Politecnico di Torino

Master’s degree in Mechatronic engineering

Master’s Degree Thesis

Autonomous mobile robots: configuration of an automated inspection system

Supervisors

Prof. Giulia Fracastoro
Dr. Oscar Pistamiglio

Candidate

Francesco De Santis

Academic year 2022 - 2023

Summary

Robotic technologies are becoming popular and almost essential nowadays in many industrial sectors and for the most disparate purposes. They provide huge benefits to companies and workers: carry out monotonous, unpredictable, and hazardous jobs in harsh environments, resulting in an increased safety and availability of personnel that can be differently deployed. Plant inspection and maintenance are generally human-intensive. Tedious and repetitive tasks may entail decreasing attention spans of employees, leading to errors that may have serious consequences. In that context, robots accomplish thoroughness, safeness, cost and time efficiencies.

This thesis work is part of the project carried out by Sprint Reply from the group Reply S.P.A., a company hard experienced with Robotic Process Automation, OCR tools, Natural Language Processing as well as physical social robots. The project aims to develop a robotic-aided solution for a world leader company in the Oil & Gas industry, to detect gas or liquid leakages, monitor the conditions of working machines, check the availability of HSE equipment and create a digital representation of the sites using 3D LIDAR scans.

Spot from Boston Dynamics is the robot used to perform the required tasks. It is an agile mobile robot that allows to automate routine inspection tasks of any area of interest, to monitor different scenarios, detecting and recognizing specific objects and capturing data safely, accurately, and frequently. Using the Software Development Kit, it is possible to create custom controls, program missions, and integrate sensor inputs into data analysis tools.

In particular, the thesis activity focuses on visual data processing that consists in building image processing algorithms. During scheduled patrols in the facility plant, Spot recognizes target elements through a widely used neural network for object detection and takes photos that are processed using computer vision. The algorithms recognize peculiar characteristics in the acquired images and verify if they match with a set of given parameters. They collect information and produce as output numerical values or alert messages that are communicated to the operators, visible through a management console.

Acknowledgments

The work presented in this thesis is the result of a shared effort between my colleagues Paolo Mazza, Pierluigi Compagnone, Vinogiga Vanniyakulasingam and I, with each of us contributing equally to the study and achievement of the project results.

Table of Contents

List of figures

List of tables

Chapter 1

Introduction

The work thesis that follows details a six-months project, from July to December 2022, carried out by the team of Sprint Reply S.p.A. and developed for a prominent company in the oil and gas industry. Although the project spanned two years, during the closeout phase, the group focused on delivering the final design to the client, reaping the benefits of common effort and showing that concrete outcomes were achieved.

Sprint Reply S.p.A. is a company within the Reply Group, a global consulting, systems integration, and digital services company. It focuses on the development and implementation of innovative solutions, including robotics, IoT, artificial intelligence, and machine learning, and is involved in a wide range of projects in different industrial sectors. They work alongside their clients to create tailored solutions that address their specific business challenges and leverage the latest technologies to provide innovative and effective answers.

In this case the project aimed to create autonomous visual inspections inside the plant of the client company, and this using Spot from Boston Dynamics.

Spot is an agile quadrupedal robot that is able to navigate challenging environments and to complete different tasks. Spot can operate autonomously or be controlled remotely, allowing flexibility and versatility in different situations. Additionally, it can mount a variety of sensors to collect data and capture images, including the powerful Spot Cam+ camera, with an optical zoom, which can provide high-definition shots of hard-to-reach equipment and instruments. These features make Spot a valuable tool for optimizing plant inspection processes, reducing costs, and improving safety.

The project focused on developing three distinct autonomous robotic inspection tours, each in a different area of the plant, with the purpose of reading values reported by meters located within the facility. During the inspections, Spot freely navigates along a prerecorded path and stops at every chosen target be read. The Spot Cam+ is employed to capture images of the devices, which are subjected to digital processing.

The digital processing involves utilizing Yolo, a well-known object detection model, to identify the meter within the image and crop it around the precise area of interest for the numerical values extraction. Thereafter, a reading algorithm, specifically designed for each meter category, is applied.

From a temporal point of view, the work was organized in two phases. In the first period, images of the meters were collected. These images were used to construct the dataset for training and validating Yolo and to design the reading algorithms. During the second phase, numerous tests were conducted within the plant to assess the overall autonomous inspection system. Simultaneously, the data gathered during these tests was used to fine-tune the algorithms created at the beginning.

To sum up, the goal of the thesis is to describe the design process of an automatic robotic inspection system. Autonomous robotic systems equipped with advanced sensing and perception capabilities have revolutionized the field of industrial inspection. These systems, combined with artificial intelligence and machine learning, offer numerous benefits over traditional inspection methods. In the first chapter of this thesis, we introduce some general aspects of these systems and the required theoretical background. Next, the methodology and the adopted solution are presented. We specifically describe the architecture, the data management, the missions' organization and two of the computer vision algorithms. Then, the results and the related analysis are presented. In the last chapter, we draw some conclusions and identify some key areas of focus for future works.

Chapter 2

Overview

2.1 Autonomous robotic plant inspection

Figure 2.1: Spot robot from Boston Dynamics performing plant inspection.

Robotic technologies have made tremendous progress in recent years, transforming entire industrial sectors, and revolutionizing the way we live and work. From manufacturing to healthcare, transportation to entertainment, robots are playing an increasingly important role in society [1]. Moreover, when robotics is combined with other technologies such as artificial intelligence, their use opportunities grow even more. AI allows robots to process vast amounts of data, make complex decisions, and adapt to changing environments in real-time. This has enabled robots to perform a wider range of tasks with greater precision and efficiency, from autonomous vehicles that navigate busy streets to drones that inspect infrastructures. Collaborative robots are also becoming more common in manufacturing and other industries. They work alongside human workers, performing repetitive or dangerous tasks while the humans focus on more complex or creative work. This not only improves efficiency but also increases safety for workers.

Overall, automation continues to evolve at a rapid pace and opens new possibilities for many people. Even though there are countless applications for robotic technologies, autonomous or semi-autonomous robotic missions still stand out as a prime example of how these advancements can be utilized to their maximum potential.

Plant inspections refer to the process of assessing and examining the physical condition and operational efficiency of a facility or industrial plant, with the aim of ensuring compliance with relevant regulations, identifying potential hazards or risks, and detecting any defects or malfunctions that could affect production or worker safety [2].

As matter of fact, site inspections still rely heavily on manpower. Typically, they involve an operator physically patrolling the plant, gathering information from IoT devices or taking photographs. They must use proper equipment, exercise caution, and remain constantly attentive when performing any action. Hence, collected data need to be transmitted to diverse databases and systems, where analysts employ data analysis solutions to investigate large volumes of information. If the identification of problematic areas occurs, they write reports that are dispensed among pertinent staff members, in order to make business decisions as where to direct their focus and resources.

The whole process requires significant investments of money and time, especially in large industrial facilities, and the likelihood of human error and inefficiencies is high.

Although inspections are routine, they must be accurate, consistent, and reliable to meet safety, environmental, and performance standards.

The exclusive use of IoT sensors as solution may be too costly since it requires a sophisticated sensor technology, an IoT network infrastructure for energy supply and data transmission, as well as a reliable monitoring system. Many companies are beginning to use remote-controlled drones and robots, with the industry moving toward fully autonomous inspections.

It is quite clear that robotic plant inspections are changing the way industrial plants are inspected and maintained. Advanced robots are themselves equipped with sophisticated sensors and cameras, and navigate through the complex terrain of industrial plants, collect data, and identify potential issues before they become serious problems. With the ability to operate in a controlled manner and efficiently, robots can improve safety, reduce downtime, and increase productivity. From oil refineries to chemical plants, these robots are being deployed to perform inspections that are too dangerous or difficult for human workers to undertake and, are a cost-efficient alternative, with respect to fully sensor-based solutions, for consistent and reliable asset monitoring.

For data collection to be fully automated, an autonomous robot needs to move freely into the field and employ computer vision and artificial intelligence to collect data. Data collection systems could be automatically linked to the data analysis systems, ensuring that relevant information is consistently gathered and translated into reports in real-time. Presently, most systems can analyze data instantaneously, but they are not always capable of identifying which analytics should be run on which data to gain the desired insights.

Self-sufficient robots can monitor their battery level and return to the charging station as needed, eliminating the requirement for someone to retrieve it and prepare it for the next mission.

Currently, semi-autonomous inspections are more common than autonomous ones. This is attributed to various factors, including the challenge of executing certain tasks without human involvement at all. Nevertheless, the advantages of autonomous technology over semi-autonomous technology are widely acknowledged, particularly in terms of safety and efficiency. Consequently, the trend is rapidly shifting, as investments in autonomous systems continue to rise and adoption improves alongside technological advancements [3].

2.2 Mobile robots

Mobile robots are self-contained vehicles designed to move autonomously in a given environment. They have become increasingly popular in recent years due to technological advancements in sensors, computation, and communication, which have made them more intelligent and adaptable [4]. Mobile robots are used in a wide range of applications across various industries, spanning from manufactory to logistics, from healthcare to the military sector. They are used for tasks such as material handling, assembly, and quality control inspections.

One of the key advantages of mobile robots is their ability to operate continuously without the need for breaks or rest periods. This makes them ideal for tasks that require long periods of operation. Additionally, mobile robots can work in hazardous environments without risking human safety, making them ideal for tasks such as inspections or maintenance in areas such as oil rigs, pipelines, or underground tunnels [5].

Mobile robots are equipped with a large range of technologies including actuators, sensors, navigation algorithms, and control systems. Actuators such as hydraulic systems and motors enable the physical movements, sensors make robots navigate the environment, avoid obstacles, and detect and interact with objects. Sensors include cameras, LIDAR, sonar, and GPS, among others. The information gathered from them is processed by the robot's onboard computer, which determines the robot's next actions based on the surrounding environment. Navigation algorithms, such as collision avoidance, mapping, and path planning, help the robot determine the most efficient way to navigate, while the control systems use the navigation algorithms to determine how to control the actuators [6].

In recent years, mobile robots have become increasingly intelligent and adaptable, thanks to advancements in artificial intelligence and machine learning. These technologies enable mobile robots to learn from their experiences and improve their performance over time, making them more efficient and effective in their tasks.

Overall, mobile robots offer numerous advantages over traditional manual methods, including increased efficiency, accuracy, and safety. As such, they have become an

increasingly important tool for businesses and industries looking to streamline their operations and improve their bottom line.

There are several types of mobile robots that are particularly suitable for robotic inspections. The most common types are wheeled robots, tracked robots, legged robots, and aerial robots.

(a) (b) (c)

(d)

Figure 2.2: Wheeled robot (a), tracked robot (b), legged robot (c), aerial robot (d).

Wheeled robots are the most widely used type of mobile robot due to their simplicity, reliability, and cost-effectiveness. They are suitable for flat surfaces and can move at high speeds.

Tracked robots are similar to wheeled robots but have tracks instead of wheels. They are suitable for rough terrains, slopes, and uneven surfaces.

Legged robots have legs that allow them to move like animals, climb stairs, and traverse obstacles. They are suitable for complex environments such as construction sites and disaster zones.

Aerial robots, also known as drones, are flying robots that can inspect assets from above. They are suitable for large-scale inspections, such as power lines and pipelines.

When it comes to robotic inspections, the most suitable type of mobile robot depends on the specific application and the environment. For example, wheeled robots are ideal for inspecting flat surfaces such as floors and walls, while tracked robots are better for inspecting rough terrains such as construction sites and mines. Legged robots are suitable for inspecting complex environments such as power plants and disaster zones, while aerial robots are ideal for inspecting large-scale assets such as power lines and wind turbines.

Another factor to consider when selecting a mobile robot for robotic inspections is the type of sensors it uses. Mobile robots can be equipped with various sensors as said before, such as cameras, laser scanners, and thermal sensors. Cameras are the most common type of sensor used in mobile robots, as they provide a visual representation of the environment. Laser scanners are used to create 3D maps of the environment and measure distances. Thermal sensors are used to detect temperature variations, which can indicate the presence of leaks or overheating. The choice of sensors depends on the specific application and the type of asset being inspected. For example, cameras are ideal for inspecting surfaces and detecting defects such as cracks and corrosion. Laser scanners are suitable for creating 3D maps of complex environments such as mines and construction sites. Thermal sensors are ideal for detecting temperature variations in pipelines and tanks, which can indicate the presence of leaks or overheating.

In addition to sensors, mobile robots can also be equipped with various tools and accessories, such as grippers, manipulators, and cleaning tools. Grippers and manipulators are used to manipulate objects and perform tasks such as opening valves and turning knobs.

Finally, the choice of mobile robot for robotic inspections also depends on the level of autonomy required. Mobile robots can be classified into three levels of autonomy: teleoperated, semi-autonomous, and fully autonomous. Teleoperated robots are controlled by a human operator who remotely operates the robot using a joystick or a computer interface. Semi-autonomous robots can operate autonomously but require human intervention for tasks such as obstacle avoidance and path planning. Fully autonomous robots can operate without human intervention and can perform tasks such as navigation, obstacle avoidance, and inspection. The level of autonomy required depends on the complexity of the environment and the tasks to be performed.

When considering practical applications for these robots, their mobility system is a critical attribute that must be evaluated.

In the context of plant inspections, the two main options for robot types are wheeled and legged robots. Tracked robots are typically reserved for use in other industries, such as military, construction, and mining, as they are specifically designed for rough terrain.

Currently, wheeled mobile robots are more commonly used for plant inspections due to their ease of design, control, and manufacturing compared to legged robots. They are also

generally more energy efficient, faster, and less expensive, making them a cost-effective option for inspecting large plants. However, their use is limited as they are only able to move on flat surfaces and are unable to overcome certain obstacles. Legged robots, on the other hand, have the ability to climb stairs and navigate almost any uneven ground, making them a more versatile option for inspections.

The advantages of each system are outlined in table 2.1.

Comparison between wheel robot with legged robot.

Technical Criteria	*Wheel Robot*	*Legged Robot*
Maneuverability	X	✓
Transvers ability	X	✓
controllability	✓	X
Terrain Land	X	✓
Efficiency	X	✓
Stability	✓	X
Cost effective	✓	X
Navigation over obstacles	X	✓

Table 2.1: Comparison between wheel robot with legged robot.

As far as mobility and stability of movement are concerned, quadruped robots are superior to other legged robots [7]. The robot's four legs are readily manipulated, designed, and maintained and consequently, robotics companies such as Boston Dynamics, ANYbotics, Unitree Robotics, OR Ghost Robotics have developed quadruped robots that can navigate real-world environments with ease.

Based on the above, while performing the project discussed in the thesis, we decided to use legged robots for conducting inspections within the facility in exam, and in particular Spot from Boston Dynamics. Specifically, the inspections carried out in the plant entailed several crucial elements that necessitated the use of this type of robot. These include traversing various types of terrain, such as asphalt roads, rough terrain, and grass, in order to reach the location where the robot would have carried out visual interpretation tasks. The path also had some unevenness, such as climbing sidewalks or overcoming obstacles. Additionally, some of the meters that needed to be read during the inspection tour were elevated and could only be accessed via stairs. Finally, the robot might encounter dynamic obstacles along its path, such as vehicles or pedestrians, requiring it to react quickly and with maximum stability.

2.3 SPOT from Boston Dynamics

Figure 2.3: Boston Dynamics

Spot, developed by Boston Dynamics, is probably the most sophisticated quadrupedal robot available for industrial and commercial purposes [8]. Boston Dynamics began investigating advanced robot solutions in the early 1990s, and in 2017 and 2021, they released models that quickly gained recognition in the industry. These models are Spot Enterprise and Spot Arm. While sharing the primary characteristics and operating principles, the key distinction between these models lies in the payloads they handle. Specifically, Spot Arm features a manipulator mounted on its body structure, which limits the payloads capacity if compared to the more versatile Spot Enterprise. Spot Arm is still a version on which developers are bringing patches and therefore is not that reliable yet. On the other hand, Spot Enterprise is a staple for autonomous patrols and robotic inspections.

(a) (b)

Figure 2.4: Spot Enterprise (a) and Spot Arm (b).

From a dimensional standpoint, Spot Enterprise measures 1100 mm in length and 500 mm in width. Its height, indeed, is subject to variation based on the flexibility of the robot's leg joints. Specifically, it ranges between 520 and 700 mm while in motion, with a default walking height of 610 mm, and 191 mm while in a sitting position. Its net weight is 32.7 kg when the battery is mounted and can move with up to 14 kg payloads on the back. Additionally, it is worth noting that the robot possesses 12 degrees of freedom and can reach a speed of up to 1.6 m/s.

