitting is important. Overfitting happens when your model fits too well to the training set. It then becomes difficult for the model to generalize to new examples that were not in the training set. For example, your model recognizes specific images in your training set instead of general patterns.
Both these aspects depend on the choices made during the creation of the image set, here some rules that should be followed:

1. Collect images of the object from different angles and perspectives.
2. Gather images of the object in variable lighting conditions.
3. Gather images with different object sizes and distances for greater

variance.

4. Ensure your future input images are clearly visible. Otherwise, train the model to classify objects that are partially visible by using low visibility datapoints in your training dataset.

To avoid overfitting, then, it is important to use images enough different from one other, if not the net could be trained to recognize that specific image and not the pattern related to the target object. While to increase accuracy the most images as possible should be used without causing overfitting, so it would be important to consider all the possible angulation and conditions.

For this specific task, a slightly different approach has been chosen, that is because the environment in which the convolutional network should work is quite stable, so the same conditions are always maintained. The task is supposed to be used in a factory or a laboratory where the light is always the same and the objects are the same size. For this reason, a few images also give satisfactory results in terms of accuracy, but the bigger is the data set the better should be the results if the previous rules were followed.
As rule of thumb at least 100 images should be used for each label, in this specific case only one label is necessary and 219 images have been used for the training; the target of the application is to detect objects placed in a random way, so in the image set both single objects and multiple objects are present. All the photos have been taken with the same camera in the same environment. Once this operation was concluded the classification started, that is assigning to each image the information about the presence and the position of objects, in particular the idea is to have a text file for

each image in which the bounding boxes are defined. A bounding box is a rectangular that encapsulates the target object, it is defined by the position of the lower left corner and the width and height of the box. To simplify this operation some tools exist, their aim is to help users to draw the bounding boxes over the image and save the information in the text file, in particular *LabelImg* was used for this task [11], but in section 5.1 a new desktop application is developed to simplify the labelling process.

Once all the 219 images were classified the training phase started. By using convolutional neural networks like YOLO only the last layers usually must be trained, because these are pretrained nets that have already been trained with millions of images, so the longest part has already been made. For this kind of operation, suitable hardware should be used, like high performing GPUs, to speed up the process, in this case the cloud computing hardware of Google has been used. They made available on Google Colaboratory (also known as Colab) GPUs that can be used freely by whoever has an account, Colab is a free Jupyter notebook environment that runs in the cloud and stores its notebooks on Google Drive. By using this tool, the training phase with the cloud computing takes around 2.50 hours.

After the training phase has been terminated the evaluation of the network must be carried out, to do it many different tools are available. Mean average precision (mAP) is a very common tool to evaluate classification task, it relies on the concept of precision that defines how accurate the

predictions are. i.e., the percentage of the predictions are correct:

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

TP = True Positives (Predicted as positive as was correct)
FP = False Positives (Predicted as positive but was incorrect)

Recall is another important index, it measures how many true positives are found inside all the real positives, that is:

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

FN = False Negatives (Failed to predict an object that was there)

By knowing these two values the definition of Average Precision (AP) can be made as the area under the precision-recall curve, that can be seen in 3.10.

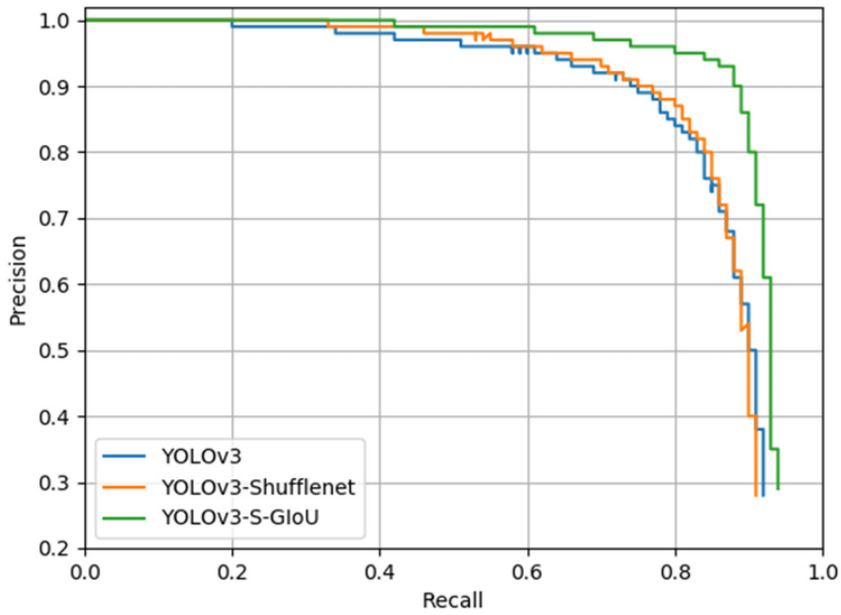


Figure 3.10: Precision-recall curve for a classifier.

When average precision is used for binary classifiers the definition is clear and unique, instead if localization is also of interest then another definition must be used. In object detection task two are the main objectives, identify correctly the target and localize it, to evaluate the latter one the IoU index is used. IoU, Intersection over Union, is a metric for evaluating the overlap between two boxes, given the true box containing the object and the one estimated by the classifier, the scope is to know how much these two boxes are similar, that is the ratio between the intersection and the union, the bigger the ratio is the higher is the similarity.
So, for object detection precision and recall are computed by using IoU values for a specific threshold, i.e., considering a threshold of 0.7 if the

IoU is greater than 0.7 the sample is considered as Positive, if lower as Negative.

To compute the mean average precision a new python script has been written, the aim of it is to run the convolutional neural networks on the test images and compare the results with the true boxes that were previously defined. Once the predictions have been made the algorithm computes the intersection and the union between the two bounding boxes to get the IoU ratio, then if the IoU is greater than a given threshold the sample is addressed as *True Positive*, instead if IoU is lower than a threshold it is addressed as *False Positive*, finally if the number of true boxes is greater than the number of estimated boxes minus the *False Positive* then the sample is considered *False Negative*.

By taking 30 images, not belonging to the train set, for testing purposes the previous computation can be made to get the average precision and the average recall. The results are:

$$AveragePrecision = 0.9047 \tag{3.5}$$

$$AverageRecall = 0.9523 \tag{3.6}$$

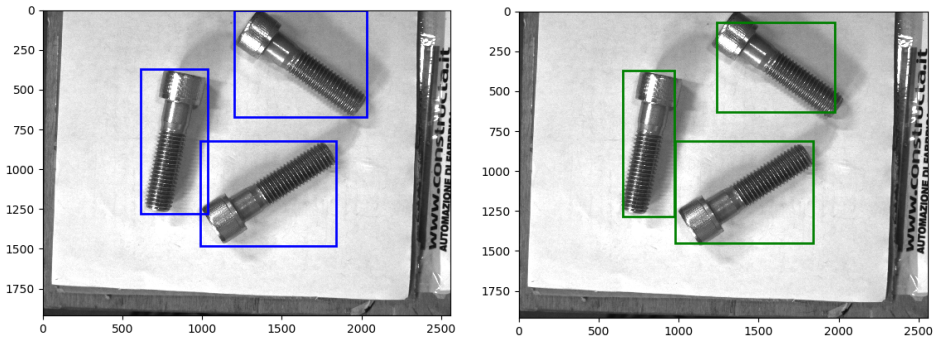


Figure 3.11: True boxes (left image) and estimated boxes (right image)

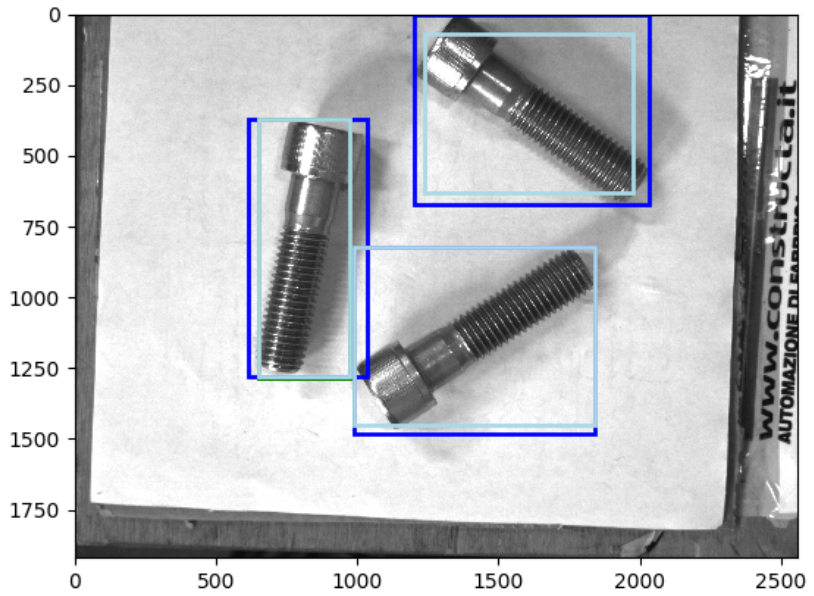


Figure 3.12: Intersection between estimated and true bounding boxes, in light blue the intersection box is highlighted.