When considering the environment in which Spot operates, it is crucial to account for several factors, such as ingress protection (IP), operating temperature, maximum step height, and the ability to traverse slopes. Specifically, the robot is rated as IP54, which may prove to be a limitation for certain outdoor applications as it can only withstand light

rain and is not suitable for use in adverse weather conditions such as heavy rain or snow. Additionally, the operating temperature range of Spot is between -20° C and +45° C, which enables a broad range of indoor and outdoor applications but may also be a limitation as it is not suitable for extremely cold or hot environments. A notable capability of Spot is its ability to navigate and traverse changes in terrain, including a maximum step height of 300mm and the capability to traverse slopes between -30 and +30 degrees.

Spot is equipped with two plug-in batteries that can be inserted from below. Each battery has an approximate weight of 5 kg and can typically run for 90 minutes, although the actual runtime may vary depending on the tasks Spot performs.

Throughout its body, the robot has cameras and sensors that serve various functions. These cameras include black and white, color fisheye, range (depth), and infrared functions. The sensors are used for robot perception and obstacle avoidance, and they have a range of 2 meters from the robot's position [9].

Spot Enterprise supports both payloads from Boston Dynamics itself and third-party companies. Among the ones from the robot's mother company are worth mentioning Spot CAM+, Spot CAM+IR and Spot Core I/O.

The Spot CAM+ [10]equips Spot with a specialized camera, transforming it into a potent inspection tool capable of surveying remote or hazardous environments with emphasis on key inspection details. Its primary features comprise of a spherical camera offering a 360 x 170° view, a Pan-Tilt-Zoom (PTZ) camera with 30x optical zoom, and two-way audio speakers and microphones. Additionally, four pairs of LEDs provide illumination in dark environments, a roll cage protects the device from damage, and the device features protected cabling and sealed electronics. Configuration options are also available for front or rear mounting, and a USB port allows for the storage of image data.

To the basic version of the Spot CAM+ a thermal camera can be added to enable detailed thermal and visual inspections (Spot CAM+IR).

As regards the Spot Core I/O [11], it is used to enhance both the computation and communications available on the Spot platform. It connects sensors, cameras, and other devices to Spot, process the data collected into actionable insights, and relay those insights over 5G/LTE. Some main features are:

- Compact CPU and GPU with customizable inputs and outputs
- 5, 12 and 24V regulated power output
- RJ45 standard ethernet adapter
- Easy cable sealing to maintain IP54 rating
- Built-in 5G/LTE modem with CBRS support for private networks and option to use AT&T's public network
- Option to add LIDAR for enhanced autonomy.

(a) (b)

Figure 2.5: Spot Cam+ (a) and Spot I/O with LIDAR (b).

There are two modes in which Spot can operate, teleoperation and autowalk, with the key difference being the need for human intervention.

In teleoperation mode, Spot can be remotely controlled via a tablet. The tablet displays the robot's surroundings captured by its onboard cameras, and the operator can switch between different camera views. The operator can control Spot's movements, speed, body height, and issue commands like sitting, standing, or performing data gathering actions [12].

On the other hand, autowalk mode enables the recording and replay of a series of movements and location-based actions with the robot, referred to as missions [13]. The autowalk mode consists of two parts:

- Recording a mission: this involves teleoperating Spot through a route and creating actions to be performed along the way.
- Replaying missions: this involves Spot performing the same movements and actions as previously recorded, while adapting to minor changes in the environment.

Spot automatically sets waypoints along the path when a mission is being recorded. During the replay of the mission, Spot walks from one waypoint to the next. The criteria for placing waypoints are as:

• Waypoints are set at 2-meter intervals along straight paths.

• A waypoint is set within a 0.3-meter path segment if Spot turns more than 30 degrees or either its elevation changes by more than 0.3 meters.

• A waypoint is set when an action is recorded.

When replaying the mission Spot calculates its position by comparing features in its current sensor data with features in the data snapshots taken at each waypoint during mission recording. The base Spot platform tracks visual features within 2 m of the robot with its stereo cameras, but this range can be significantly enlarged with a LiDAR, thus improving Spot navigation capabilities. Spot automatically compensates for small changes in the environment adapting its route to avoid obstacles and attempting to complete the mission, but large discrepancies may require human intervention.

2.4 Digital image processing

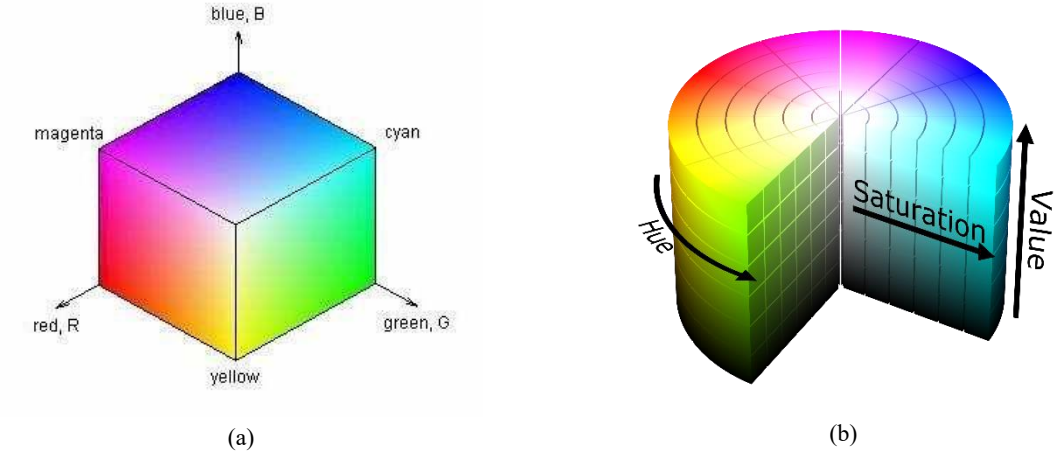
Image processing is a field of computer science and engineering that deals with the analysis and manipulation of digital images. It involves a wide range of techniques for improving, restoring, and analyzing images, and has numerous applications [14]. The goal of image processing is to extract useful information from images, or to enhance them in some way that makes them easier to analyze or interpret. To achieve these goals, image processing typically involves a combination of mathematics and statistics.

In computer vision, images are represented as 2-dimensional numerical grids, where each grid value corresponds to a pixel. Pixels are represented by a numerical value, which typically ranges from 0 to 255 and gives information on the brightness or intensity of the color at that pixel. In grayscale images, which use a single value per pixel to represent brightness and thus just a single image-matrix, a pixel with a value of 0 is completely black, while a pixel with a value of 255 is completely white. Colors, on the other hand, are represented using what are called color spaces, that are models employing typically three components to address colors as points in a coordinates system. There are several color spaces and all of them are commonly used in image processing and computer vision, to extract different features from the image.

The most common color space is the RGB color space. It is the standard to represent colors on most computer displays and cameras and represents colors as combinations of red, green, and blue intensities. RGB color space can be visually represented as a cube. The primary colors of red, green, and blue are located at three corners of the cube, while the secondary colors, cyan, magenta, and yellow are located at the other three corners. The origin represents black, and the corner farthest from the origin represents white. Colors within the RGB model are defined by vectors originating from the origin and extending to points inside or on the surface of the cube.

HSV is a color space designed to represent colors as we humans do. Its components are hue, saturation, and brightness, that are more intuitive to understand than the percentage composition of primary colors. HSV is often visualized as a cylinder, where the hue

component, corresponding to what we perceive as pure color, is represented by the angular dimension around the cylinder. The primary colors of red, green, and blue are located at 0°, 120°, and 240°, respectively. Saturation, indeed, which corresponds to the intensity of a particular hue, varies along the radial dimension. Colors with high saturation appear more vibrant and intense, while those with lower saturation appear weaker and tend towards gray. Finally, he third component is represented along the vertical axis and only affects the brightness of the color. The central vertical axis of the cylinder represents achromatic grayscale colors, with white at the top and black at the bottom.



(a) (b)

Figure 2.6: RGB color space (a) and HSV color space (b).

We describe below some algorithms used in image processing that are relevant for our work.

Spatial Filtering

Spatial filtering is a technique that applies a transformation to all pixel values, where the output of the filter depends on the single pixel and its neighborhood. Low-pass filters are a type of filter that is commonly used in image processing, essentially to blur an image for smoothing purposes. Spatial filters may utilize either linear or non-linear operations. Linear operations involve a 2D convolution between an image and a filter kernel across all color channels. The discrete 2D convolution of a kernel k with an image I(x,y), of size m × n, is defined as:

$$(k * I)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} k(s, t) \cdot I(x - s, y - t).$$

When performing spatial filtering on a multi-channel image, the convolution process is carried out independently on each channel, and then the resulting outputs are combined and concatenated.

The filtering action depends on the coefficient values and the size of the filter.

Gaussian filter applies a smoothing effect to an image to eliminate high-frequency noise. This technique is frequently employed as a preliminary step before running edge detection algorithms, as these algorithms are particularly affected by noise. The filter is circular and symmetric. The values of the kernel are computed by sampling from the Gaussian distribution and the resulting matrix is then normalized such that the sum of its elements is equal to one [15]:

$$k(s,t) = K * e^{(s^2+t^2)/(2*\sigma\hat{}2)} .$$

Bilateral filter is a non-linear method that reduces noise while preserving edges in an image. It is characterized by the following equation:

$$B(I,x) = \sum_{x_i \in \Omega} I(X_i) f_r I(X_i), I(X)) g_s(x_i, x),$$

where *I* is a k-dimensional image, Ω is the set of pixels *xi* in an n^k window, *fr* and *gs* are, respectively, the range attenuation and spatial attenuation weight functions [16].

For every pixel, the filter calculates a weighted average based on the spatial distance and the intensity difference between the pixel and its neighbors. To pixels with similar intensity values are given higher weights, while to those with large differences are given lower weights.

Thresholding

Thresholding is an image processing technique used for segmenting an image into regions, based on the intensity values of its pixels. This technique works by setting a threshold value and classifying each pixel in the image as either a foreground or background pixel depending on whether its intensity value is above or below the threshold, respectively [17]. This approach is typically employed when the intensity histogram of an image exhibits two dominant modes, and thus, by picking a certain threshold value T, is possible to separate them. The term global thresholding is used when the value T is constant. Conversely, variable thresholding refers to cases where the value of T varies across the image. In local thresholding, T at a given pixel is determined by the characteristics of the neighborhood surrounding the pixel. If T depends on the spatial domain, variable thresholding is referred to as adaptive thresholding. If the intensity distributions of object and background pixels are sufficiently distinct, a single threshold can be used for the entire image.

The output of thresholding is a binary image.

Otsu's method is a variance-based approach for determining a global threshold value. The algorithm performs an iterative search through all possible threshold values and selects

the one that results in the minimum within-class variance. The within-class variance is defined as the weighted sum of the variances of the two classes background and foreground, as shown in the equation:

$$\sigma^2(t) = \omega_{bg}(t) * \sigma^2{}_{bg}(t) + \omega_{fg}(t) * \sigma^2{}_{fg}(t),$$

where t is threshold value considered, $\omega_{bg}(t)$ and $\omega_{fg}(t)$ are the portion of pixels of the image respectively categorized as background and foreground when considering t as threshold, and $\sigma^2{}_{bg}(t)$ and $\sigma^2{}_{fg}(t)$ are the variance of the two sets of pixels [18].

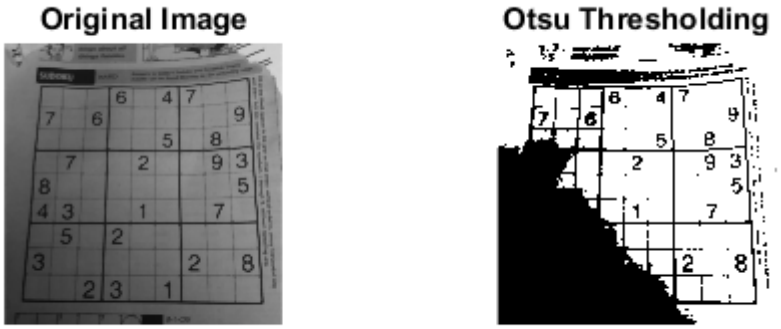


Figure 2.7: Example of a thresholding operation.

Morphological transformation

The fundamental concept of morphological transformation involves utilizing a structuring element, which resembles a kernel, to operate on specific parts of the input image [15][19]. Two basic operations are dilation and erosion.

Erosion is a morphological technique used to contract or remove small foreground regions in a binary image. In this operation, the structuring element is superimposed on each foreground pixel of the image, with the center of the structuring element aligned with the input pixel position. The value of the input pixel is then replaced with the minimum pixel value in the region delineated by the structuring element. As a result, if any of the neighboring pixels are background pixels, the input pixel assumes their value, otherwise it retains its original value. If we consider a binary image I as a rectangular array of foreground A (1s) and background pixels A^c (0s), and B as a structuring element, we can define the erosion operation as:

$$I \ominus B = \{z \mid (B)_z \subseteq A \ \& \ A \subseteq I\} \cup \{A^c \mid A^c \subseteq I\}$$

where $(B)_z$ is the structuring element translated by z, defined as:

$$(B)_z = \{c \mid c = b + z, \forall b \in B\}.$$

Dilation can be considered as the opposite of erosion and is typically used to fill gaps between sets of foreground pixels in a binary image, expanding the boundaries of white regions. In contrast to erosion, dilation superimposes the structuring element on each background pixel of the image. The value of the input pixel is then replaced by the maximum value of the pixels within the region defined by the structuring element. If at least one of these pixels is a foreground pixel, the input pixel receives that value. Otherwise, it remains a background pixel. The formal definition of dilation is given by:

$$I \oplus B = \{z \mid [(B)_z \cap I] \subseteq I\}$$

Other common morphological transformations include opening and closing operations.

(a) (b) (c)

Figure 2.8: Effects of erosion (b) and dilation (c) on the original image (a).

The *opening* operator is used to smooth object contours, remove regions that can not contain the structuring element, break narrow bridges, and eliminate small protrusions in an image. It is defined as:

$$I \circ B = (I \ominus B) \oplus B,$$

which states that the opening of image I by structuring element B is the erosion of I by B followed by dilation of the result by B.

Figure 2.9: Example of the opening operation.

In contrast, the *closing* operator generally fuses narrow breaks and long thin gulfs, fills gaps in the contour, and eliminates small holes. It also tends to smooth contours. The closing of image I by structuring element B is defined by:

$$I \bullet B = (I \oplus B) \ominus B,$$

which states that the closing of I by B is the dilation of I by B followed by erosion of the result by the same structuring element.

Figure 2.10: Example of the closing operation.

Opening and closing operations are dual to each other.

Convex Hull

The geometrical definition of convex hull says that it is the smallest convex set containing the shape in exam. Within the computer vision literature, there are numerous algorithms available for computing the convex hull of a cluster of points in an image. The one implemented in OpenCV uses an algorithm proposed by Sklansky [20], that has proven to be valid by the academics for the so called *weakly externally visible polygons.*

Weakly externally visible polygons are a class of simple polygons. A simple polygon is a two-dimensional shape that is bounded by a closed path, where no two edges intersect. An externally visible polygon, indeed, is one where any point inside the polygon can see at least one point on the boundary of the polygon. Finally, A weakly externally visible polygon is a special case of an externally visible polygon, where the visibility condition only applies to vertices and edges on the exterior boundary of the polygon.

The author describes the algorithm starting from a simple polygon like the one in figure 2.11(a), finding the extreme vertices in the horizontal and vertical directions, and dividing the plane in four triangular closed regions, where T, B, L, R refer to the top, bottom, left and right extremes, so that all points of the polygon are within the rectangle. The idea is to create four monotone chains within the regions R_i, in order to build the convex hull in

steps. These steps are shown for region R1 and then simply repeated for the remaining three regions. Assuming that two vertices, I and I+1, have been added to the monotone chain starting at L, the plane is divided into three regions, as depicted in figure 2.11(b). The subsequent vertex, I+2, is handled in the following manner:

1. If I+2 is not present in R1, it is rejected.
2. If I+2 is located in A3, it is retained and the process is repeated.
3. If I+2 is located in A2, or in A1 and positioned above I+1, and a vertex was discarded due to line 3 on the immediately preceding iteration, it is rejected.
4. If I+2 is located in A1, vertex I+1 is discarded.