3.6 Orientation estimation

Once the object is detected and the bounding box is found the robot needs the pose estimation of the target. To perform this operation different approaches are available, in the following different opportunities are presented.

By using the shape of the bounding box, it could be possible to get information about the orientation of the object, that is if one side of the rectangular is much greater than the other it means that side is closer to the main orientation, if the two side are similar it should mean the object is in a diagonal position with respect the camera. By using this knowledge some information can be gathered, the orientation could be estimated by computing the arco tangent of the ratio between the two sides, while the center with the middle point of the bounding box.

The second approach is still based on YOLO, but it's more elaborated, that is rotating the image 360 times (one for each degree) and for each one finds the bounding box, the degree corresponding to the minimal area of the bounding box should be the orientation of the object; but this approach would take much time and still it's not sure that the correct result would be found.

Another one is based on creating a new convolutional neural network that predicts the orientation, this could be a very clever and efficient way to proceed but in the literature the results of this approach are not satisfactory at all, so it will not be taken in consideration.

Finally, the approach used in this specific task is a traditional computer vision algorithm, which tries to find the main orientation of objects based on their specific geometric shapes.

This algorithm is thought for estimating the orientation of a specific class

of objects, that is objects with one size much greater than the other and with linear profile, for pieces that do not correspond to this description the algorithm should be refined.

To develop the algorithm screws are used as sample, these ones are long objects in which the thread is the part that is more highlighted from an edge detection. To get the edges of the image Canny has been used, it is an operator that uses a multi-stage algorithm to detect a wide range of edges in images; in figure 3.13 the image before and after the edge detection can be seen.
After that the algorithm starts looking for lines in the image enough long to be part of the thread, to do it the Hough operator is used, it works as follow. The set of all straight lines in the picture constitutes a two-parameter family. If a parametrization is fixed for the family, then an arbitrary straight line can be represented by a single point in the parameter space. By using the form shown in equation 3.7 it is possible to define a line by means of θ and ρ.

$$x \cdot cos\theta + y \cdot sin\theta = \rho \tag{3.7}$$

If we restrict θ to the interval [0, π], then the normal parameters for a line are unique. With this restriction, every line in the $x - y$ plane corresponds to a unique point in the θ-ρ plane.
Supposing to have a set $\{(x_1, y_1), ..., (x_n, y_n)\}$ of figure points, to find the straight lines they must be transformed into the sinusoidal curves:

$$\rho = x_i \cdot cos\theta + y_i \cdot sin\theta \tag{3.8}$$

At that point it is possible to show that the curves corresponding to

colinear figure points have a common point of intersection. This point in the $\theta - \rho$ plane defines the line passing through the colinear points. Thus, the problem of detecting colinear points can be converted to the problem of finding concurrent curves [6].
By using these approach and defining as constraint the minimum length of these lines it is possible to find several correspondent lines inside the bounding box containing the screw. Then filtering these result is important, the idea is to use a statistical approach to remove the outliers and take the final estimated orientation by computing the median of the remaining lines.
The other information needed to the robot, beyond the orientation, is the picking point; two are the ways to get it. The first approach is to use the information of YOLO, that is using the center of the bounding box. The second approach is to use the center of the lines that represent the thread. Both these ones work properly and has been tested successfully, in the simulation the latter approach will be preferred. Knowing the orientation and the position of the center of the object it is possible to move the robot to that position for the picking.

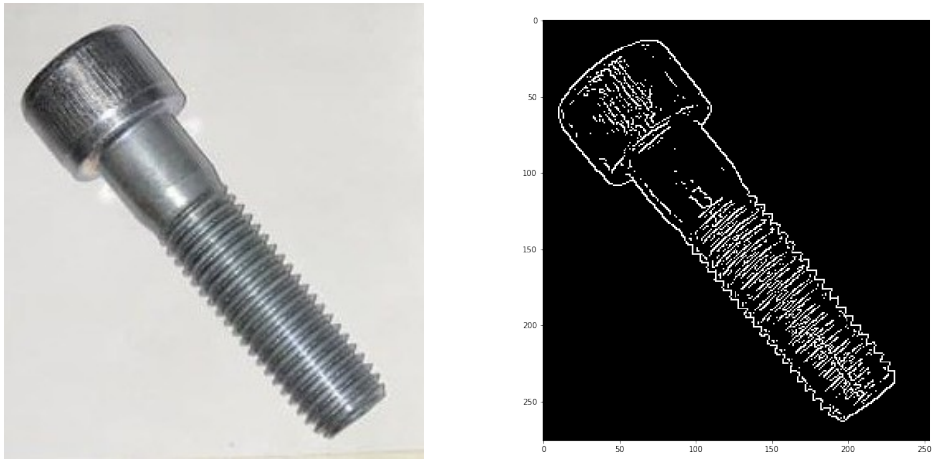


Figure 3.13: Image inside bounding box and its edge detection

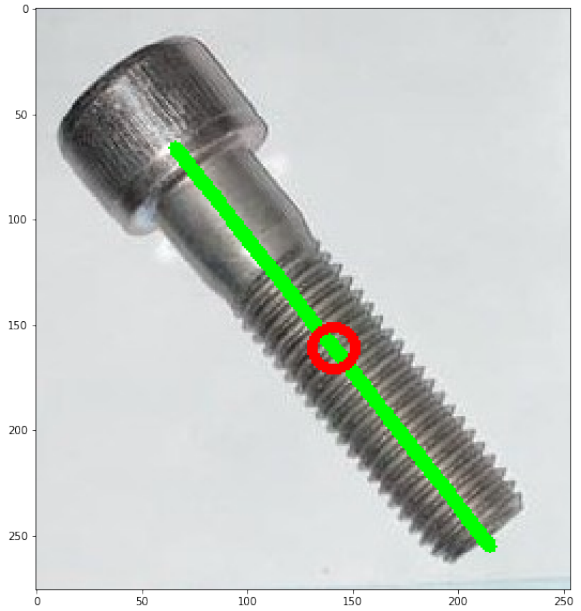


Figure 3.14: Result of the orientation algorithm on one screw, the green line represents the main axis and the red point the point in which the robot will pick the object

3.7 Second Stage Orientation

As said in section 2.4 in some scenarios it may be convenient to use an intermediate station to refine the grasping. That should be a position in which the object is released after it has been picked up. If this position is not used a poor pose estimation might arise, that is because in the main scene many objects are present and disturbances and noises may exist in the image. Instead, if this station is used the object is released on a clean surface without any occlusion or distortion, the consequence is that the conditions in which the object is placed are always the same, so a high

repeatability can be obtained by an algorithm.
The main problem in using the information coming from the main scene is that the middle point is not always in the real center of screw, and it is more difficult to understand if the head of the screw is below or above the picking point, and this information is very important if the object must be released in a given configuration.
Another limitation of not using this step is that every time objects are picked with the same inclination with respect to the surface, that is the gripper is always vertical with respect to the ground. If the final placement must be precise the object should always be picked up with the same inclination with respect to the ground, by picking and releasing them over a planar surface this issue is always solved.
By considering the above reasons it seems reasonable to use this procedure almost always, even if the time required will inevitably increase. Once the object is released the steps are the following ones.
The robot will run YOLO again on the new scene image to predict the bounding box. In this case the frame is clean from disturbances or other objects so the prediction of the neural network will be very reliable, so that we can use the center point of the bounding box as a new picking point. By doing that it is expected that each time YOLO will return the same result, because objects are the same size and all the environmental conditions are the same.
To understand the position of the head of the screw, instead, the bounding box is divided in two halves, above and below. The idea is like the one used for searching the lines, the half with more lines will be identified like the ”tail” while the other one like the ”head”. At this point the robot can grasp the object again in the new point.

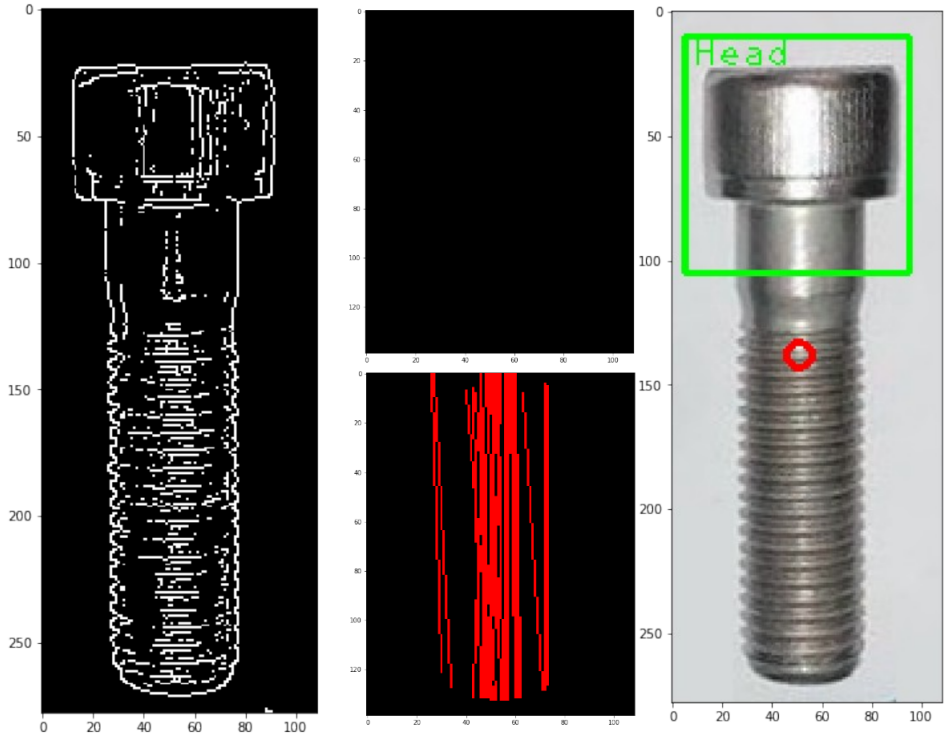


Figure 3.15: Result of the second stage orientation recognition

Chapter 4

Generalization of the algorithm

Until this point the task was tested only with a particular object and with objects that lie on the same plane, because without a 3D camera it was not possible to get information about the distance of the objects from the camera.
In this chapter both these two constrains will be faced trying to relax them to generalize the algorithm. So, in the following a technique to get 3D information from a 2D camera will be tested and two new neural networks will be trained.

4.1 3D Vision

3D vision is the branch of computer vision whose aim is to build a 3D model of the scene gathering information along x,y and z axes. There are many ways to get 3D images, in the following lines some of them will be explored:

1. Time-of-flight (TOF)

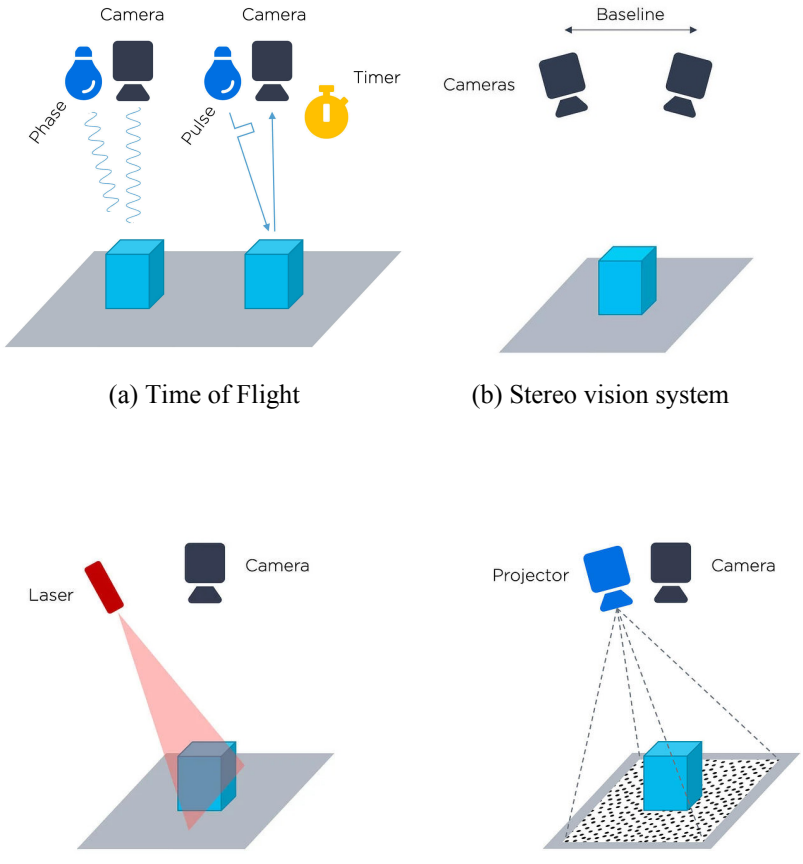
2. Stereo vision
3. Laser triangulation
4. Structured light

Mimicking human vision, the stereo vision technique uses a pair of cameras to record the same 2D view of the target object taken from two different angles. Knowing the fixed relative positioning of the two cameras, software compares corresponding points in the two flat images, identifies disparities and through triangulation produces a full 3D point cloud [1] (4.1b).

A line laser is a relatively cheap and simple active light source that can be used together with a camera to avoid the stereo vision setup with two cameras and an RPP. This old technique is simple, low cost, fast and highly accurate. Laser triangulation is one of the most popular and commonly used 3D imaging techniques and is deployed in a wide range of interior and exterior applications [1] (4.1c).

The structured light technique is like that of laser triangulation but is extended to a whole field. It is called a full-field method because it provides an entire 3D image of the object, and not just of a single cross-sectional line. It records the geometric distortion of a known illumination pattern projected onto the surface of a static three-dimensional target object [1] (4.1d).

Taking a time-domain rather than a spatial-domain approach to 3D imaging, time-of-flight laser scanners, sometimes also called LIDAR systems or laser radars, effectively remove the baseline. By measuring the time delay between emitted laser light and the reflected laser light from the object's surface, we can get a precise distance measurement [1] (4.1a).



(a) Time of Flight (b) Stereo vision system

(c) Laser trinagulation (d) Structured light

Figure 4.1: Different 3D acquisition systems

4.1.1 Stereo-Vision

Stereo vision is the extraction of 3D information from images gathered with 2D camera, it is one of the major vision modules by which one can induce the depth of the surface shape and the volume information of the

objects. Stereo vision systems usually are built with a system of two 2D cameras rigidly connected, the frames grabbed by these two cameras are named conjugate images and it is possible to get information about the depth by exploring the concept of disparity.

Stereo vision deals with three major problems: correspondence geometry, camera geometry and scene geometry. Of these, stereo matching deals with correspondence geometry and remains the major research area [9]. The two images gathered by the cameras in this system must be rectified, that consists in projecting the images onto a common image plane in such a way that the corresponding points have the same row coordinates. In that way the image planes are coplanar and the epipolar lines are collinear and thus the search for matching points becomes a one-dimensional search problem.

$$disparity = x - x' = B \cdot \frac{f}{Z} \tag{4.1}$$

x and x' are the distances between the corresponding points in image plane. B is the distance between two cameras, that is known and fixed, and f is the focal length of camera, known by the camera calibration problem. So, in short, the above equation says that the depth of a point in a scene is inversely proportional to the difference in distance of corresponding image points and their camera centers. So, with this information, we can derive the depth of all pixels in an image.