Finally, if line 3 is executed, I+2 replaces I+1 in the next iteration, and if line 4 is executed, vertex I takes the place of I+1.

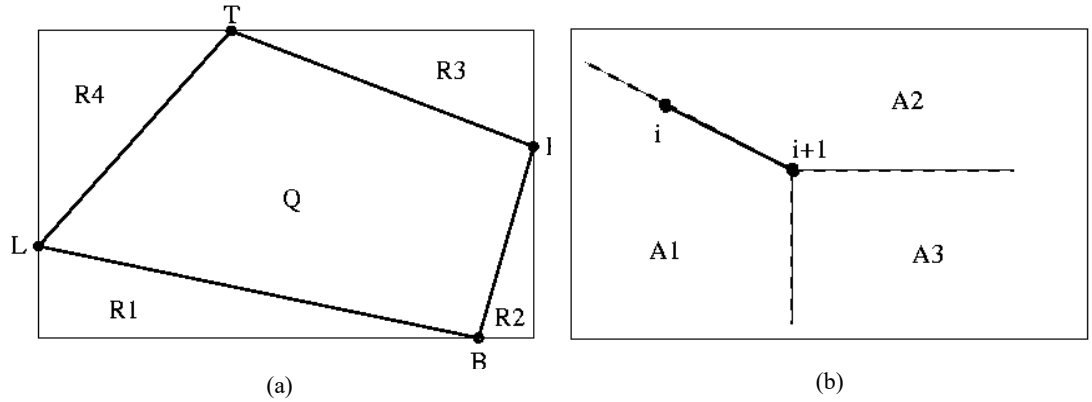


Figure 2.11: Simple polygon (a) and plan divided into regions (b).

Following what outlined above, a horizontal-vertical monotonic polygon can be obtained. However, any non-convex vertices must be still eliminated. Starting from one of the external vertices of the polygon and applying the following inequality, it is possible to determine whether a given vertex belongs to the polygon's convex hull or not.

$$S_i \triangleq (x_{i+1} - x_{i-1})(y_{i-1} - y_i) + (y_{i+1} - y_{i-1})(x_i - x_{i-1}),$$

where, x_i and y_i are the cartesian coordinates of vertex V_i.

V_i is a convex vertex if $S_i<0$, while if $S_i>=0$ Vi is removed and thus will not belong to the convex hull. Tracing the shape of the convex vertices, the convex hull of the starting polygon is obtained.

This algorithm is simple from a complexity point of view and easy to implement. However, it is less efficient than other algorithms for computing the convex hull, such as the QuickHull and Graham Scan algorithms, which have a faster average-case performance.

Figure 2.11: Example of convex hulls in white of the green contours.

2.5 Object detection with YOLO

Object detection is a crucial research area within the field of computer vision. Its objective is to identify and locate objects in an image or video by creating bounding boxes around the detected objects, and determining their position in the scene. These bounding boxes are rectangular in shape and indicate the object's location in the image. Object detection is a subfield of object recognition, and, like many related problems, relies heavily on deep learning techniques, including neural networks.

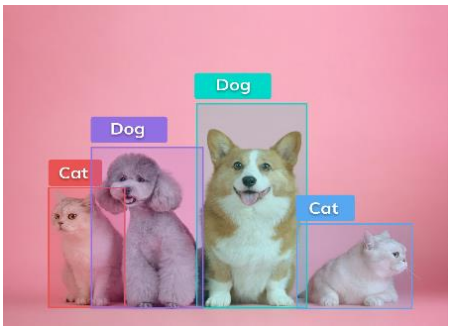


Figure 2.12: Object detection.

You Only Look Once (YOLO) [21] is a real-time object detection algorithm that relies on Convolutional Neural Network (CNN), a specific artificial neural network mainly designed for image classifications. It is widely known for its exceptional speed and accuracy in detecting objects. YOLO is a single-stage detector, it directly predicts the bounding boxes and class probabilities for each object in a single forward pass of the neural network. Traditional two-stage object detection methods, such as the R-CNN algorithms, first propose regions of interest using a Region Proposal Network (RPN) and then classify those regions. In short, YOLO divides the input image into a grid of cells and assigns each cell a set of bounding boxes. Each bounding box represents a potential object in the image and is associated with a class label and a confidence score. The confidence score reflects the probability that the bounding box contains an object and is computed based on the intersection-over-union (IoU) between the bounding box and the ground truth object. To classify objects in the image, YOLO adopts a multi-scale detection approach. It initially processes the input image at various scales and then merges the detections from each scale to generate the final set of bounding boxes. This allows YOLO to identify objects of different sizes and scales within the image. To improve the accuracy of object detection, YOLO integrates several techniques such as anchor boxes,

non-maximum suppression, and hard negative mining. These techniques enable YOLO to handle occlusions, overlaps, and other real-world image challenges.

Currently a stable version of YOLO is YOLOv5 [22]. A schematic representation of the architecture is following in figure 2.13.

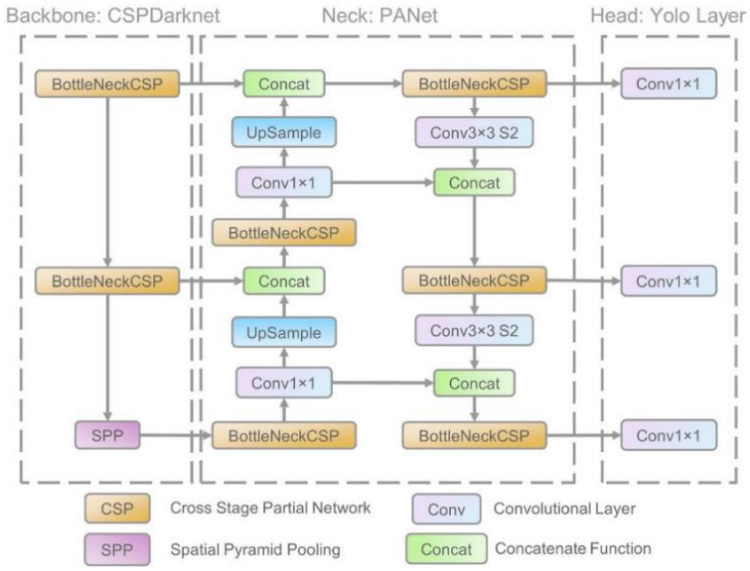


Figure 2.13: Yolov5 Network architecture.

2.6 Software and tools

In this section we present a list of software and tools used for the design of the robotic missions.

SPOT SDK

Spot SDK, or Software Development Kit, is the toolkit that allows developers to create custom applications for Spot [23]. It offers a range of features that enable to design and control Spot in ways that are tailored to the specific use cases. These include tools for controlling Spot's movements or setting and using payloads. The SDK includes a range of documentation and tutorials, as well as sample code that developers can use as a starting point for their own applications. Client applications utilize the Spot API to communicate with Spot's services via a client-server model. The applications can run on various platforms, including tablets, laptops, cloud-based programs, or payloads connected to Spot, as long as a network connection to Spot can be established through an IP network like the Internet, or a direct WiFi or ethernet connection to the robot. The majority of the Spot API uses gRPC as its application-level protocol, which was chosen for its secure, fast protocol that supports multiple programming languages and environments. The gRPC specification lists the supported remote procedure calls (RPCs) for the service. The Python library included in the SDK simplifies the use of Protocol Buffers and gRPC by providing a more straightforward abstraction. Clearpath Robotics

developed the Spot ROS driver, a powerful library built on top of the Spot SDK, which enables users to control and interact with the robot using ROS. The Spot ROS driver also allows for integration with other ROS-based systems and devices.

OpenCv

OpenCV (Open Source Computer Vision) is an open-source library of programming functions that specializes in real-time computer vision. It was initially developed by Intel in 1999 and later released under a BSD license in 2000. OpenCV has since become one of the most widely used computer vision libraries and is supported on various platforms. OpenCV provides over 2,500 optimized ready-to-use algorithms, both for low-level and high-level image processing. These algorithms can be implemented in C++, Python, and Java, among other programming languages [24].

The key functions that were used for the development of the automatic reading algorithms were:

- *cvtColor*: this function converts from one color space to another. It was mainly used for the conversion from the RGB color space to Grayscale and HSV.
- *threshold*: it segments a grayscale image using a threshold value. We used both BINARY and OTSU thresholding.
- *HoughLines*: this function detects straight lines inside a binary image using the Hough Transform.
- *GaussianBlur* and *bilateralFilter*: these functions are used to smooth an Image.
- *morphologyEx*
- *convexHull*: computes the convex hull of a set of points.

LabelImg

LabelImg [25] was used for the data labeling. This is a very easy to use tool for visual image annotation with a very intuitive graphical user interface. You can draw bounding boxes around the objects and then select the classes from a manually defined configuration file. The annotations followed the PascalVOC format.

ROS

ROS is an open-source set of software frameworks for robot software development that allow hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management [26]. It establishes a peer-to-peer network of processes that may be distributed across different machines or run on a single machine. The nodes are designed to function at a very small scale. A robot control system would often include a large number of nodes. A sensor, a motor, a processing algorithm or a monitoring algorithm can all be considered nodes in this network. The publisher-subscriber model is used to asynchronously pass messages between nodes via edges known as topics that connect them. Services that function similarly to remote procedure calls enable synchronous communication as well. A client

requests a service from a providing ROS node by sending a request message and then waiting for a response.
A ROS client library compiles srv files, which are used to define services, into source code. When a node is launched, it identifies itself to the ROS Master, whose main goal is to establish topic-based node-to-node communication and manage parameter server updates. Service requests and messages do not go through the master. The ROS Master is launched by running the roscore command. The Master also provides the Parameter Server which is used by nodes to store and retrieve parameters at runtime. This architecture brings some perks like fault tolerance due to node isolation, reduced code complexity compared to monolithic software and easy to use minimal APIs that hide implementation details. ROS currently supports TCP/IP-based and UDP-based message transport. The default transport layer used for ROS messages and services is called TCPROS. It uses standard TCP/IP sockets for streaming message data over persistent TCP/IP connections. Incoming messages are received over a TCP server socket with a header containing the message data type and its routing information. The ROS client library for Python is called rospy. Python programmers can use it to easily interact with ROS topics and services. Rospy's architecture prioritizes implementation speed over runtime performance to enable speedy prototyping and testing of algorithms within ROS.

AWS EC2

Elastic Compute Cloud is a service provided by Amazon that allows you to launch as many virtual servers as you need, configure security and networking, and manage storage. EC2 has become almost a standard solution when it comes to quickly deploy some application on a server given the ease of use when it comes to scale up or down to handle changes in requirements or spikes in popularity. A user can configure, create, launch, and terminate as many virtual machines as needed, also called instances. The main component of an instance is a read-only filesystem image that includes an operating system and any additional software required to deliver a service [27].

Docker

Docker was invented to solve the problem of dependency hell. Modern applications are built from existing libraries and frameworks and depend on other services and software. These components each have their own set of dependencies, some of which may be incompatible with those of other components. Docker addresses the issues of conflicting dependencies and missing dependencies by packaging all requirements with the program in a container. Docker makes use of some powerful kernel-level technology and makes it available to the developer like Linux Containers, cgroups, and a copy-on-write filesystem. Docker provides tools to make working with and building containers as simple as possible. Containers isolate processes from each other. They can be thought as a lightweight analogue of a virtual machine. The foundation of Docker is made up of Linux Containers and LXC, a user-space control package for Linux Containers. Kernel-level namespaces are used by LXC to separate the container from the host. The user namespace ensures that the root user of the container does not have root rights on the host by separating the user databases of the host and the container. Only processes running in the container, not those on the host, are to be displayed and managed by the process namespace. Moreover, the network namespace gives the container a virtual IP address

and a dedicated network device. Virtual machines virtualize at the hardware level, whereas containers virtualize at the operating system level. Containers, on the other hand, allow to access protected portions of the operating system. Because each container has its own abstracted networking layer, processes, and other components, two containers running on the same operating system are unaware that they are sharing resources. An extension for creating and operating multi-container Docker applications is called Docker Compose. The services of the application are configured using a YAML file. Then they are created and started from this unique configuration with a single command. It has commands for controlling the application's entire lifecycle: starting, stopping, and rebuilding services; checking the status of running services; streaming the log output; executing a command on a service [28].

PyTorch

PyTorch is a machine learning library that is both fast and easy to use. It supports an imperative and Pythonic programming style, allowing for models to be directly written as code, making debugging simpler. Defining layers, composing models, loading data, running optimizers, and parallelizing the training process are all expressed using the standard coding concepts of general-purpose programming. It is consistent with other scientific computing libraries. It is efficient and it supports hardware accelerators like GPUs and TPUs. The library is based on a dynamic computational graph, which allows users to change the graph's structure on the fly and perform memory-efficient computations. PyTorch prioritizes interoperability, enabling users to easily exchange data with the many Python external libraries. It performs reverse-mode automatic differentiation, which computes the gradient of a scalar output with respect to a multivariate input. PyTorch is mostly written in C++ for high performance, with the core libtorch library implementing the tensor data structure, GPU and CPU operators, and basic parallel primitives, as well as providing the automatic differentiation system, including gradient formulas for most built-in functions [29].

Chapter 3

Methodology and adopted solution

The overall project presented in this thesis was very challenging to manage, because it required significant effort and resources, and often involved complex technical issues to be solved. The team of engineers worked collaboratively and systematically to explore possible solutions for implementing autonomous robotic inspections in industrial plants, their sustainability, and the potential expansion to similar scenarios to the one in exam.

In the present case, during the patrol, at predetermined checkpoints, the chosen robot had to recognize and read pressure gauges, graduated scale ampoules containing liquid, and digital displays. Moreover, it had to show the collected information on a dashboard, that operators could access to. In short, inspection tasks that were previously carried out by humans are now asked to be performed by robots.

The plant to be inspected was pretty wide, and the environmental conditions such as the presence of uneven terrain, steps and stairs to climb, led to the choice of a quadrupedal robot to perform the required tasks. The key requirements for the robot to be selected were to possess a technology that was already strong and secure, to be highly reliable and sturdy, the possibility to intervene as little as possible in automatic processes and organizing the work in a fluid manner. That resulted in choosing Spot Enterprise by Boston Dynamics. Spot allows to register an inspection mission once and repeat it as many times you want, with low degree of error and high adaptability to a dynamic environment.

Finding and interpreting indicators and devices along the route is the main job to be accomplished during the missions. At every stop where the robot must examine a component, it is known in advance which kind and where approximately each component is located. Spot is still asked to detect targets through cameras. The detection is performed thanks to a pre-trained deep learning model that had been fine-tuned over a custom dataset gathered inside the plant. This step may be both necessary to get a better shot of the indicator by adjusting the camera position and to crop the image in its correspondence with a bounding box surrounding it. To extract the indicator values and perform the interpretation task, it was decided to develop a unique computer vision algorithm per class of component.

Three missions with various checkpoints each were defined in order to segment the workload and recharge the robot battery.

Despite the convenience offered by using automation, unexpected events still needed to be addressed. Due to accessibility or readability issues, for example, some of the indicators that a human operator typically check in a same patrol could not be considered. In addition, there were common challenges that arose when the robot paused to capture

images, such as the likelihood of it being positioned slightly distant from the designated checkpoint and the photo having a different shooting angle. Moreover, all the images gathered and saved might or might not be processed by the corresponding image processing algorithm, according to the quality of the subject, varying for example with the weather conditions or the cleanliness of the lens crystal.

3.1 Data

An image dataset containing high-definition photos was built to train the reader detection algorithm of YOLO and to start developing the automatic reading algorithms. It was collected by means of 6 initial site inspections, and both phone cameras and Spot PTZ were used to gather the data to ensure a diverse range of quality and resolution. Most of the pictures that ended up being in the dataset were frames extracted from videos. A total of 14781 images were gathered.

The dataset was manually labeled with LabelImg. The annotations containing information about the label were saved in the PASCAL VOC format as XML files.

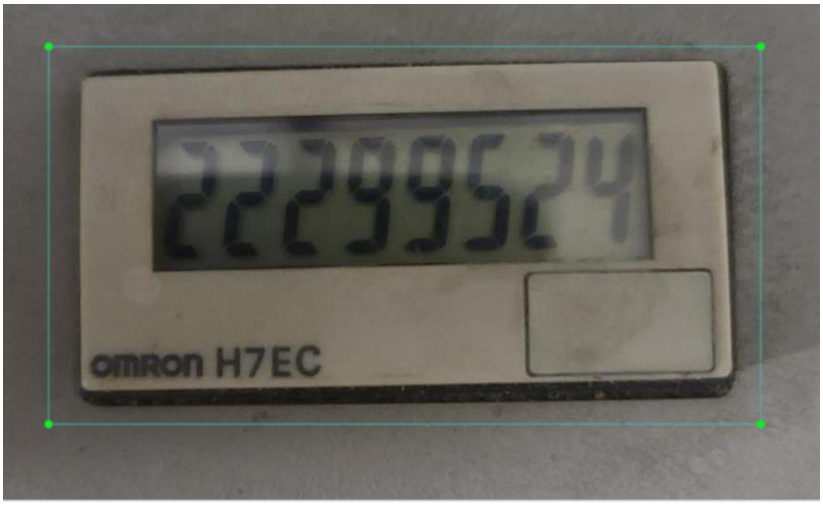


Figure 3.1: LabelImg usage example.

```
<annotation>
        <folder>Images</folder>
        <filename>20220902_091201_1.jpg</filename>
        <path>/home/Images/20220902_091201_1.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>700</width>
                <height>425</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>indicatore_digitale_rect</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>40</xmin>
                        <ymin>39</ymin>
                        <xmax>643</xmax>
                        <ymax>360</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 3.2: Pascal VOC annotation example.

As shown in figure 3.2, the annotation file contains information regarding the file path, the bounding boxes size and localization and the related classes.

Ten classes of devices were identified and used to label the dataset:

- “indicatore_analogico”
- “contatore_acqua_meccanico”
- “contatore_acqua_digitale_circle”
- “contatore_acqua_digitale_square”
- “contatore_acqua_generale”
- “vaso_espansione”
- “valvola_indicatore_di_stato”
- “indicatore_digitale_circle”

- “indicatore_digitale_rect”
- “livello_olio”.

Even if the class of the indicator was known before the shot was taken, based on the callbacks sequence during the missions, it was preferred to keep these classes for yet unknown future uses.

3.2 Meters detection

During any mission replay, the exact position of Spot when taking pictures was not the same from time to time, and so the shooting angle. To prevent some photos from being discarded because the target was not centered, the approach that was thought to be followed was to first take an initial shot that would surely contain a certain device and after that, a fine-tuned YOLOv5 model would be used to detect the exact position of the gauge inside the image. The coordinates of the device would then be used to point correctly the PTZ camera and take a better shot. The nature of the mission allowed to know which component was present in each shot, so that the only result of interest was the detection output, while the recognized class was discarded.

3.2.1 Training

The indicator detection model was fine tuned for a maximum number of 150 epochs starting from the Yolov5s weights using an image size of 512, batch size 32 and starting learning rate 0.01. The optimizer used was Stochastic Gradient Descent. Some data augmentation techniques were used like Gaussian Blurring and CLAHE. The dataset was split using a stratified strategy based on the classes keeping the following proportions 70%, 15% and 15% respectively. After 150 epochs the early stopping caused the training to stop and the weights from the last best epoch were restored.

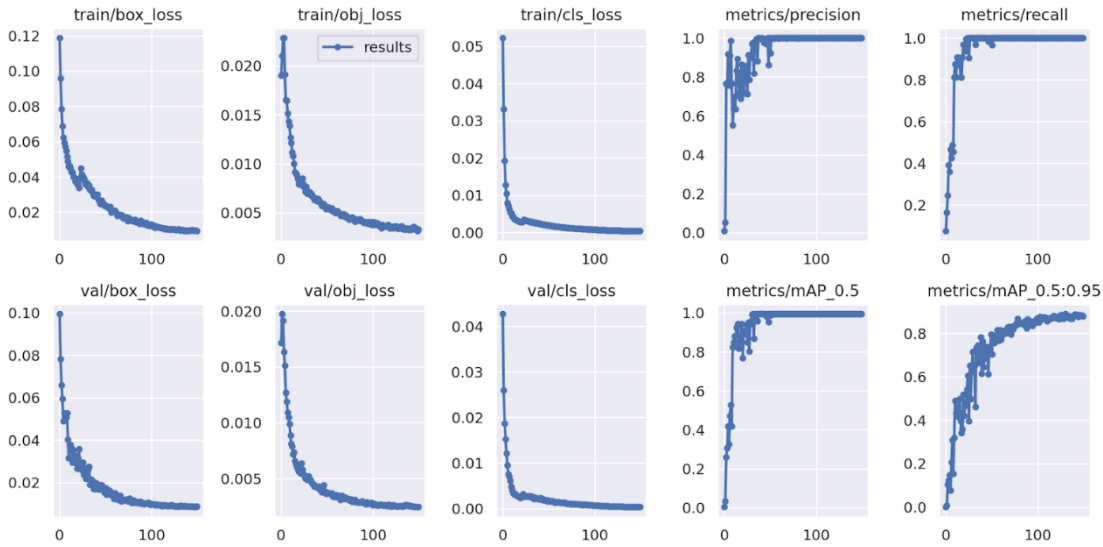


Figure 3.3: YOLO training and validation metrics for 150 epochs.

3.3 Autonomous Missions

Organizing the missions required a detailed on-site setup to deploy the intended solution with an in-field pilot. We configured the missions for three inspection tours within the plant and ran some on field integration tests. These initial on-site surveys proved highly beneficial in confirming the feasibility of the proposed solution and prompting a more comprehensive examination of all relevant components to be evaluated. Due to issues of inaccessibility or legibility, in fact, the number of items to be inspected had to be revised from the original plan. Moreover, we explored the potential of using the autonomous driving feature of Spot, and therefore the routes and points of interest on which to arrange the missions were revised. We thought for example how to avoid busy routs or how to make the robot climb stairs in maximum safety for the plant workers.

The inspections were named with the following codes: MISS01, MISS02, MISS03.

We kept carrying out autonomous missions with the goal of running at least eight autonomous inspections per day. In the initial phase of the pilot, MISS03 indoor environments were inaccessible due to ongoing maintenance, thus, we focused on registering and fine-tuning the first two missions.

To register a mission with the tablet for the Spot robot, we adhered to the following steps, manually piloting the robot:

- Power on Spot at the docking station;
- Connect the tablet to the Spot network;
- Open the Spot application on the tablet;

- ❖ Select the "Mission" tab at the bottom of the screen;
- ❖ Tap the "Create Mission" button;
- ❖ Start registration.

During the mission registration, at any point of interest, we performed the following actions:

- ❖ Stop Spot;
- ❖ Position it in such a manner that the photo from the PTZ camera was of high quality and aligned with the algorithm's requirements for all future mission repetitions;
- ❖ Store the pose parameters;
- ❖ Add a custom callback at the waypoint of interest.

After the last callback, Spot was returned to the docking station by following the permitted path, docked, and finally, the mission was considered completed.

Some of the major points of interest during the registration of the missions were related to the possible interactions between Spot and other agents like people or vehicles. During the mission Spot always walked along the pedestrian paths or sidewalks and crossed the road only when a crosswalk was available. The crossing of the crosswalks required some attention given the impossibility to signal in any human way its intention to cross the crosswalk.

During the replay of the recorded missions, we had to deal with a certain number of faults due to the robot, its firmware, the payloads and the tablet:

- During the mission replay, the robot had to walk near a wall and, even if the registration was rather smooth, it showed a rather strange behavior, as zig-zag walking. This was solved by replacing the robot with a newer one with an updated firmware.
- The tablet often showed connection faults which led to a sudden stop of the mission when replayed in supervised mode. This issue was solved by replacing the tablet with a newer version.
- The PTZ camera showed some random faults probably due to degraded connectivity to the robot. This was not solved during the in-site pilot but later on, replacing the camera with a newer model.

In the initial phase, we gradually refined several callback PTZ parameters in order to take pictures that would be more suitable for the algorithms to well perform. As result, the data that we gathered was organized according to whether the parameters were finalized or not.

Each mission came with a yaml file that defined the callbacks that needed to be executed during the replay. There was a one-to-one correspondence between the stop names defined during mission registration and the top level keys that were defined in the yaml file. For each callback could then be defined multiple spot wrapper operations that had to be considered. The naming of the operations to take followed the following structure: {optional_numerical_prefix}_python_method_name.

The spot_cam_ptz method required at least three parameters: pan, tilt and zoom.

The key environmental feature that determined the passage from a partial to a full autonomy was the need to open and close different types of manual doors that grant the access to indoor areas. Security requirements did not allow us to keep the doors open and thus the only possible ways to accomplish the task, were to replace the doors with automatic ones or to use a robot to open and close the doors. The first idea represented a deeply invasive approach that would cause a lasting impact on the environment and so the second one was the solution we decided to explore.

The only commercially available robot capable of such a task was Boston Dynamics' Spot Arm. The only pitfall was that this version of the Spot robot is capable of carrying only one big payload between the lidar and PTZ camera and provides much less space in terms of custom payloads that can be carried on its back.

At the end of the project, it was still not possible to add an action involving the autonomous interaction with a door inside a mission, so the experimentation had to be partially autonomous with the intervention of an operator in the two points of interest who would manually open and close the doors.

3.3.1 MISS01

This inspection route was the first one registered. It was the most influenced by weather conditions and lighting interference, so it was the one with the shortest daily time window in which could be executed. This was the longest mission in terms of distance traveled, with 11 readers of interest. The presence of one door made it only partially autonomous, because it was necessary to physically intervene, opening the door.

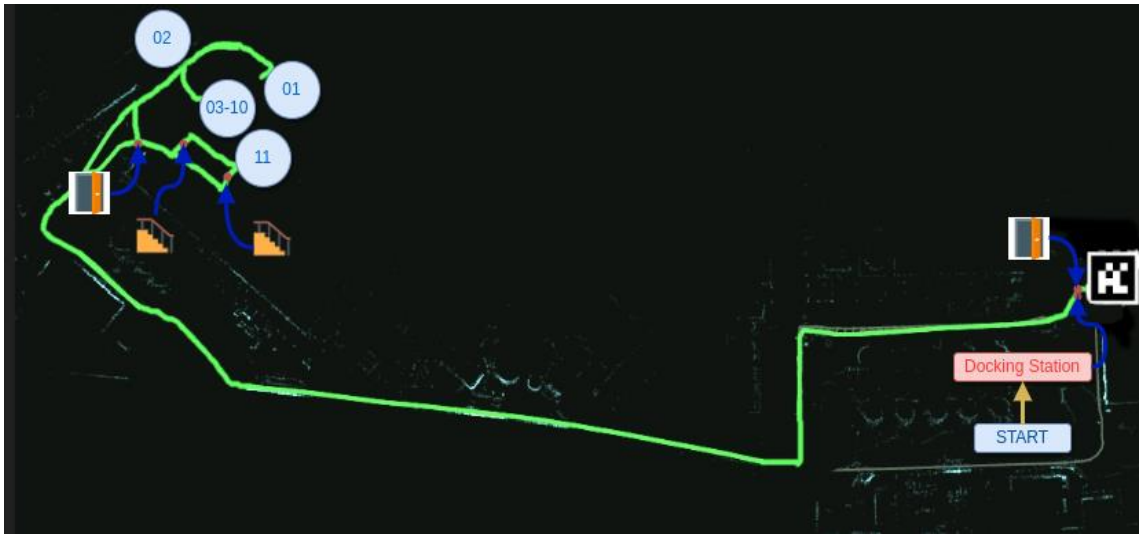
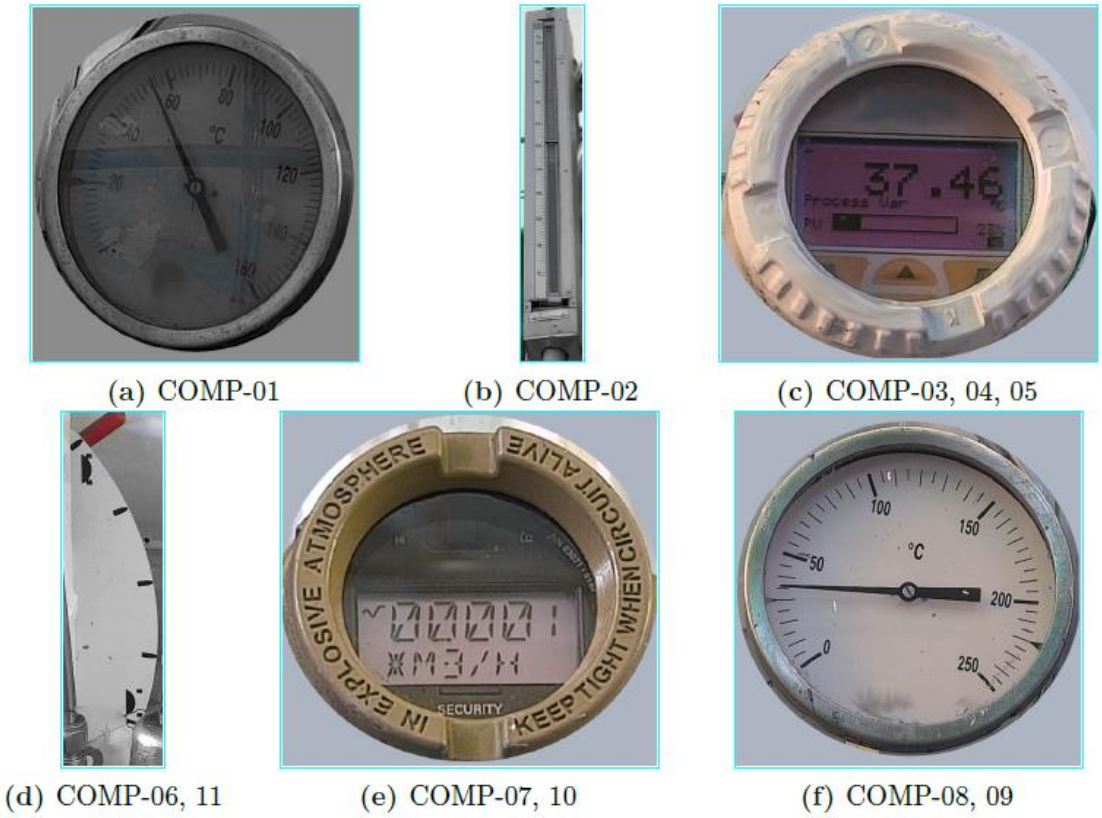


Figure 3.4: Mission MISS01 representation.

A total of 12 stops were planned: 11 to take a spot_cam_ptz callback, 1 for upload.



(a) COMP-01 (b) COMP-02 (c) COMP-03, 04, 05

(d) COMP-06, 11 (e) COMP-07, 10 (f) COMP-08, 09

Figure 3.5: MISS01 components

Checkpoint 1 was an analog gauge. The wear and the high exposure to sun caused it to turn orange. Furthermore, shadows were often present when there was a clear sky, for most of the day. No meter pointing was conducted because it could cause the inclusion of other close indicators. The original image was thus cropped just using the bounding box resulting from running the indicator detection model.

The second stop was for an expansion vessel level indicator. It was the only occurrence of this class in any inspection route. No pointing refinement was possible because it was very close to another instance of the same component. This checkpoint required some additional work to position both the camera and robot as best as possible because of the high location of the tool and narrow tiled path in front of it where Spot could navigate. The possibility of interpretation by a human operator was highly influenced by environmental conditions and lighting given how similar the liquid color and its background are. It was easy to detect under good lighting and a small margin was added to the bounding box in order to keep the component fully intact.

The next three checkpoints were round digital gauges. They were all located in the same spot outdoors. The first two required meter pointing to get a better shot, but the last one couldn't because it was very close to other components belonging to the same class. Given that there were so many components too close to each other, these indicators and the few following ones required particular attention to position the PTZ camera to only include the one of interest for each checkpoint. This kind of component was very easy to detect, so a confidence threshold as low as 0.5 allowed to always find it inside the image. A small margin of five percent of the bounding box was added to the crop in order to keep the perimeter circle intact.

The sixth checkpoint was a valve status indicator. Just as the liquid level indicator and the expansion vessel level indicator, this component only appears once across all inspection tours, and it could not be considered a gauge or display. The picture of this component did not require particularly high resolution given that there were few evident features that made it easy to interpret. The only point that required some attention was to avoid shadows over the plate in order to avoid confusion with the small black dots that indicate the extremes of the indicator. This component was not easy to detect for the indicator detection model mainly because the angle at which the picture was taken during the mission was rather different from the ones in the training dataset. With a very low confidence threshold we were still able to almost always detect it and then crop it with an additional five percent border in order to be sure to include the entire plate.

The next checkpoint was a round digital gauge. This one was one of the hardest to position because a protruding edge surrounding the gauge generates a shadow over it for most of the day. No pointing refinement was used over this component. Detection with confidence threshold 0.5 was used to generate the bounding box for the crop. A small margin of five percent of the bounding box was added to the crop in order to keep the perimeter circle intact.

Stops number eight and nine were analog gauges. The first was subject to major dirt which made it almost always impossible to read even by the human operator. The second was much more easily readable but it was located in a quite high position so the shot was taken at difficult angle. The configuration used was the same but for the algorithmic pointing which was missing for the first one. The detection was very easy in both cases so a threshold of 0.5 was good enough.

Checkpoint ten was a round digital gauge. The one problem with this component was the reflexes that could appear over it, so the lightning conditions were the theme. The detection model was run with a confidence threshold of 0.5 and the crop was extracted with a 5% additional margin.

The last stop was again a valve status indicator. Just as in the previous stop this shot did not require particularly high resolution. Here we had to be careful not to include shadows over the plate. The angle at which the shot was taken was not very centered so it required some additional work to make it as good as possible. No algorithmic pointing was executed here given the poor performance of the detection model. We had to set the confidence threshold very low in order to always detect the component. The found bounding box required to add a very big margin of 30% of the width to always crop the full plate.

Finally in the last stop Spot uploads the result to the cloud, then back to the docking station following one of the two possible routes.

3.3.2 MISS02

This inspection route was the second registered because it required some further evaluation regarding the access to the indoor environment. It had 14 devices of interest and 15 shots had to be taken. Most of the components were located indoors. The presence of one door made it only partially autonomous, because again it was necessary to physically intervene, opening the door.

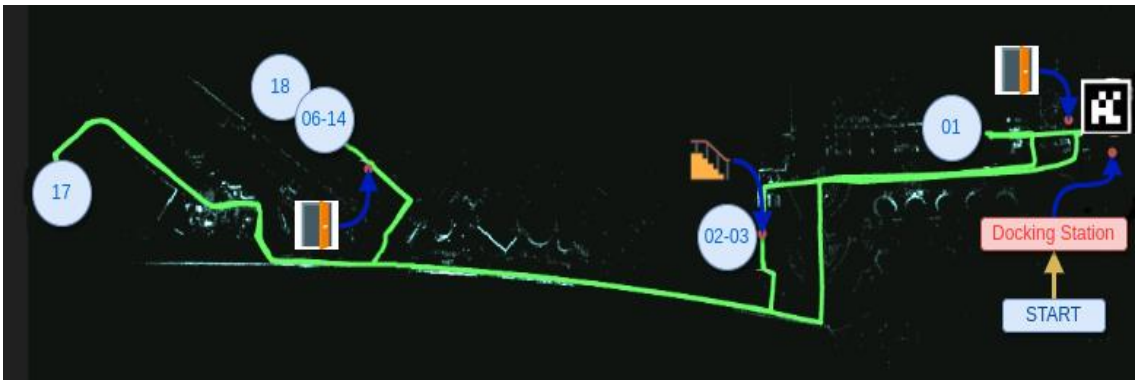
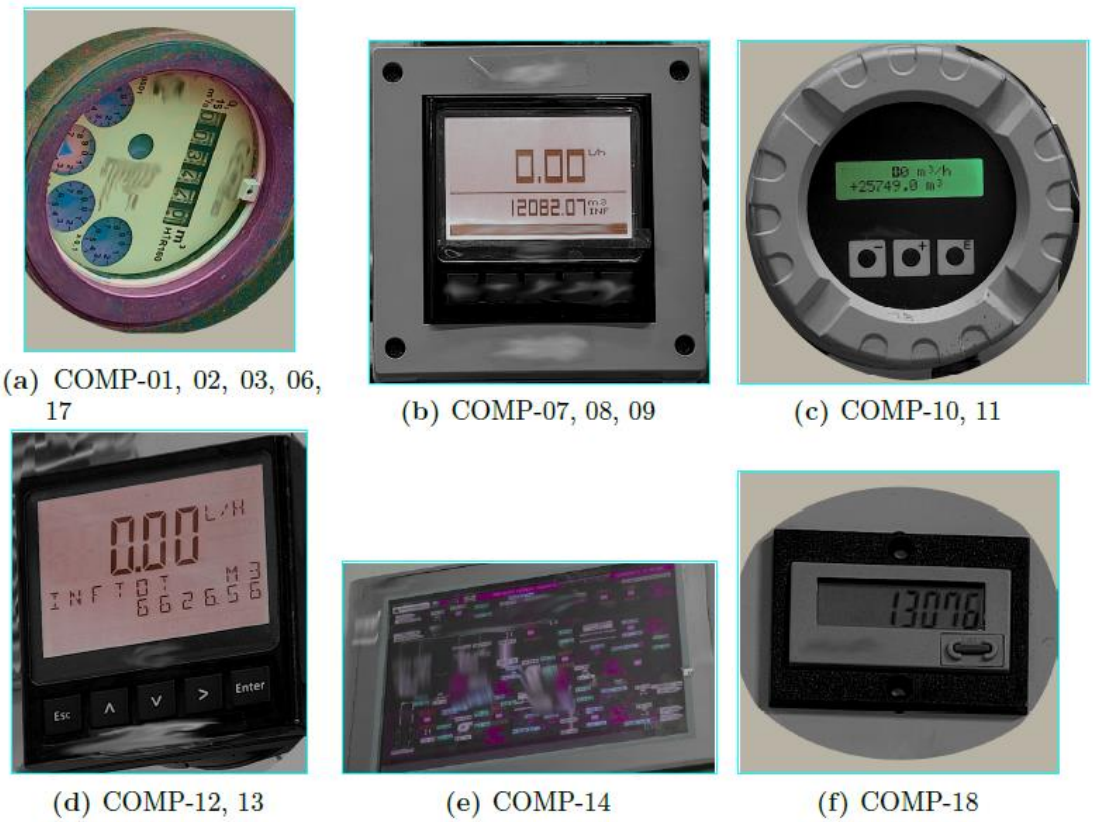


Figure 3.6: Mission MISS02 representation.

A total of fifteen stops were made: 14 to take a spot_cam_ptz callback (at stop 14 two callbacks with different PTZ parameters), 1 for upload.



(a) COMP-01, 02, 03, 06, 17 (b) COMP-07, 08, 09 (c) COMP-10, 11

(d) COMP-12, 13 (e) COMP-14 (f) COMP-18

Figure 3.7: MISS02 components.

The first four stops were mechanical water meters. The first three were in an outdoors setting, but the fourth was indoor. They all required camera pointing refinement via detection except for the second. This was due to major reflexes and shadows over the reader that made it hard for YOLO to correctly detect all the times. Consequently, the detection confidence threshold was kept very low at 0.01. Even if the detection was not successful, the appropriate automatic reading algorithm was executed on the full original image. In all cases a small margin was added to the bounding box used for the crop in order to keep the perimeter of the reader intact.

The next three items were square water meters stacked on top of each other with a very small gap to separate them from each other. In no case was used pointing refinement because an eventual zoom out could have include one of the other readers in the image and cause confusion in the correct reading to extract at each stop. The positioning of the PTZ parameters was carefully conducted to include one display at a time. This kind of display was very easy to detect so a high threshold was good enough to always detect all of them. A ten percent margin was added to the bounding box crop in order to help the automatic reading algorithm that expects some of the background around the display.