Since in this specific task only a 2D camera was available the classical stereo vision system could not be faced in the traditional way, but it was necessary to use the same camera in different position for detecting

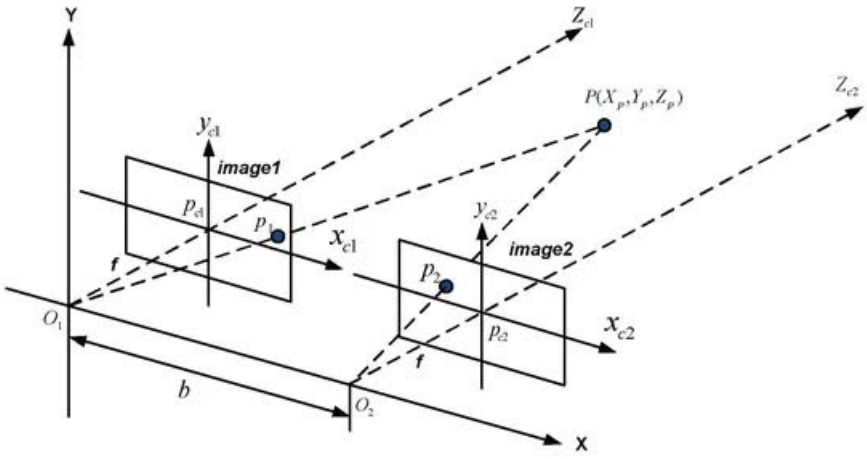


Figure 4.2: Stereo vision architecture

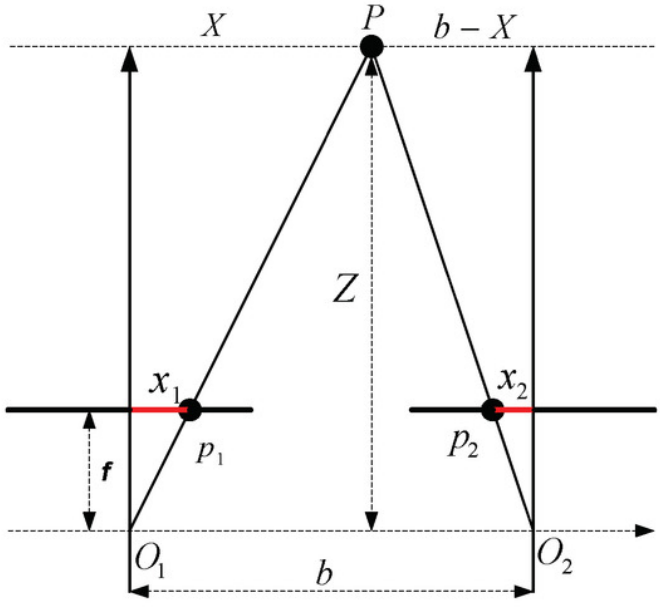
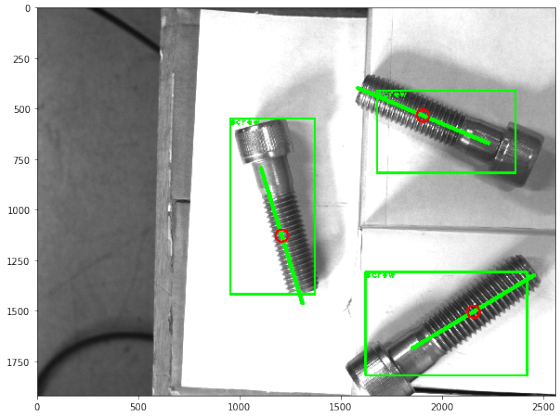
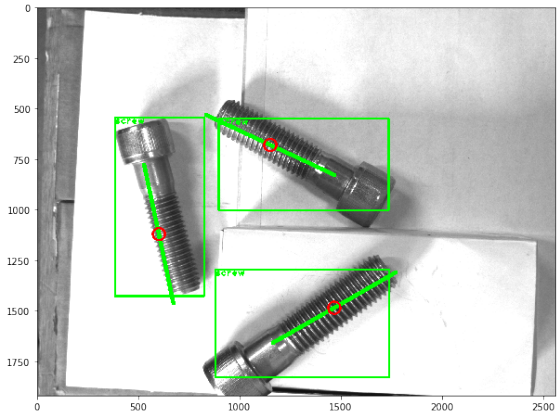


Figure 4.3: Stereo vision rectified structure

the correspondences and estimating the distances. So, with respect to the previous case the camera will take a shot of the scene from a predefined position, then it will move to a second position moving only along the y axis of the robot for 6 cm and it will take a second shot.
Once both images are available it is possible to compute the disparity and then estimate the distance at which the target point is. In figure 4.4 two images can be observed, figure 4.4a represents the left one and figure 4.4b the right one; in both the images the same three bounding boxes are detected, for each one the picking point is found and then used as reference point to compute the disparity.
By using the already available information, as the focal length and the distance between the two shot positions, with the disparity it is possible to know the distance of each point from the camera. In this particular case these distances are reported in figure 4.5, the screw on the bottom is the closest one with $\simeq 49.4cm$, then the one on the top right with $\simeq 56.9cm$ and the furthest on the left with $\simeq 60.1cm$.
By measuring the real distances and comparing them with the ones estimated the error is about half centimeter; this result is not bad, but at the same time in the industry this error would not be tolerated. So, even if the algorithm produces very good results, these ones are not enough for be applied to applications in which a high accuracy is required. To conclude, it can be said that a 3D camera would be more appropriate for this type of application.



(a)



(b)

Figure 4.4: Left and right image acquire by the camera

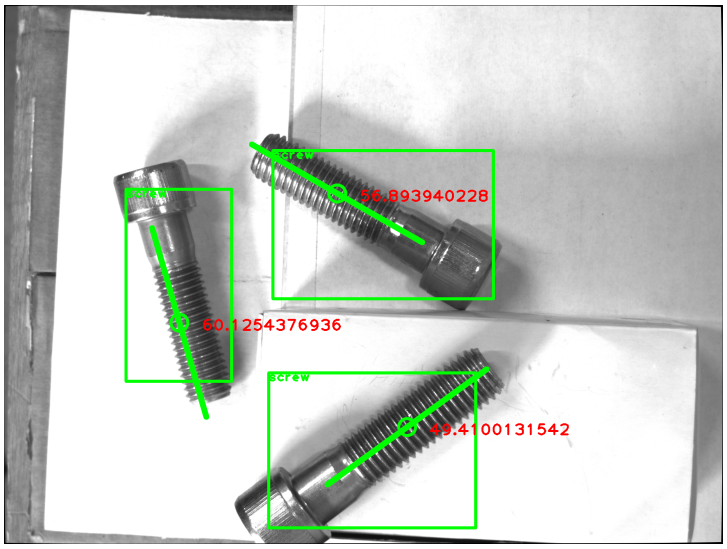


Figure 4.5: Scene with the estimated values of depth, expected values : 60, 56.3, 50 cm

4.2 Multi objects recognition

In chapter 3 an algorithm to detect objects and to recognize the orientation has been developed and tested for a singular object, a screw. In this section the algorithm will be tested with new sets of objects, and it will be proved it could be a general purpose approach to solve bin picking problems.

4.2.1 Training 2 new CNNs

To detect new objects 2 new classifiers must be trained, as samples a lighter and a marking pen are chosen. To the train the two neural networks 2 sets of 40 images are used, as before all images have been labelled before

Figure 4.6: Sample images from the lighter train set

Figure 4.7: Sample images from the pencil train set

carring out the training; once this has been done YOLO must be re-trained with the new sets. Since the images are quite clean from disturbances the results are very good, that is an average loss around 0.08 and average mAP equal to 0.96.

But it was already known that YOLO could work properly with these two objects, the real challenge was to prove that also the orientation algorithm could work, so that it could be proved that it is general purpose for a class of objects.
To do that the same algorithm used for the screw has been tested on the

test set of the lighter and the marking pen, the results were excellent, without changing any parameters the code worked perfectly. In figure 4.8 two samples from the test sets are plotted, the bounding box, the orientation and the picking point can be observed and they match the real ones.
In this particular case nothing was changed before running the algorithm, but in general the only parameters that should be changed is the minimum/maximum length of the lines detected by the Hough transform, in particular this measurement could be obtained by using:

$$MinimumLenght = \frac{0.7 \cdot ObjectLenght}{PixelSize \cdot Distance} \tag{4.2}$$

In equation 4.2 the only parameters the user should set are the real size of the object and the distance of it from the camera. Once these two parameters are known the algorithm will consider as minimum length for the lines the 70 percent of the expected length in pixels of the object.