Stops 10 and 11 were round digital water meters. A 0.5 confidence threshold was good enough to always detect these displays. The former one required camera pointing refinement, while the latter did not because a zoom out could include other meters in the shot. A small margin of five percent of the image dimensions was added to the crop bounding box in order to keep the circle perimeter intact.

The next two were square water meters. These were slightly different from the previous one both in character font and display content. Both of them did not require the application of the pointing routine because this could have caused the inclusion in the camera frame of additional surrounding meters. Just as before these were very easy to detect for the custom reader detection model and a small margin was added to the crop bounding box in agreement with the assumptions made by its automatic reading algorithm.

Checkpoint 14 was an osmosis display. This was the only occurrence of the component in all inspection tours. The first approach that we tried to follow was to take one shot of the entire display and extract the three readings, but the numbers were almost always unreadable and out of focus. After some attempt we decided to switch to two shots of the region of interest corresponding to the upper left and lower right corners. No pointing refinement was possible here because the reader detection model was trained over the entire display. Therefore, the same went for the actual detection of the display. Consequently, the algorithms were applied over the original shot taken by the camera with no crop.

Checkpoints 15 and 16 were out of scope.

Checkpoint 17 was a mechanical water meter. No pointing refinement was not needed given that the quality of the first shot was already clear enough. Detection was rather easy for this component under this environment conditions, so a confidence threshold of 0.5 guaranteed a correct cropping.

The last stop was a rectangular digital display. This was initially out of scope because of its reachability by the robot, but once we started the infield pilot we were able to find a good position so we decided to include it back. At first, we thought we had to develop a new custom automatic reading algorithm but a few small adaptations to the rectangular digital display algorithm was enough to interpret this component as well. No pointing or detection was possible on this component given that it was not included in the data collection used for training the reader detection model.

Finally in the last stop the robot uploaded the result to the cloud, than back to the docking station following the same route.

3.3.3 MISS03

This inspection route was the last one registered due to the presence of a construction site and scaffolding that prevented access to the premises. This was the shortest mission with only 4 gauges of interest and 5 shots taken. All the components were located indoors. The presence of two doors made it only partially autonomous.

Figure 3.10: Mission MISS03 representation.

A total of six stops were made: 4 to take a spot_cam_ptz callback, 1 to wait for the opening of the second door, 1 for upload. In the first stop we made a spot_cam_set_led_brightness callback to set the led brightness to the maximum given the fact that the first shot is taken indoors with low environment light.

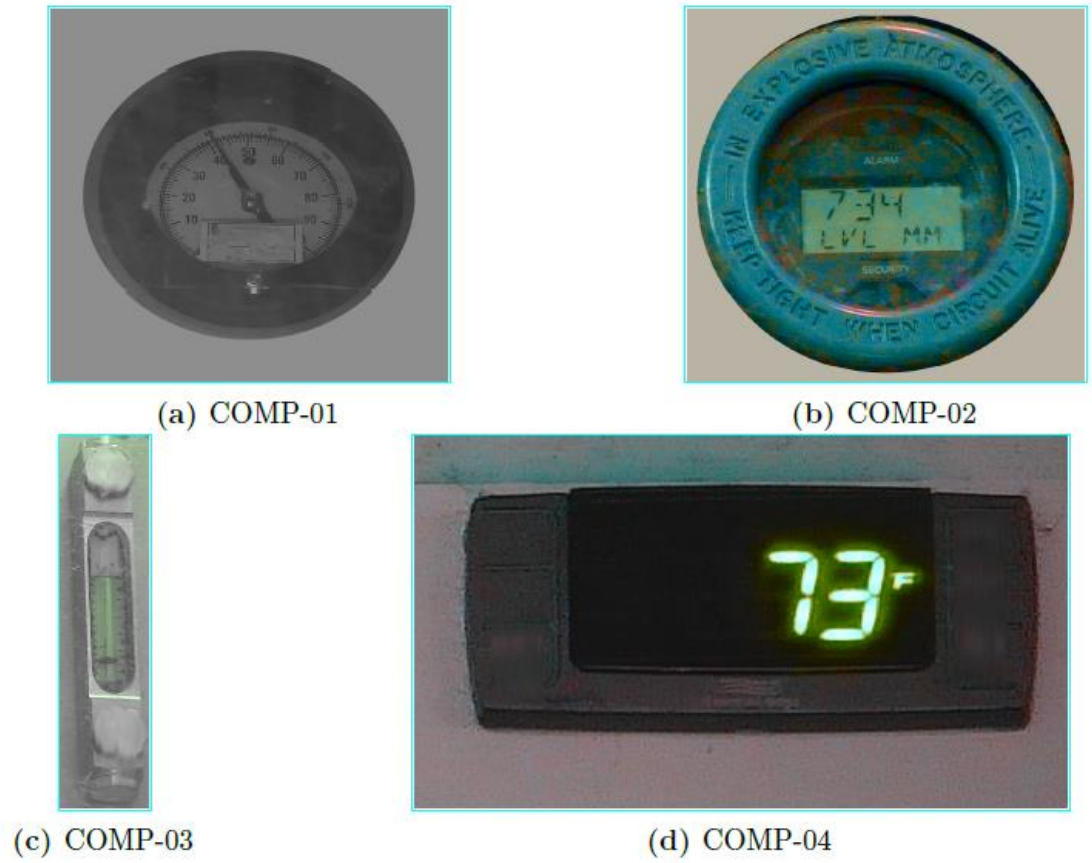


Figure 3.11: MISS03 components.

Checkpoint1 is an analogue gauge that required climbing some stairs to reach. It was behind a plastic glass and the only possible position was right below it. This caused the gauge to appear more like a very flattened ellipse rather than a circle. No detection for pointing refinement was made. A very low confidence threshold of 0.2 was required to detect and then crop the gauge from the taken picture. After taking the shot we lowered the led brightness to one sixth. After leaving the first indoor environment, Spot went outside and, after reaching the door of the second indoor space, waited for 10 seconds for the operator to manually open the door.

The second stop included two shots to be taken both of the same digital display at an interval of one second. This reader periodically changed. Both shots required pointing refinement via detection. This was a component easy to identify, so a threshold of 0.5 was good enough to always detect it. A border corresponding to 5% of the bounding box was added in order to keep the round perimeter of the component intact.

The third component was out of scope, so we moved on to the fourth one. It was a liquid level reader. No detection for pointing refinement was made. This was a component easy to identify, so a threshold of 0.5 was good enough to always detect it. A border corresponding to 5% of the bounding box was added in order to be sure to keep both endings inside the crop.

The fifth component was a rectangular digital display. No detection for pointing refinement was required. The shot for this component was taken at a few meters distance which did not make it easy to identify, so a threshold of 0.2 was necessary to always detect it. A border corresponding to 5% of the bounding box was added in order to be sure to keep some of the surrounding wall inside the crop.

Finally in the last stop Spot uploaded the data to the cloud, then back to the docking station following the same route.

3.4 Algorithms

The creation of the computer vision algorithms required having the same general approach, following in particular this multi-stage pipeline starting from the shot of the robot: identification of the area of interest, focalization on a more limited area, and reading of the value. Hence, the target meters were organized into smaller groups, in order to create parameterized algorithms that could be modified through externalized parameters and could correctly work for different gauges and readers, and environmental conditions.

The refinements and finalization of the algorithms were based on the increasing dataset collected during the robotic missions. This was an iterative process where the performances were improving from time to time, with continuous adjustments making up for unsatisfactory results.

Text and non-text are the two main categories into which the collection of components could be divided into. Some of these show on a display alphanumeric values to be extracted, others do not and thus their reading is less immediate. Building algorithms that read text is not simple though, and it took time to figure out how to work around some problems that have arisen.

The main operation we had to deal with can be broken down into two tasks: scene text detection and scene text recognition. Scene text detection refers to the process of locating and identifying the regions of an image that contain text. The output of a scene text detection algorithm is usually a bounding box around each detected text region. Scene text recognition, on the other hand, involves recognizing the actual text within the detected regions. The output of a scene text recognition algorithm is the text string corresponding to the detected region.

Text reading algorithms were initially approached as an OCR task, typically used for scanned documents, but given the numerous different situations in which they needed to be applied and the constantly changing font and format of the text that needed to be extracted, the tested tools' performances were not that good. For this reason, we used rather a scene text approach, which produced much better outcomes.

The distinction between OCR and scene text recognition becomes apparent through the examples below.

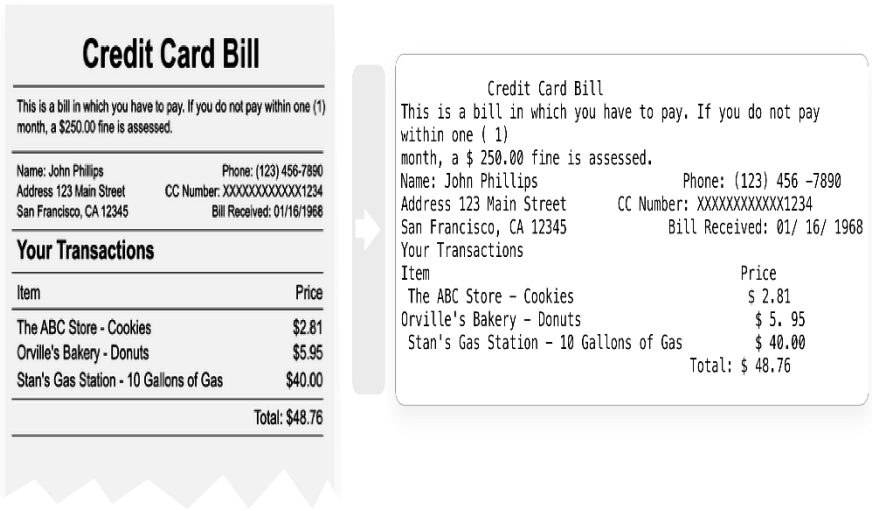


Figure 3.12: OCR.

Figure 3.13: Scene text recognition.

In non-text algorithms indeed, we used a case-by-case approach. This was very time consuming since the reading tasks were quite complex, even if the logic behind the scripts was standard, and reviewing similar case-studies in scientific literature helped a lot. We used classical computer vision techniques only and no tools but the OpenCv library.

One of the key considerations we made when developing the algorithms was the reliability of the results under a variety of lighting conditions and image quality levels. In cases where this didn't work, some parameters would be parametrized so that they could be defined a modified at convenience in a configuration file.

Python was used to develop all the algorithms. This was primarily because it has many production-tested tools and frameworks and is a very suitable programming language for quick experiment iterations. We were able to deploy the algorithms using a dockerized structure that enabled us to use various Python versions in order to avoid compatibility issues.

Two examples of computer vision algorithms developed are presented in the upcoming pages.

3.4.1 Gray round water meter reader

Gray round water meter reader is the algorithm used to read values from the display of indicators such as the one shown in the figure 3.14.

This type of meter reads different types of quantities. In our case, a measurement of the height of a volume of liquid was indicated, which was expressed through two different values, once in an absolute manner, with mm as unit of measurement, then in percentage terms, with respect to the maximum value of the storable water height volume. The values were shown on the screen at regular time intervals, and that is what made the reading routine tricky. In fact, in the management of the robot missions, a second checkpoint was added only at a later stage, just to be sure that Spot could take a photo of the percentage value at least once. The designed algorithm can read each value correctly supplied, although it has been decided to accept as good only the readings of the percentage values, in order to have useful information about the working status of the machine to which the device was attached to. It was impossible to organize the checkpoints to the second, so sometimes, unfortunately, a photo of the absolute volume indication was collected two times out of two.

Figure 3.14: Indicator type in exam.

The device was placed in a controlled room, indoors, in a not too bright space. Given the light conditions, although there were same indicators to be read during other missions, it was preferred to create a custom algorithm for this precise callback. Looking at an image shot during a mission, shown in figure 3.15, it is easy to notice the effects of environmental conditions when compared to a stock photo like the one above.

Despite the darkness of the scene, Spot took good photos, with consistent quality results. It was immediate to think at creating an algorithm using features from the color space and not working on the shape of the device.

Even if noisy, it was often well centered and readable.

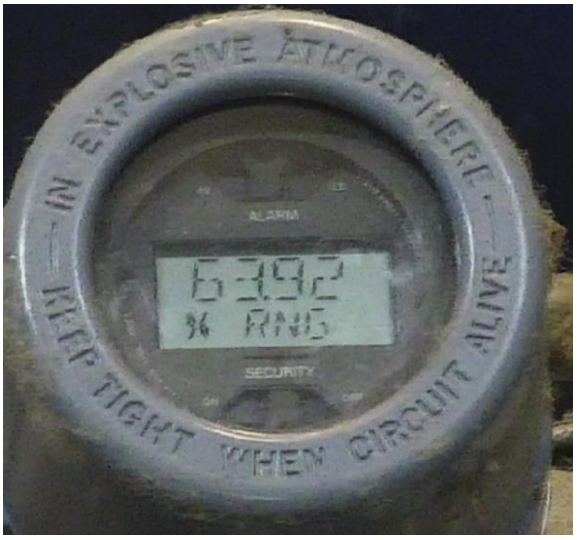


Figure 3.15: Photo of the meter shot by Spot.

.

In general, the logic behind the algorithm consists of two main steps:

- Extraction of the digital display from the indicator
- Running an OCR tool on a cropped portion of the screen

In particular, the procedure was the following:

- *Bilateral filter*
  First of all the image was prepared to be processed through a strong intervention of noise removal through the bilateral filter function. Since the kernel of the function used is particularly large, the output image was almost blurry to naked eye (fig X). Usually less sophisticated tools work on the difference between pixels in the spatial domain. Thanks instead to the peculiarity of the bilateral filter of preserving information also on the pixels intensity domain, the result is particularly acceptable if considering having to use in few steps a color mask.

- *HSV convertion and color masking*
  The image gets transformed from RGB to HSV. The HSV color space separates the color information (hue and saturation) from the brightness information (value), making it easier to manipulate the color and brightness of an image independently. HSV helps when performing color-based processing tasks such as segmentation, that means separating objects of a particular color from the background. An upper bound array and a lower bound array were selected through an algorithm posted by a user on Stackoverflow[30] and externalized to a configuration file. They correspond to those values that in RGB look like this sort of green/grey of the display. Using them to apply a color mask results in having a binary image showing the diplay, in white, on a dark background.

- *Morphology transformation*
  Morphology transform is used to close small black holes in in the foreground. It is common that some pixels do not fall within the color space selected for the mask even if, at first glance, it would not appear to be the case. In this particular scenario, indeed, those to be removed were the shapes of the digits shown on the screen.

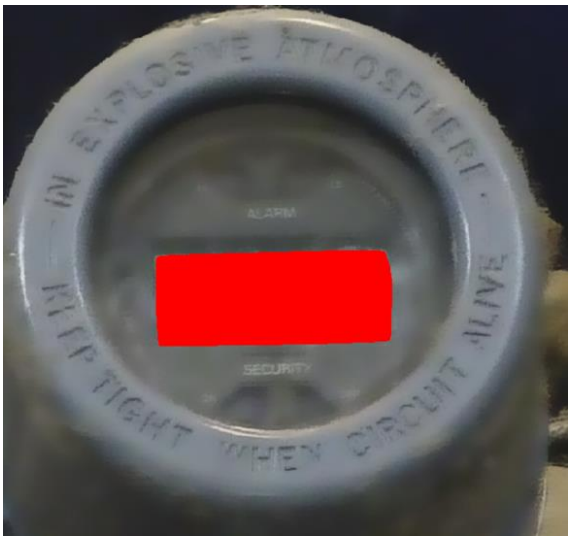
- *Find contour, sort the contour with the biggest area and obtain its convex hull*
  The binary image of the previous step is now used to draw the convex hull of the extracted contour of the display. The step through which you choose the contour with the biggest area is only necessary in the cases where the image noise makes the mask not work on the display uniquely and small groups pixels gets separated from the background as well.
-
- *Extracting the bounding box of the convex hull and cropping the photo*
  The contour and its convex hull are stored in memory as a list of coordinates. It is therefore easy to extrapolate the bounding box of the convex hull by looking for the minor and major abscissa point and the major and minor ordinate point to obtain respectively the left and right extreme and the lower and upper extreme of the bounding box. then the image is cropped using these coordinates.

- *Further cropping the image*
  The display presents digits and letters on two levels. Numbers are shown on the first line, units of measurement on the second one. Now the image is further cropped to obtain two distinct images, one for each line, containing a small text area only, to be then fed to the OCR tool. The parameters that regulate how much and how to crop the picture were found through a trial-and-error procedure and are stored in the configuration file, ready to be modified if needed.

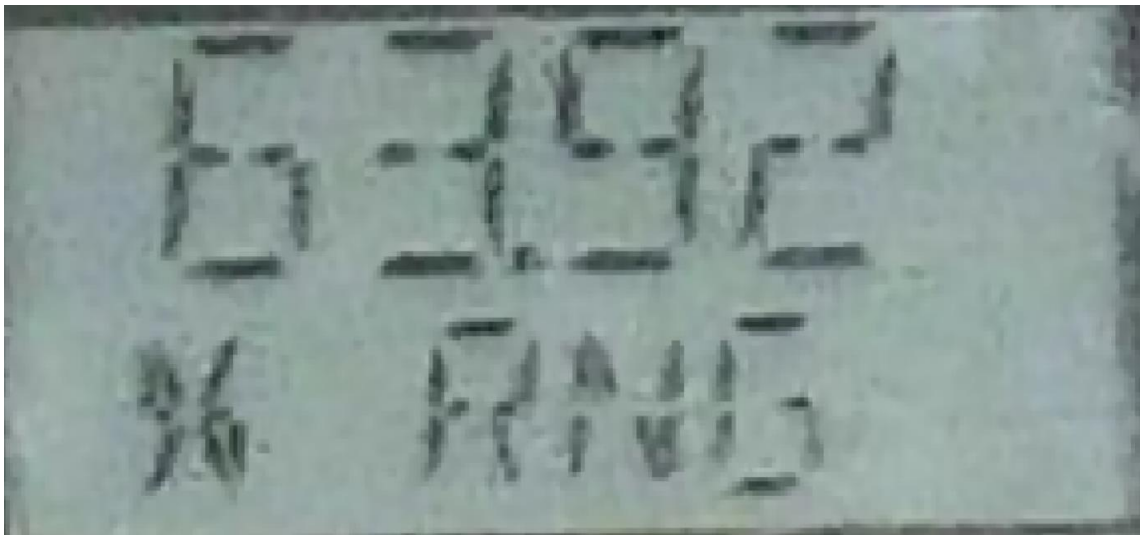
- *OCR performing and eventual string manipulation*
  The final step involves an OCR tool, Parseq in particular. The results are almost always acceptable, although errors may occur, and we need to fix them if possible. In percentage measurements, for example, Parseq sometimes does not read the comma, and a simple logic is ready to intervene. Other times, some numbers are mistaken for letters and vice versa, but since we know when to read numbers and when to read letters, an easy routine whitelists numbers when reading the first line, letters when reading the second line. Characters that are not alphanumerical, awkwardly read by the tool due to noise in the photo, are simply removed.

All the images in figure 3.16 are outputs of the “Gray round water meter reader” algorithm that are interesting to show, and useful for debugging purposes. The images shown in this case are produced by the algorithm's processing of the image at the beginning of the paragraph.

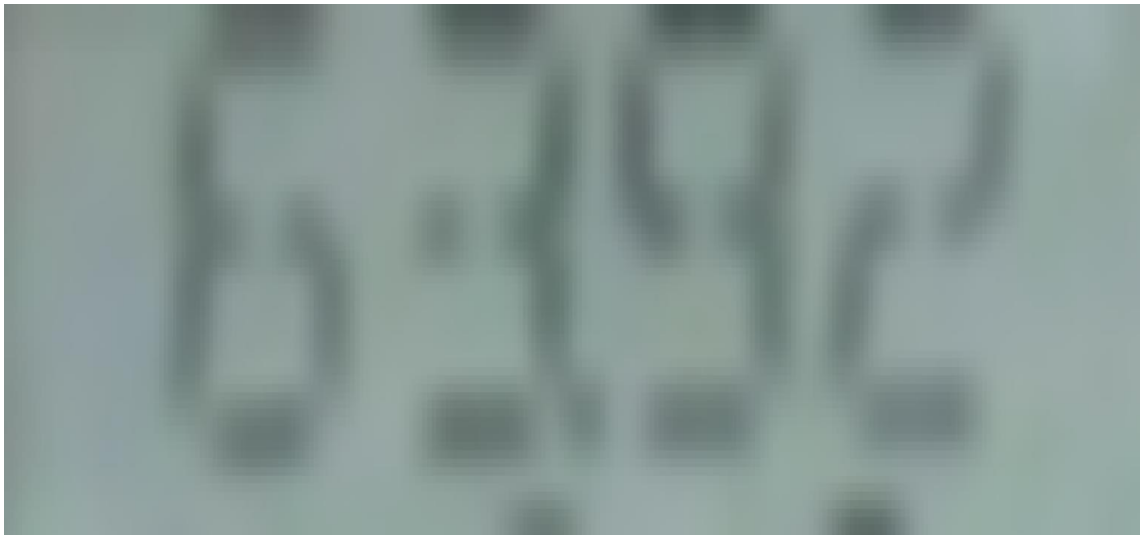
(a)


IN EXPLOSIVE ATMOSPHERE
KEEP TIGHT WHEN CIRCUIT ALIVE
ALARM
SECURITY

(b)



(c)



(d)

(e)

Figure 3.16: Outputs of the steps above described.

Here it is the pseudocode of the algorithm and a prototype of the JSON file storing the externalized parameters.

```
greyRoundMeterREader(image){

    image = bilateralFilter(image, 25, 75, 75)
    imageHSV = convertColorSpace(image, RGB_2_HSV)
    binaryImage = rangeMasking(imageHSV,
                               [h_lower, s_lower, v_lower],
                               [h_upper, s_uper, v_upper])
    binaryImage = closing(binaryImage, kernel=(9, 9), iterations=3)

```

```
    contours = findContours(binaryImage, RETR_EXTERNAL,
                            CHAIN_APPROX_SIMPLE)
    contourConvexHull = ConvexHull(largestAreaContour(contours))

    displayCrop = cropRectangleAroundTheContour(image,
contourConvexHull)
    measureCrop = CropUpperHalf(displayCrop,
measureCorrectionFactorHeight,
                                measureCorrectionFactorWidth)
    unitMeasureCrop = CropLowerHalf(displayCrop,
unitMeasureCorrectionFactorHeight,
                                    unitMeasureCorrectionFactorWidth)

    measure = textExtraction(measureCrop)
    unitMeasure = textExtraction(unitMeasureCrop)

    if any(i in unitMeasure for i in unitMeasureCharacters):
        if expectedDecimalDigits > 0 and "," not in measure:
            result = InsertComma(measure, expectedDecimalDigits)

    return result
}

textExtraction(image){

    image = image.resize((32, 128), BICUBIC.normalize(0.5, 0.5))
    text = parseqOCR(image)

    return text
}
```

```
{
    "h_lower": "42",
    "s_lower": "15",
    "v_lower": "133",
    "h_upper": "92",
    "s_upper": "255",
    "v_upper": "225",

    "measureCorrectionFactorHeight": 0.05,
    "measureCorrectionFactorWidth": 0.45,

    "unitMeasureCorrectionFactorHeight": 0.05,
    "unitMeasureCorrectionFactorWidth": 0.45,

    "unitMeasureCharacters": ["X", "%", "R", "G", "*"],
    "expectedDecimalDigits": 2