The fact of the algorithm worked well with these two object do not mean that it should work with any object, indeed these two were not chosen randomly. The lighter and the marking pen, like the screw, have two characteristic that are very important:

- One side is much greater than the other one.
- The profile of the objects is linear, that is it can be approximated by straight lines.

To conclude, every object that has these characteristics theoretically could be used in the application without any relevant change.
Instead, the procedure explained in section 3.7 to detect the upper part of the object can not be generalized, that is because it relies on the detection

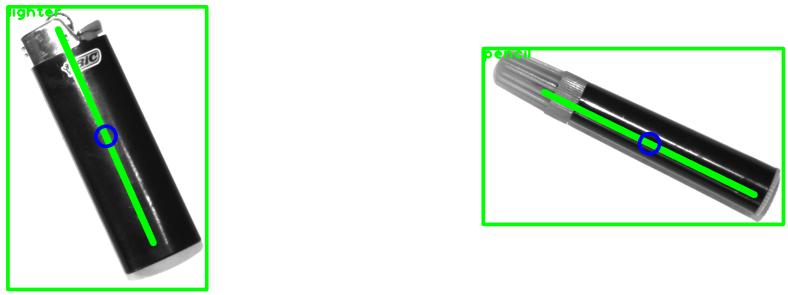


Figure 4.8: Result of the detection and orientation algorithm on the two new classes of objects

of a particular detail in the object. So, in this case for each new object a new approach should be developed.

Chapter 5

Oriented bounding boxes for accuracy index

To evaluate the results of the orientation accuracy, the real orientation of each object in the test set was needed. Understand how much the estimated orientation was far from the real one was of interest.
Doing this operation by hand would have required a lot of time and effort, while the best idea was to create a tool that could do this operation in a fast and reliable way. In this chapter the idea behind this procedure is explained.

5.1 Labelling Desktop Application

Many applications exist for drawing oriented boxes on images for classification purposes, these are tools that allow the user to create rectangle on the image to classify them in each category. For example, in section 3.5 the program *labelImg* has been used to classify the training set before

training the convolutional neural network; this kind of apps are very user friendly but all of them have a big limitation, they do not deal with orientation.
This fact can be easily explained because the kind of problems treated in this thesis has not been addressed by the scientific community as relevant yet, that is no one has solved it in a clear way. But many researchers worldwide have tried to solve it, the application that is explained in the following lines wants to be a help to whoever wants to face this kind of problem.

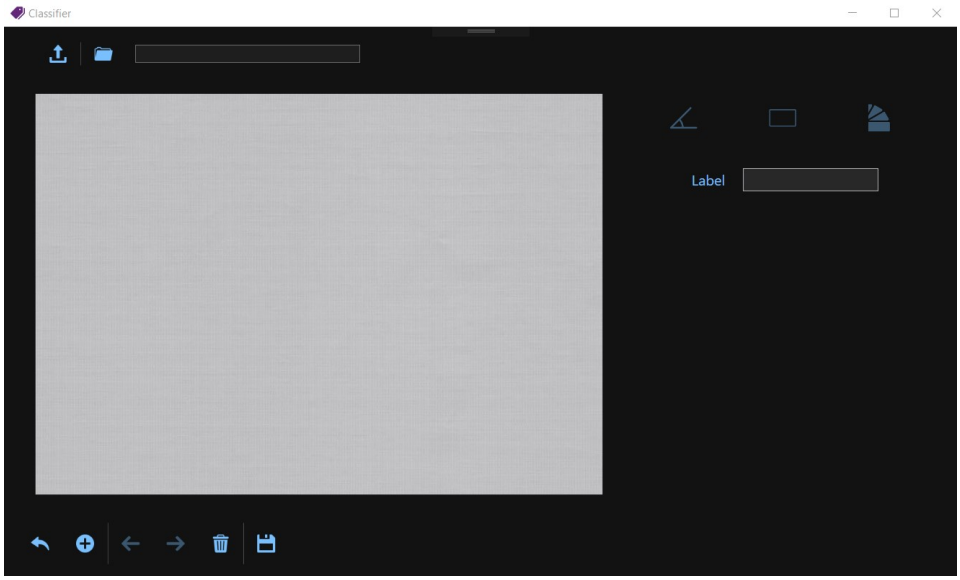


Figure 5.1: Initial page of the app for the labelling

The application (fig. 5.1) has three main tools:

- Drawing the classical bounding box as a rectangle.

- Drawing an oriented bounding box as an oriented rectangle.
- Drawing a line representing the main orientation of the object.

The app does not deal only with the orientation problem, because often this is not needed, for this reason the classical approach with the standard rectangle is still present.
Then the other two tools are similar, but they might be needed for different kinds of applications. If predicting the position and orientation of the whole object in space is needed, then the oriented rectangle would be the best choice. Instead, if only the orientation is of interest, it might be enough using a line to classify the angle of the object with respect to the x axis of the image.
These three tools are used inside an app that is very user friendly and has other utility to make the job of classification easier.

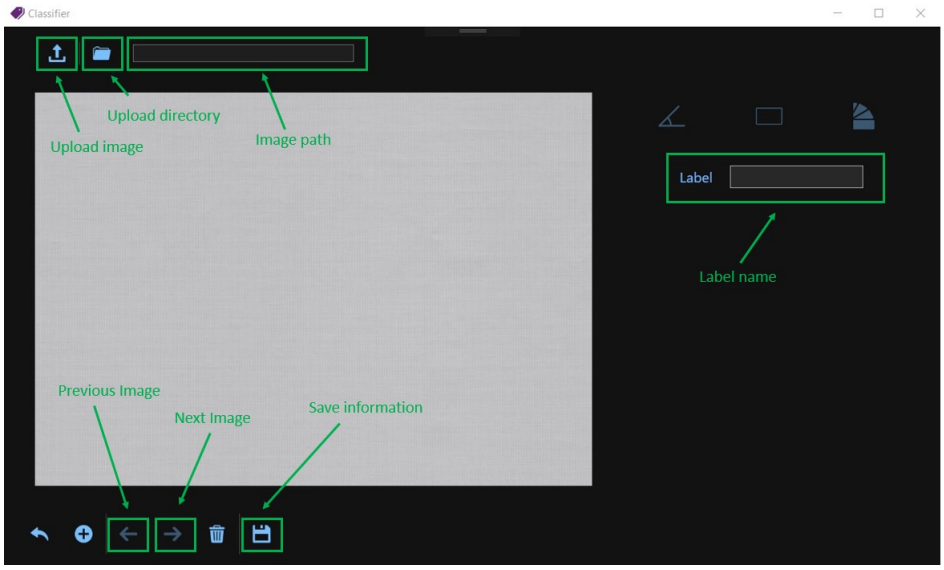


Figure 5.2: Instructions for the app

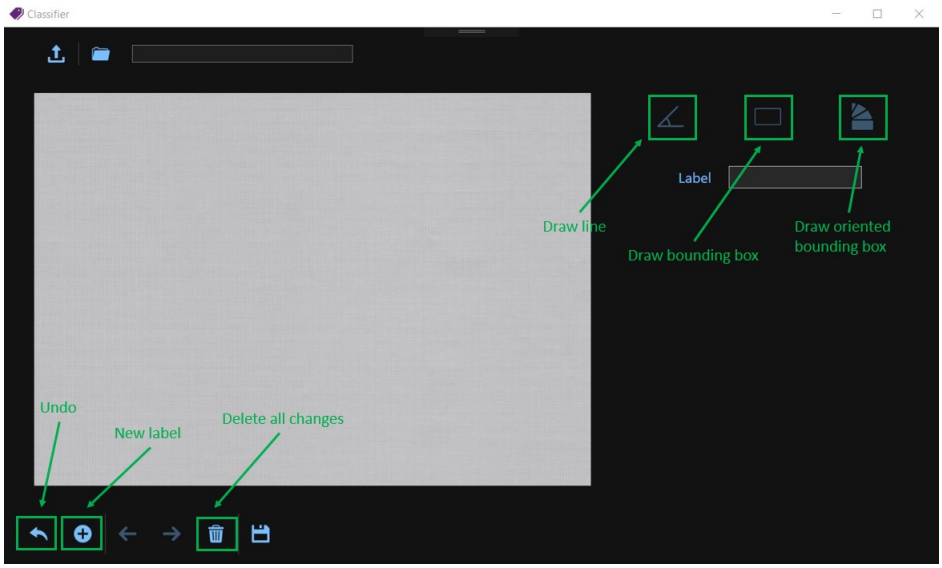


Figure 5.3: Instructions for the app

In figures 5.2 and 5.3 a brief explanation of the app tool is provided, the main features are:

- Single images or whole folder of images can be loaded;
- It works with multiple objects inside the same image and also with different labels, that can be changed in the *Text Box* on the right;
- Different type of information can be saved: rectangles, oriented rectangles and lines;
- A file with the same name of the image is saved with the information related to the labels.