}
```

3.4.2 Liquid level reader

Liquid level detection is an algorithm used to read the level of the liquid in a graduated ampoule along a temperature scale as in figure 3.17.

Figure 3.17: Indicator type in exam.

The ampoule is contained by a metal frame, the liquid is a yellowish oil, and the graduated scale reads values from -20 °C to 100 °C. The unit on the scale is 10 °C and therefore we do not expect to read the device too precisely. A rough reading still provides good indications on the working status of the machine at which the device is paired, whether it is working at operating temperatures or not, and hence failures are about to happen.

The meter measures in particular the working conditions of a boiler set in a protected room. It is placed in a dark environment where the lighting conditions are poor. With correctly setting the robot while collecting the pictures, however, we had constant quality of images and thus the possibility to work on a precise set of photos in terms of colors and brightness, when running the algorithm. Moreover, this wasn't the case of photos damaged by weather conditions or shooting time because they had been collected during indoor missions.

Due not to being able to get too close with the robot to the device because of an obstacle, the ampoule was always photographed from afar and from a lateral perspective, as in the case of figure 3.18, forcing to work on lower quality photos than expected.
Moreover, the liquid was shading the graduated scale and therefore reading the numbers was not helpful in getting results.

Figure 3.18: Photo of the meter shot by Spot.

The method used to perform the measure was purely graphical: having a priori information on the working principle of the device like the extremes of the graduated scale or the typical working temperatures, the meter was readable if working on specific features of the photo.

The logic used was to compare the height of the liquid with respect to the total height of the ampoule, and at the end transform the ratio using a proportion to obtain the real value. The main tasks were:

- Detection and extraction of the ampoule

- Detection and extraction of the liquid column
- Temperature value calculation

The algorithm is described in detail below:

- *Cropping of the input image*
  Yolo almost always detects the instrument, but the bounding box that is really obtained is larger than what expected during the initial phase of the project, when we started building the algorithms.
  Since at the beginning we thought to use the same algorithm for two different devices, similar but not the same, the parameters regulating the cropping are externalized in a settings file. Later, however, only one graduated ampoule was included in the mission and therefore these parameters remained the same.

- *Conversion of the image into grayscale, Gaussian blur and Adaptive threshold*
  The image undergoes a blurring operation to remove noise after being converted to grayscale. The grayscale image is then adaptively thresholded to produce an initial mask, with the best threshold value being chosen with a trial-and-error procedure.

- *Morphology operations*
  With morphological transformation processes, the mask is cleaned up a little. The less dense white-pixels portions are removed keeping only the largest regions, while small holes or small black points are filled.

- *Finding contours and sorting the biggest one*
  Contours in the binary image are found, extracted, and saved as a list of coordinates. We are interested just in the contour with the biggest area, resulting to be the outline of the ampoule.

- *Convex hull of the biggest contour extraction*
  The convex hull of a contour is the smallest convex polygon that completely encloses the shape represented by the contour. We prefer working on a smooth shape rather than on the contour itself.

- *Extracting the upper and lower extremes of the bounding box of the ampoule and calculation of its length*
  Among the list of coordinates of the contour of the ampoule we extract those with greater and lesser component along the y-axis. Their difference in absolute value corresponds to the length of the ampoule.

- *Converting the cropped image in HSV*
  The cropped photo of the previous step this time is transformed into HSV because in this color space it is easier to distinguish the column of liquid from

the rest of the photo.

- *Gaussian blur*
  Again, Gaussian blur is applied to reduce the image noise.

- *Color masking*
  Color masking is what was used to select the yellowish pixels in the image. Through an algorithm posted by a user on Stack Overflow [30], it was easy to find the upper and lower limits of the color mask. It must be said that in any case the result varies a lot according to brightness and contrast, and therefore once again the values have been collected in the file containing the externalized parameters. The output of this step is a binary image, where just the pixels in the selected color range are highlighted.

- *Morphological operations*
  Cleaning the mask was necessary to maintain only the shape with the largest white-pixels area, which certainly corresponds to the liquid column. If not, most likely some isolated pixels still fall within the yellow range, even though it doesn't look like that to the human eye.

- *Finding contours, sorting the biggest one and convex hull extraction*
  The same procedure as before is repeated to extract the coordinates of the upper and lower extremes of the bounding box of the liquid column, corresponding to the points with lesser and greater y-axis value. Their difference in absolute value corresponds to the length of the liquid column.

- *Calculation the measured temperature value*
  By utilizing an appropriate conversion, it is possible to determine the temperature measured by the thermometer by comparing the length of the liquid column to the length of the ampoule. Figure 3.19 can help in comprehending this concept.

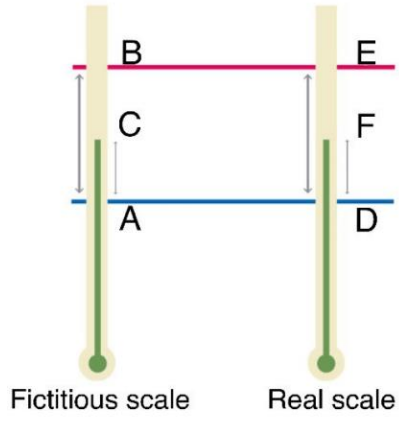


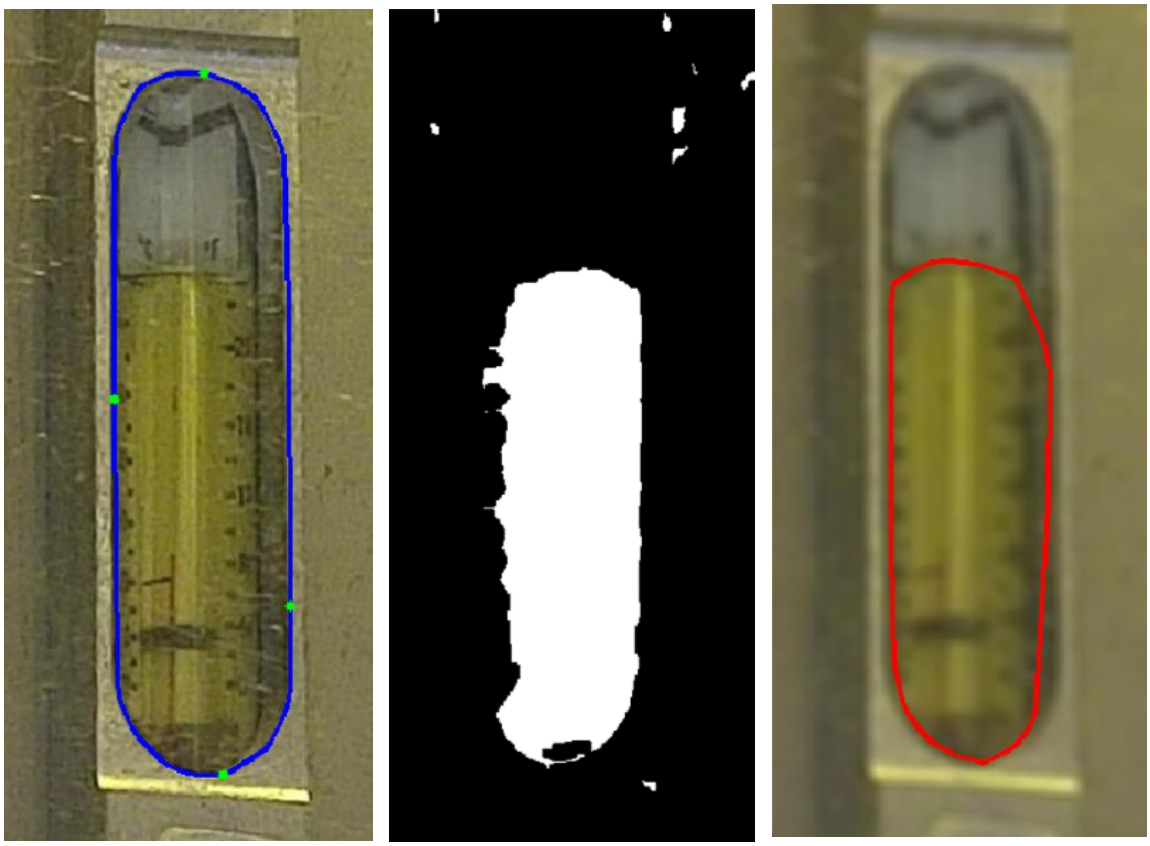
Figure 3.19: Graphic representation of the conversion scales.

We imagined having two aligned scales: the one on the left represents the percentage ration between the liquid length and the ampoule length in a graphical manner, the scale on the right represents the real scale at which the temperature is measured. Suppose that A and B represent the minimum and maximum values on the fictitious scale, namely 0 and 100, while D and E represent the minimum and maximum values on the real scale, -20 and 100 respectively. It is possible to determine the value F using the given formula:

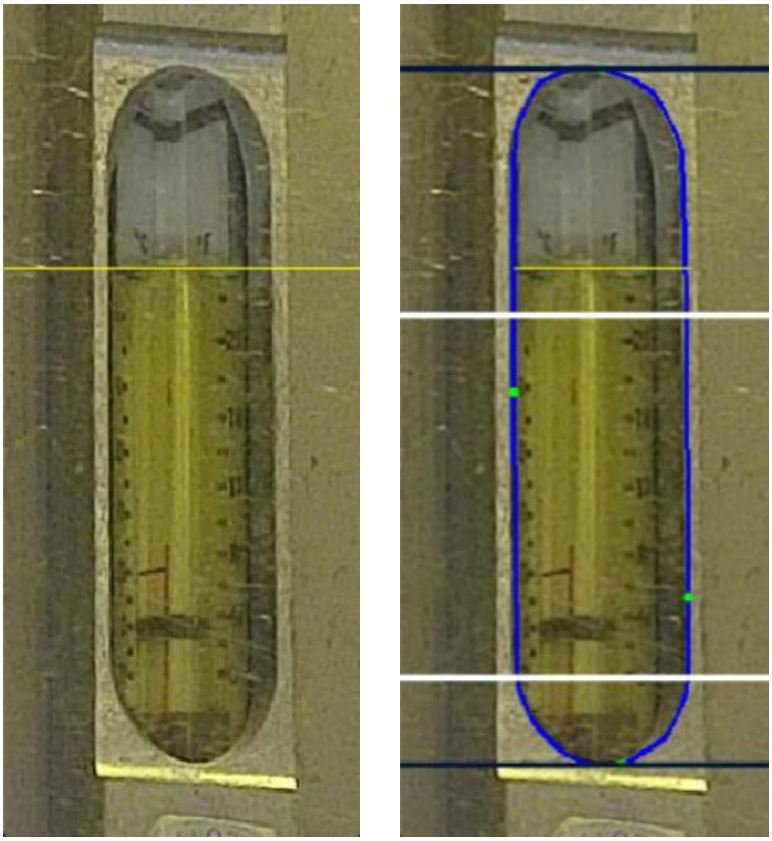
$$F = \frac{(E - D) * (C - A)}{B - A} + D,$$

where (B – A) is the length of the ampoule, (C - A) is the length of the liquid column, F is the real temperature value.

All the images from figure 3.20 are outputs of liquid level reader algorithm. They are interesting to show, and useful for debugging purposes. These images are produced by the algorithm's processing of the image at the beginning of the paragraph.



(a) (b) (c)



(d) (e)

Figure 3.20: Outputs of the steps above described.

Here it is the pseudocode of the algorithm and a prototype of the JSON file storing the externalized parameters.

```
liquidLevelReader(image) {

    image = image[(cropParameter_yBot*image.height):(image.height-
(cropParameter_yTop*image.height)), 0:image.width]

    grayImage = convertColorColorSpace(image, RGB_2_GRAY)

    blurredGrayImage = GaussianBlur(grayImage, (15, 15))

    thresholdedImage = adaptiveThreshold(blurredGrayImage, 255,
ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY_INV, 41, 2)

    thresholdedImage = opening(thresholdedImage, kernel=(3,3),
iterations=3)

    thresholdedImage = closing(thresholdedImage, kernel=(4,3),
iterations=2)

    contours = findContours(thresholdedImage, RETR_EXTERNAL,
CHAIN_APPROX_NONE)

```

```
    contourConvexHull = convexHull(largestAreaContour(contours))

    lenghtAmpoule = yBot_contourConvexHull-yTop_contourConvexHull

    yBot_contourConvexHullNew = yBot_contourConvexHull + (yBot_correctionCoefficient*lenghtAmpoule)
    yTop_contourConvexHullNew = yTop_contourConvexHull - (yTop_correctionCoefficient*lenghtAmpoule)

    lenghtScale = yTop_contourConvexHullNew - yBot_contourConvexHullNew

    imageHSV = convertColorSpace(image, RGB_2_HSV)

    blurredImageHSV = GaussianBlur(imageHSV, (11,11))

    binaryImage = 255 - rangeMasking(blurredImageHSV, [h_lower, s_lower, v_lower], [h_upper, s_upper, v_upper])

    binaryImage = closing(binaryImage, kernel=(3,3), iterations=1)

    contours = findContours(binaryImage, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)

    contourConvexHull = convexHull(largestAreaContour(contours))

    y_liquid = yBot_contourConvexHull

    y_liquidFictitious=(((yTop_contourConvexHullNew - y_liquid)*100)/lenghtScale)

    y_liquidReal = y_liquidFictitious * ((maxValueScale - minValueScale)/100) + minValueScale

    return y_liquidReal
}
```

```
{
    "h_lower": "0",
    "s_lower": "0",
    "v_lower": "0",
    "h_upper": "179",
    "s_upper": "115",
    "v_upper": "225",

    "cropParameter_yTop":0.19,
    "crop_parameter_yBot":0.3,

    "minValueScale": "-20",
    "maxValueScale": "100",
    "units": "°C",

    "yTop_correctionCoefficient": "0.355",
    "yBot_correctionCoefficient": "0.125"
```