In figure 5.4 the classical bounding box is used, for each typology of tool a user control is added to the image with the information related to

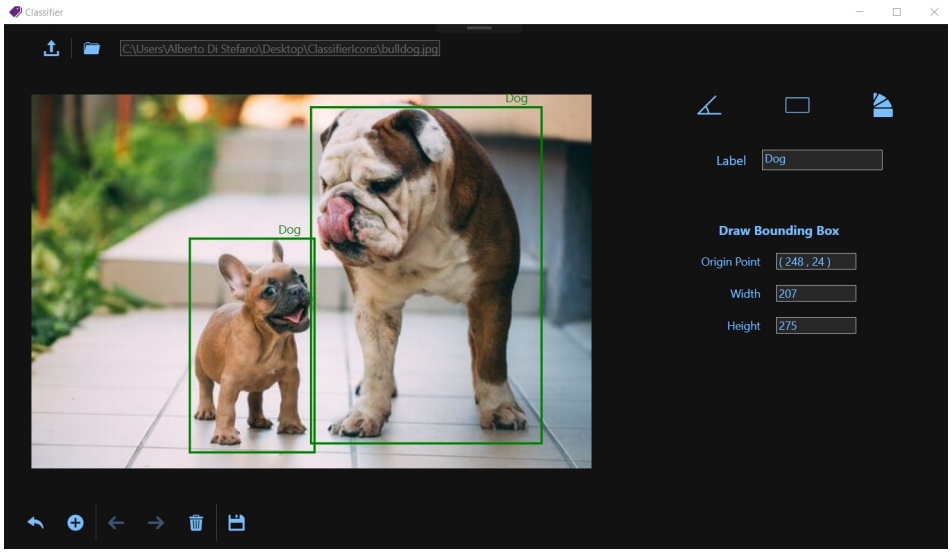


Figure 5.4: Using the classical bounding box for the classification

the position of the boxes in the image. In particular for the rectangle the origin point, the width and the height are provided, the same information will be saved in the related *.txt* file.

In figure 5.5 oriented bounding boxes, this is one of the tool that was not present in the previous applications. To draw this kind of form the user must select four points in the image and then the app will draw the rectangle related to the user choice. In the image it can be observed that the app works well also with different labels in the same image.

In figure 5.6 the last tool is used, that is the line. In the image two screws are placed with different orientations and the user has drawn the lines that fit the main orientation of the two objects. In particular in this case in the user control on the right the two extreme points are shown together with the orientation of the object, the orientation goes from -90 to +90, where the 0 represent an object with the main axis parallel to the

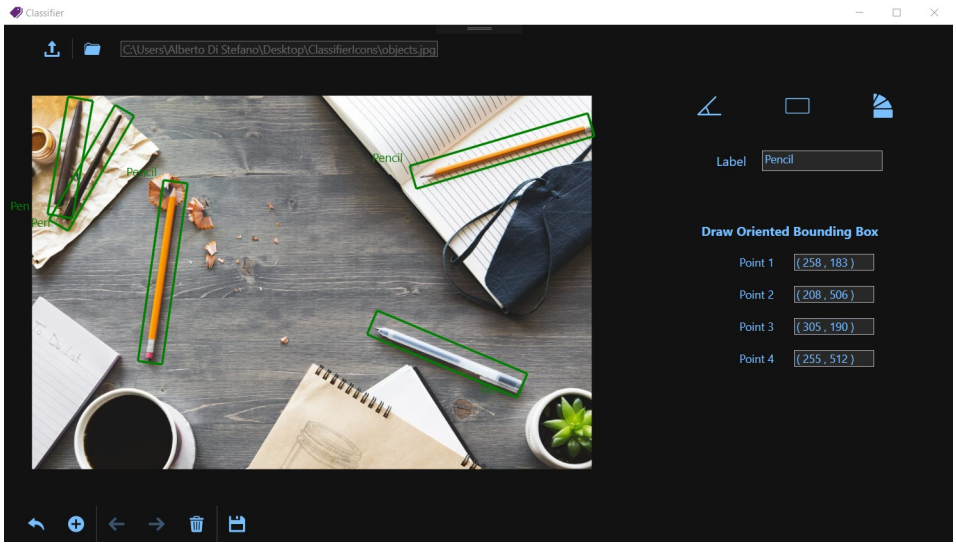


Figure 5.5: Using the oriented bounding boxes for classifying different objects on a table

horizontal axis.

5.2 Evaluation of the orientation task precision

In section 3.6 an algorithm to find out the orientation of the object detected by YOLO has been formulated, as already pointed out the algorithm works quite well, but until now this evaluation was based only on a purely subjective point of view. This means that by looking at the graphical results on the test set the orientation of the line plotted seems to match the real one.

To be more precise and be able to provide reliable indexes about the performances a routine has been implemented.

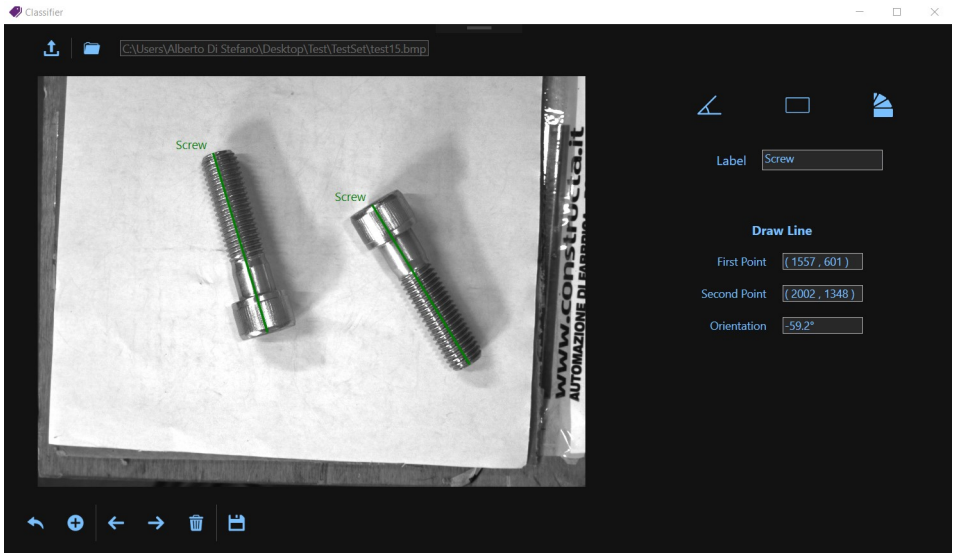


Figure 5.6: Using the oriented line for classify/get the orientation of each object

The idea is to repeat the same operation made to evaluate the performance of YOLO. The first step is finding the real orientation of each object in the test image set and save all the information about each image in a file, this operation is done entirely by using the app explained in section 5.1. The second step is running the whole algorithm for the pose estimation and save on text file the information about the estimated orientation of each object, it is important that the two orientations are calculated in the same way with the same reference system.

Once both these actions are made the difference between the real and the estimated orientation has been computed; then the average error has been obtained with the formula:

$$AverageError = \frac{\sum_i |RealAngle_i - EstimatedAngle_i|}{NumberOfItems} \quad (5.1)$$

The formula 5.1 is used to compute the error in the estimation of the angle, in particular a python script has been written. This script opens the two text files related to each image and compares the orientation of the corresponding bounding boxes appending the difference to a list. Once each image has been inspected the script uses the above formula to compute the average error, by trying it on the test set the error is of 2.83 degrees (or 0.05 radiant).

This result is positive, that is because the error is very low, moreover the real measures have been taken by hand so they are not very accurate and it is very unlikely that some algorithm could reach 0 error, that is not because the algorithm does not work, but because the measures taken by hands are not reliable.

Chapter 6

Experimental analysis

6.1 Experiments

In this final chapter the test and the evaluation of the task will be made. As mentioned in section 2.4 two are the main experiments of interest: the bin picking application with two shot positions and an intermediate stop for refining the grasping and the case with only one shot position.
The latter one is now taken in account, in this case the scene must be composed by pieces that are almost on the same plane, they can overlap but not too much, for example in figure 6.1 it can be observed a casual initial configuration. Two problems will arise by having objects on different planes, the first one is that the robot has a pre-defined height, and so it will try to arrive to that position, the second one is that the camera calibration has been made for a specific plane, then by changing the plane the previous calibration does not hold anymore and so the distances computed by the algorithm along x and y will be imprecise.
From the robot point of view instead the procedure will be the following

one:

1. The robot is brought from the *home* position to the *shot* one;
2. The camera frames the scene and the algorithm (running still on ROS) provides the picking point and the orientation of the gripper;
3. The robot goes on the target point and starts going down until the pre-defined height and closes the gripper;
4. The robot goes up and move to the intermediate station for releasing the piece;
5. Once the robot is over the object the camera grabs another photo computing the middle point and estimating the position of the screw's head with respect to the center;
6. At this point the gripper will go to the new picking point, and only if the head is below the center the robot will rotate of 180 degree;
7. Finally the object will be released in a pattern in vertical position with the head on the ground.

This routine is repeated until all the pieces have been moved from the initial configuration to the final one, an example of the final pattern is provided in figure 6.2.

Figure 6.1: Initial configuration: objects placed in random positions

Figure 6.2: Final configuration: objects in desired positions

In figure 6.3 can be seen on the left the robot over the scene while the camera is elaborating the image, while on the top right corner it is represented the simulation on *RViz* running on the computer and in the bottom right corner there is the image elaborated by the camera. In the scene 8 screws are present one over the other one, different elements can be seen in the picture:

- The green rectangle that is the bounding box provided by YOLO;
- The green line that is the main axis of inertia of the object, so it will be the rotation that the robot wrist will perform;
- the green point that is the picking point of the object.

In the figure 6.4 it can be observed the gripper over the picking point with the right orientation. After this operation the robots will move in the configuration represented in figure 6.5 where on the bottom right corner the new scene is depicted: the white bounding box is the result of Yolo and the picking point is the center of the rectangle. Finally, the object is released in figure 6.5 in a pre-defined pattern.

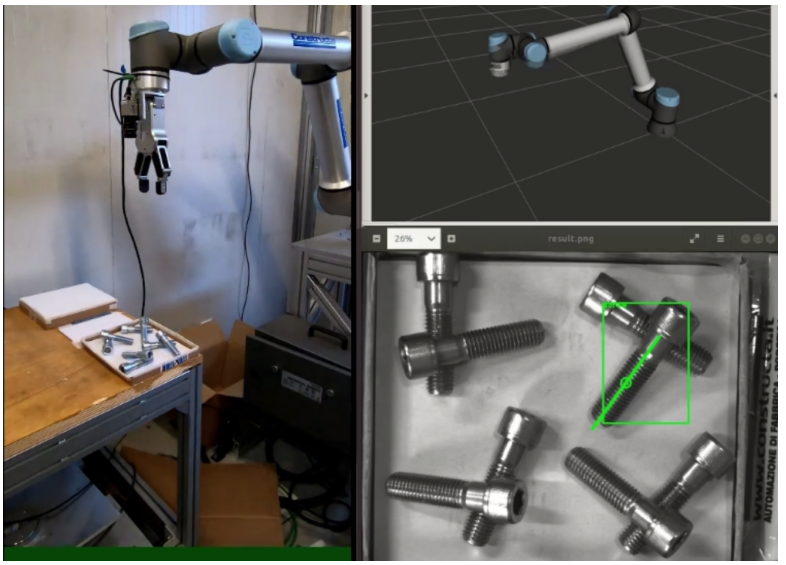


Figure 6.3: The robot in the *shot* position runs the computer vision algorithm.

Figure 6.4: The robot picks up the object in the right position and orientation.

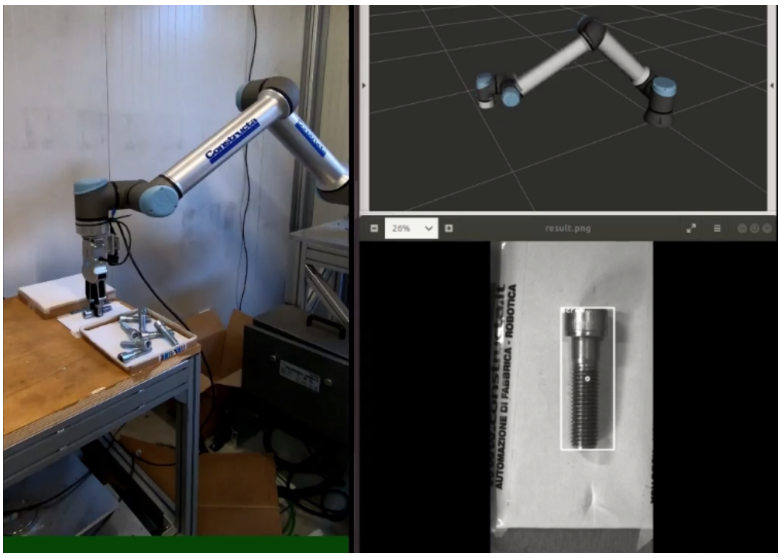


Figure 6.5: In the second stage the orientation is adjusted and the head is identified.

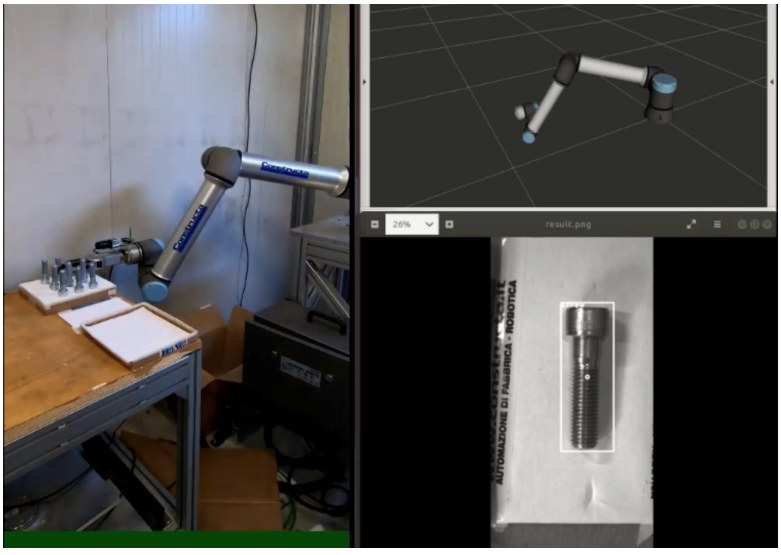


Figure 6.6: The object is released in the final configuration and the procedure starts from the begin.

The second experiment of interest is the one related to the computation of the depth, as said in section 4.1.1 to do it two images of the same scene must be available. The general procedure is very similar to the one presented in page 75, the only difference is that an additional step must be added, that is after point 2 the robot will be moved in a second position along the y axis to grab another frame and only at that point it will compute the position of the center point, this time both along x,y,z axis. In figure 6.7 the scene in this second scenario can be observed, three screws are placed at different heights, by taking two shots the robot is able to detect the distance between the camera and the objects and then it will pick up the object that is closer to the camera.

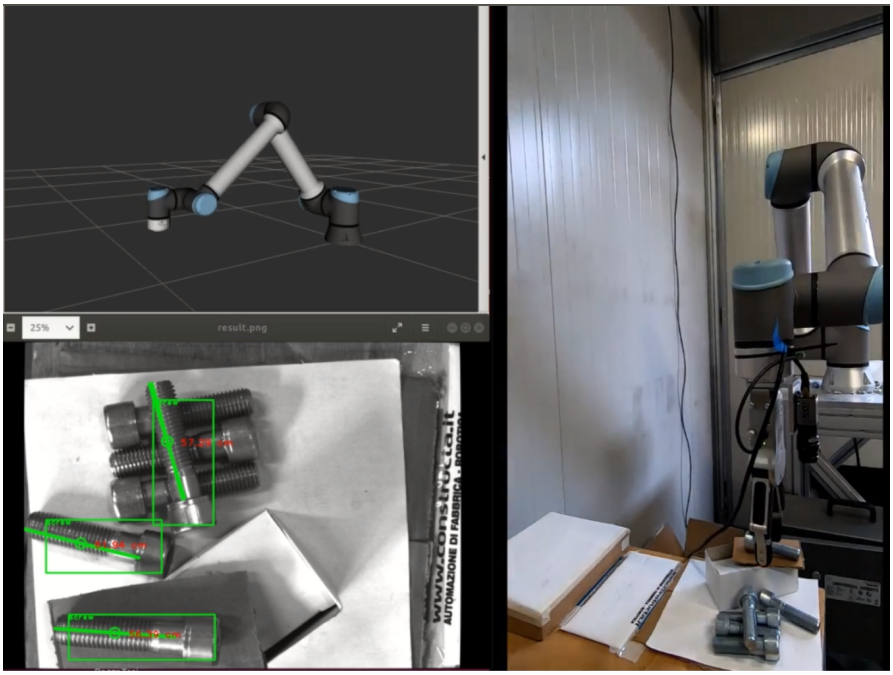


Figure 6.7: The robot is picking up the closer object to the camera.

6.2 Results

By running the previous experiments some considerations can be made. Both the task of interest work pretty well, the robot is able to perform the desired operation in a correct way: detection, picking and releasing.
The application at the moment is quite slow, it takes a bit less than 1 minute for each object; but this is due to the fact that the hardware is not optimized for working with deep learning algorithm and also because we are performing the motion with a collaborative robot in reduced mode, in that way it doesn't makes damages to itself or to the environment.

Chapter 7

Conclusions

To conclude some considerations are reported in the following lines.
Considering the first scenarios, that is the one with only one shot position, it could be said that overall the algorithm proposed works quite well, in particular if the scene conditions respect the initial assumptions everything works fine, that is pieces must be almost on the same surface and there must be enough space for the gripper to pick up the objects without colliding.
To analyze better the results each part of the application is considered separately: object detection, pose estimation and robot behaviour.

The neural networks that have been developed works very well and with a good accuracy, in particular also in harsh environments. The estimated latency for YOLO is of 22 ms but due to the poor hardware available (no GPU) the time required for the detection is over 1 second; with a suitable hardware the performances of the vision algorithm would increase a lot.

The orientation part instead is a totally new approach, differently from YOLO, and so more considerations should be made. This technique has the aim of estimate the orientation of pieces that has well defined characteristic, that is objects with one side greater than the other one and with linear edges; in this case the algorithm has been tested with a screw, a lighter and a pencil mark.
In the literature the algorithms similar to the one that has been implemented in this application has been tested with only typology of object, like for detecting car direction [4] or boats orientation [20]. Both these have shapes that could work also with the algorithm presented so far, due to their profile, so it can be said that the majority of times the orientation is needed this algorithm should work.
By comparing this approach with YOLO it can be said that it is less general purpose, and that is true because YOLO works with every type of object, while these algorithm only with a restricted class. At the same time it should be reported that implementing YOLO requires much more time than the new algorithm, that is ready to be used for every object belonging to the just said category.
Moreover the most of the times orientation is needed the objects have a side much greater than the other and linear profile, so this approach could cover the majority of scenarios.
Talking about performances, instead, the time required for the whole computation is $\simeq 0.15$ seconds with a poor hardware, so considering both the parts of the computer vision task, detection and pose estimation, during the test experiments $\simeq 1.2$ seconds was required to do the whole procedure, but it is reasonable that with a suitable hardware the time should decrease to a couple of tenths of a second.

The second stage station has revealed itself as a very important part of the process, it allows the detection of the objects and of their middle point with a very good accuracy and repeatability. Since the objects are released always in the same point with the same orientation, the condition are very similar, then YOLO always returns the same bounding box; differently than before this estimate is very reliable and for this reason it can be used for estimating the picking point.
The second part of this second stage is the identification of the head of the screw, that still works very well thank to the conditions of the environment.
This, anyways, is the only part of the task that cannot be general purpose, because any object has its own detail to distinguish the two ends, so a specific approach must be developed, but it should be an easy task looking for a detail in one of the two half of the object.

The last part to be evaluated is the robot, which is the only actuator of the application so it has the responsibility that everything works fine, in particular the most complicated aspect from the robot point of view is the picking of the objects, because in the other phases the robot moves almost to a fixed position. In the picking operation two are the main problems : the first one is that the gripper could encounter obstacles during the descending phase and the second is defining in the correct way the position to which it must move. As it can be observed in the chapter 3 the camera is able to understand the position returning the correct pixel distance, but then these distances must be transformed in the positions in meters to feed the robot; this operation may seem easy but it hides some problems. Even if the hand eye calibration has been done correctly, it has been done with

respect to a precise plane, if the plane is changed this does not hold anymore; so until the objects are displaced on the same surface the problem is solved, while if this is not the case a new transformation must be used.

In conclusion, the desired result has been reached and it is very interesting. But some limitations are present, if the objects are not on the same plane this approach is not convenient because a 3D camera would have much greater performances.
In relation to this last aspect the second scenario must be taken in account, that is the one in which two shots of the same scene are taken to get information about the depth. This second algorithm works the same as the previous one, but it take a second photo of the objects; this fact increases for sure the time required for the whole procedure, a double time must be considered for the acquisition and elaboration of the image, but at the same time it provides a more complete information.
As seen in section 4.1.1 the approach works well, but for sure by using a real stereo vision system the accuracy would increase and the computation time would decrease.

Bibliography

[4] Iu. Chyrka1 and V. Kharchenko. “1D direction estimation with a YOLO network”. In: *Research department, National aviation university* () (cit. on p. 82).

[5] Cyganek and Sieber. *An Introduction to 3D Computer Vision Techniques and Algorithms*. 2009.

[6] R. Duda and P. Hart. “Use of the Hough transformation to detect lines and curves in pictures”. In: *Commun. ACM* 15 (1972), pp. 11–15 (cit. on p. 50).

[7] Radu Horaud and Fadi Dornaika. “Hand-eye Calibration”. In: *The International Journal of Robotics Research* (1995).

[9] Hong Jeong. *Architectures for Computer Vision: From Algorithm to Chip with Verilog*. Wiley, 1981 (cit. on p. 58).

[12] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *Computer Science Department University of British Columbia* (2004) (cit. on p. 34).

[15] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: (2016) (cit. on pp. 37, 41).

[18] Christian Szegedy et al. “Going Deeper with Convolutions”. In: (2014) (cit. on p. 38).

[20] Jizhou Wang. “Simultaneous Ship Detection and Orientation Estimation in SAR Images Based on Attention Module and Angle Regression”. In: () (cit. on p. 82).

[21] Ye et al. “A new method based on hough transform for quick line and circle detection”. In: *2015 8th International Conference on Biomedical Engineering and Informatics (BMEI)* (2015).

Sitography

[1] *3D vision systems*. URL: https://www.zivid.com/3d-vision-technology-principles (cit. on p. 56).

[2] *About ROS*. URL: https://www.ros.org/about-ros/%5C#:~:text=The%5C%20Robot%5C%20Operating%5C%20System%5C%20(ROS,wide%5C%20variety%5C%20of%5C%20robotic%5C%20platforms (cit. on p. 14).

[3] *Automation and Robotics company*. URL: http://www.constructa.it (cit. on p. 10).

[8] *IDS site*. URL: https://en.ids-imaging.com/store/ui-5480cp-rev-2.html (cit. on p. 27).

[10] *Kowa site*. URL: https://www.kowa-lenses.com/en/lm12jc-mp-industrial-lens-c-mount (cit. on p. 27).

[11] *Label Img*. URL: https://github.com/tzutalin/labelImg (cit. on p. 43).

[13] *MoveIt ROS*. URL: https://moveit.ros.org (cit. on p. 24).

[14] *OpenCV*. URL: https://docs.opencv.org/4.5.2/d9/d0c/group%5C_%5C_calib3d.html%5C#gad10a5ef12ee3499a0774c7904a801b99 (cit. on p. 18).

[16] *ROS Introduction*. URL: http://wiki.ros.org/ROS/Introduction (cit. on p. 14).

[17] *RTDE Interface Universal Robot*. URL: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/ (cit. on p. 15).

[19] *Universal Robot UR10e*. URL: https://www.universal-robots.com/products/ur10-robot/ (cit. on p. 11).