```
18.
19. }
```

3.5 Proposed architecture

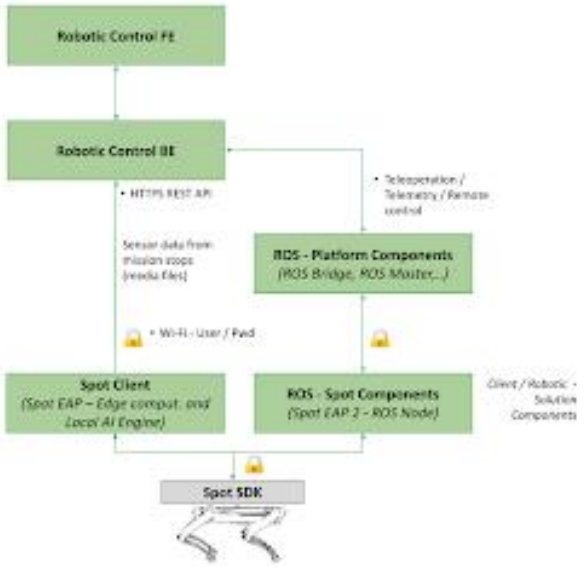


Figure 3.21: Proposed architecture.

The key component of the proposed architecture was Spot Enterprise robot as shown in figure 3.21. On its back it mounted the Spot EAP payload. A Docker container running on it acted as a client to the Spot GRPC API. This was used to send commands to the robot via custom controls developed using the Spot SDK, customizing a library called Spot ROS. Another container instead run a ROS node.

The second key component was an EC2 instance that acted both as the ROS master and as a server for a Python Web App. It allowed to send commands to the ROS node running on the Spot EAP for the purpose of teleoperation, remote control, and telemetry. The latter was a custom web application that has been developed to display the results gathered during a mission, invoking the ROS Services, and to have the possibility to control a fleet of Spot robots in future. The web application could also be used to specify the type of callbacks and the associated parameters to be used during a mission. These specifications had been defined during the mission registration phase and stored in a configuration-file that had been deployed on the Spot on-board computer.

The automatic reading algorithms were also deployed on a EC2 instance.
Given the freedom granted during the development of the algorithms both in terms of Python version and libraries adopted, the solution that we decided to embrace involved the deployment of the scripts as docker containers. The docker-files of the different algorithms followed the same template. First, we specified the Python parent image from which we were building, then we installed the dependencies using a specified requirements-file and, to conclude, we obfuscate the script using PyArmor. The docker images were built using a basic Docker Compose, while the docker runs have been launched by a Python script which navigates the folder containing all the mission replays outputs. The naming of the folders determined which category the image belonged to, so we could run the appropriate docker image. The Python script were launched by a CRON job, which is executed periodically every 5 minutes via the crontab program. The mission came along with as many settings files as the number of callbacks that needed to be executed. The docker run was done through the docker SDK for Python.

Consequently, every time new missions have been uploaded and the scripts have interpreted the gathered image, the results are displayed by the front end of the web application.

Chapter 4

Analysis and results

In order to obtain satisfactory outcomes, we needed to focus on examining the actual behavior of Spot during the plant patrols, and on the success rates of the algorithms performing the computer vision tasks. Hence, we questioned and troubleshooted the methodologies employed and gathered evidence demonstrating the feasibility of the proposed solution as proof of concept.

As soon as the on-site pilot started, it took some time to fix bugs and errors and thus, different experiments were conducted. Among these, the valid experiments accomplished both robotic and computer vision requirements, and in particular, the gauges' numerical values were correctly extracted. Spot had to be able to follow the predetermined path and take photos good enough to work on them. After completing the callbacks, it had to uploads these on the cloud, where the object detection model hat to narrow down the images around the area of interest satisfactory. Finally, the reading algorithm hat to precisely process the photos to obtain the numerical value. The image processing phase was automatically triggered once the photos were uploaded to the cloud.

Initial valid repetitions performed for each mission were still utilized to refine specific parameters, which were working properly but we felt we could improve, such as the PTZ positioning during callbacks. For this reason, they were excluded from the outcome analysis of the computer vision algorithms.

The data are presented in the form of charts and tables, to aid in the visualization and interpretation.

Table 4.1 displays information on the experiments conducted across the various missions and the division of the robot's total kilometers traveled. It also outlines the number of components involved in each mission. The total number of experiments carried out was 141.

Table 4.1: Count of missions' repetitions and kms travelled.

Patrol route Id	Number of components	Number of repetitions	Km travelled
MISS01	11	57	54.2
MISS02	16	44	38.1
MISS03	5	40	10.9
Total	32	141	103.1

MISS01 had the highest number of tests performed. This can be attributed to the fact that it was the first mission recorded, and thus the first robotic issues faced were fixed during this time. On the contrary, fewer experiments were conducted for MISS03. The area in which it was held was in undergoing maintenance work in the early testing stage, resulting in it being recorded later than the other two missions.

Detailed information about the number of repetitions of the missions vs days is given below.

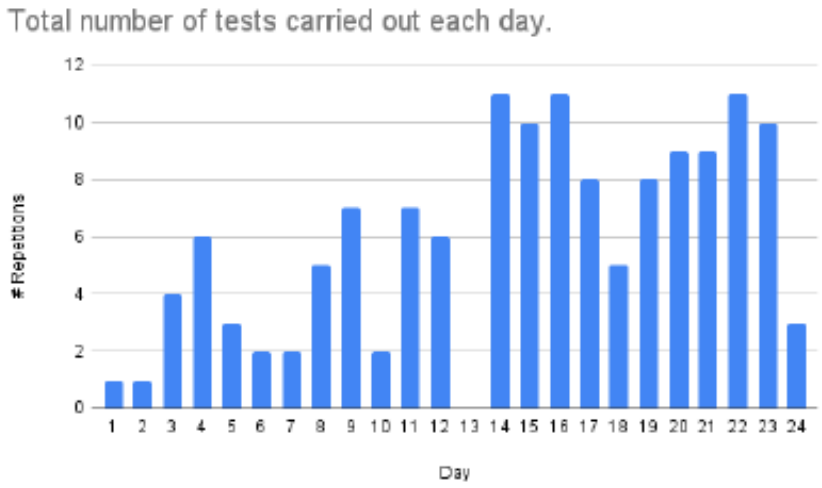


Figure 4.1: Total number of tests per day.

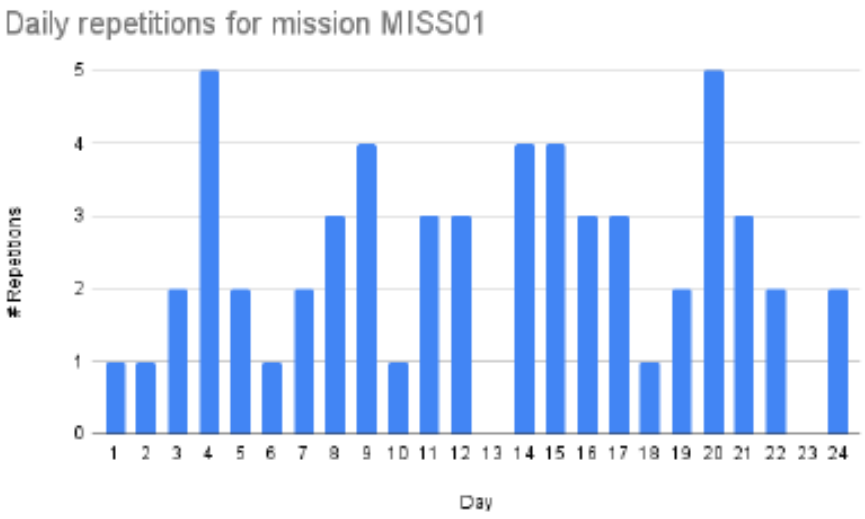


Figure 4.2: Daily mission repetitions of MISS01.

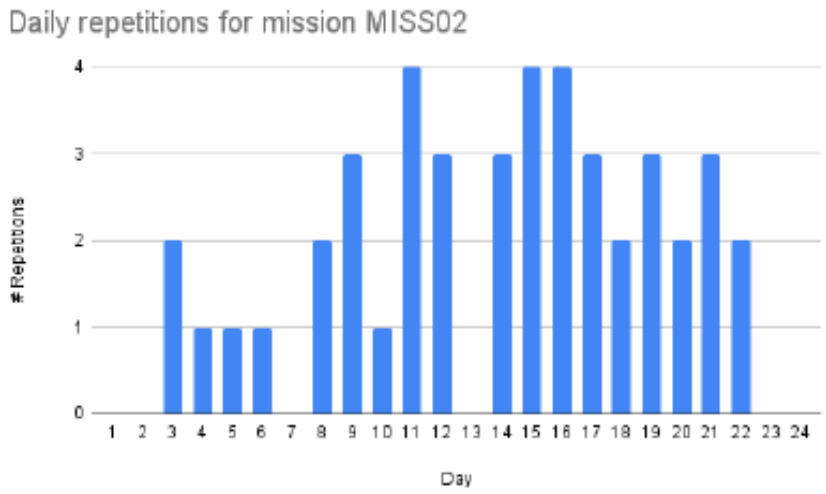


Figure 4.3: Daily mission repetitions of MISS02.

Figure 4.4: Daily mission repetitions od MISS03.

Figure 4.1 illustrates how the number of experiments conducted per day increased over time. The first weeks involved 40 iterations with an average of 3.63 iterations per day. In contrast, the last weeks comprised 101 repetitions, with an average of 8.42 iterations per day. This discrepancy arose because the first period was spent fine-tuning the general aspects and improving the computer vision algorithms using the initial images captured by Spot. Days with minimum or no iterations, excluding non-working days, were likely due to unfavorable weather conditions that prevented the robot from walking outdoors.

The majority of the last iterations concentrated on MISS03, as shown in figure 4.4, to ensure that the number of tests conducted was comparable to the other missions, while also generating significant statistics for this particular patrol route.

An Excel performance sheet was used to collect results, tracking the outcome of every individual mission repetition's component reading. Three labels were utilized to denote the success or failure of the value-reading experiment for the gauges: OK, KO, and WORK IN PROGRESS. The WORK IN PROGRESS label was used when the AI reading was not available in the performance sheet during the test period. However, this label might still appear when there was no image to process at all, possibly due to camera malfunctions or the robot intentionally skipping the callback for components that were no longer in scope. To confirm the accuracy of a reading, the autonomously read value was compared with a human reading of the same. If the difference was less than a certain tolerance, the outcome was labeled as OK. Otherwise, it is labeled KO. The experiment was considered successful when:

$$AIreading \in [0.8 * true\_reading,\ 1.2 * true\_reading]$$

We agreed on this constraint primarily for the sake of simplicity; however, an optimal solution would have been to establish distinct criteria for each type of measuring instrument.

While conducting the experiments, an accurate analysis was performed in parallel to identify the source of errors within the pipeline modules and explore potential enhancements. This was mainly done to obtain acceptable final statistics.

For example, during MISS01, a certain instrument was consistently being read inaccurately. Upon a careful analysis, it was discovered that the object detection model was unable to recognize the class of that device, leading to error. To address this issue, a distinct object detection model, focusing solely on the indicator class, was created, and implemented in place of the generic model just for the according callbacks. As a result, the autonomous reading accuracy greatly improved without the need to modify the computer vision reading algorithm. In other instances, the computer vision algorithm itself was found to be at fault, and efforts were made to rectify it.

When dealing with errors related to digital image processing, an option used was to attempt to rectify the issue retrospectively by updating the reading pipeline and re-executing it on all instances of the affected indicator class, in order to improve performance. However, it was important to also consider the potential for adverse effects on previously accurate readings, which might complicate efforts to find an optimal solution in certain cases.

Still different forms of experimental errors might arise, such those depending on robotic malfunctions or external factors, that could not be rectified retrospectively, but that could provide valuable insights to prevent similar issues in future iterations. For instance, robotic errors might include PTZ misdirection or autofocus malfunction, while external factors might involve interference with the reading caused by reflections on gauge glasses

or dirty glass. Such problems often resulted in a source image that was not readable from human eye.

The error analysis has been conducted in a systematic manner, utilizing two distinct categorization methods for classification purposes. One approach involved an in-depth internal KO classification, that served as an internal tool for improving the reading pipeline by identifying the specific module responsible for errors. The other approach, the KO type, was more general and categorizes errors based on their characteristics. This method was primarily utilized for direct communication between the team members of the test results and was assigned automatically based on the internal KO status, which was instead set manually after a human review of the algorithm execution.

The different types of KO are grouped in the table 4.2. Their brief description is following.

Table 4.2: Errors classification labels.

Internal KO status	KO type
ROBOT STOP MISSING	Robot KO
AUTOFOCUS KO	Point Engine KO
DETECTION KO	Point Engine KO
UNREADABLE KO	External KO
DARK KO	External KO
DIRTY KO	External KO
DECIMAL SEPARATOR KO	AI KO
AI READING KO	AI KO

ROBOT STOP MISSING: The callback was not executed because the robot could not reach it, due to accidental reasons, such as the presence of construction sites or closed doors, or intentional reasons, temporary exclusion from the mission due to technical issues.

AUTOFOCUS KO: The captured picture is blurry due to incorrect focusing of the PTZ on the target component.

DETECTION KO: The target component is not centered in the image because of errors in the PTZ pointing algorithm.

UNREADABLE KO: The presence of light reflections or inconsistent digits makes it difficult to read the value even with the naked eye.

DARK KO: Inadequate brightness makes it challenging or impossible to read the value by eye or with the AI algorithm.

DIRTY KO: The component is dirty and therefore not suitable for reading the value.

DECIMAL SEPARATOR KO: The AI algorithm reads the value correctly but misplaces the decimal separator. This is a common issue for digital indicators.

AI READING KO: The reading algorithm fails despite the input image being taken correctly.

ROBOT KO, POINT ENGINE KO, EXTERNAL KO, AI KO are simply macro groups of the above-described status.

Throughout the 141 repetitions of the three patrol routes, a total of 1527 images were collected to supply the automatic reading algorithms, as shown in table 4.3. Out of these, 1007 were classified as OK, indicating that the algorithms were able to accurately extract the reading based on the predefined success criteria. However, in 505 instances, the automatic reading pipeline failed due to one of the reasons outlined in table 4.2. Additionally, there were only 15 images that remained in WORK IN PROGRESS state, as the algorithms were unable to analyze them but still under investigation. Thus, the overall success rate of the proposed solution stands at 65.95%.

Table 4.3: Count and percentage breakdown of the mission results.

Outcome	Count	Percentage
WORK IN PROGRESS	15	0.98%
KO	505	33.07%
OK	1007	65.95%
Total	1527	100%

Applying various aggregations to the raw results, we were able to derive multiple insights regarding the performance of the computer vision algorithms and the behavior of the robot and its PTZ camera during the missions. In total, we obtained 1446 repetitions that involved capturing photos using the PTZ camera of points of interest during the mission. Of these, only 44 KOs were attributed to environmental conditions such as reflections on the screen, dirt on the indicator, and poor lighting. Besides the improvements in the software pointing routine of the camera and its parameters, 110 instances were still not interpretable by the algorithms, either because they required major changes in the proposed solution or because the image was not readable. In 58 cases, there was a fault in the robot. These were isolated incidents that should not affect the overall evaluation of the system's performance but could be used as an indication of adaptations that need to be made to the environment to avoid them in the future. If we exclude external or robotic/camera KOs, we are left with the actual outcome of the automatic reading algorithms. We have 227 algorithmic KOs and 1007 OKs, which represent 15.70% and 69.64% of the total, respectively, as shown in table 4.4. The remaining KOs make up 14.66% of the total. Therefore, we can conclude that both external and internal factors of the proposed computer vision-based solution had an equal impact on its feasibility. Consequently, improvements needed to be made in both directions, which may overlap sometimes.

Table 4.4: KOs count and percentage breakdown by type.

Outcome	Count	Percentage
KO - Environment	44	3.04%
KO - AI	227	15.70%
KO - Pointing/Focus	110	7.61%
KO - Robot	58	4.01%
OK	1007	69.64%
Total	1446	100%

If we analyze the results by patrol route, we can observe that MISS01 achieved an OK rate of 74%, as illustrated in table 4.5. This is more than 5% higher than the other two tours, MISS02 and MISS03, which achieved OK rates of 67% and 68%, respectively. In the overall evaluations, the first two tours carry more weight since they have a greater number of checkpoints compared to the last tour, which accounts for only about 15% of the total collected data. Additionally, the experimentation for MISS03 started late in the pilot program and time constraints prevented us from balancing the three tours.

Table 4.5: KOs count and percentage breakdown by mission.

Mission ID	OK Count	KO Count	Total	OK Percentage	KO Percentage
MISS01	446	156	602	74%	26%
MISS02	430	209	639	67%	33%
MISS03	131	62	193	68%	32%
Total	1007	427	1434	70%	30%

By examining the KOs for each patrol route as reported in table 4.6, we found that MISS01 had a high number of environmental issues, which was mainly due to its outdoor setting and difficulty reading components under unfavorable lighting conditions. It is worth noting that the other two patrol routes had almost no environmental KOs. On the other hand, MISS02 had a significant issue with pointing and focus, which accounted for most of the KOs, excluding the ones related to AI. The team conducted several tests runs to analyze the robot's position, the number of shots required, and the parameters of the PTZ camera to ensure the best possible shots. They were mainly related to ensure good performances during MISS02, that was the most challenging from this point of view.

Table 4.6: KOs count and breakdown by mission and type.

Outcome	MISS01	MISS02	MISS03	Total
Test	13	65	3	81
KO - Environment	39	5	0	44
KO - AI	76	105	46	227
KO - Pointing/Focus	30	71	9	110
KO - Robot	23	28	7	58
OK	446	430	131	1007
Total	627	704	196	1527

Sincerely, we had expected the pointing engine to perform better, but it fell short of our expectations when two components were very close on to the other, for example. We decided not to use the pointing engine in these cases and relied just on the well-configured PTZ camera parameters to capture shots that included only one component with high confidence. This kind of problem did not occur during MISS03, mostly thanks to two factors. Firstly, all the checkpoints were indoors, and secondly, the tour duration was relatively short compared to the other two missions, which allowed us to quickly make many repetitions and resolve environmental and pointing problems in a small fraction of the time.

To evaluate the automatic reading algorithms' performance, we only considered the OK and KO - AI outcomes, which had a combined total of 1234. As previously mentioned, there were 1007 OKs, resulting in a success rate of 81.6%. The mission with the highest success rate was MISS01, where we achieved an OKs rate of 85%. On the other hand, the worst performance was on the MISS03 mission, where we obtained a success rate of 74%. What said is shown in table 4.7.

Table 4.7: Automatic reading algorithms performance breakdown by mission.

Count	OKs	KOs - AI	Total	OKs Percentage
MISS01	446	76	522	85%
MISS02	430	105	535	80%
MISS03	131	46	177	74%
Total	1007	227	1234	81.6%

Examining the individual checkpoints along the patrol routes allows us to gain a more profound insight into these numbers. Tables 4.8, 4.9, 4.1010 show those values.

Table4.8: MISS01 components results breakdown.

Stop	Type	OKs	KOs Env.	KOs Robot	KOs AI	KOs Focus
01	Analog Gauge	51.8%	1.8%	0.0%	42.9%	3.6%
02	Vessel Level Indicator	73.7%	5.3%	5.3%	12.3%	3.5%
03	Round Digital Gauge	25.0%	28.6%	10.7%	10.7%	25.0%
04	Round Digital Gauge	69.1%	18.2%	0.0%	10.9%	1.8%
05	Round Digital Gauge	96.5%	0.0%	0.0%	1.8%	1.8%
06	Valve Status Indicator	90.6%	0.0%	3.8%	3.8%	1.9%
07	Round Digital Gauge	60%	14.5%	0.0%	16.4%	9.1%
08	Analog Gauge	100%	0.0%	0.0%	0.0%	0.0%
09	Analog Gauge	80.4%	1.8%	0.0%	16.1%	1.8%
10	Round Digital Gauge	73.7%	0.0%	0.0%	21.1%	5.3%
11	Valve Status Indicator	100%	0.0%	0.0%	0.0%	0.0%
Total		74.1%	6.5%	1.8%	12.6%	5.0%

Table 4.9: MISS03 components results breakdown.

Stop	Type	OKs	KOs Robot	KOs AI	KOs Focus
01	Analog Gauge	5.0%	0.0%	77.5%	17.5%
02	Round Digital Gauge	80.0%	5.0%	12.5%	2.5%
02-bis	Round Digital Gauge	69.4%	5.6%	25.0%	0.0%
04	Liquid Level Indicator	94.7%	2.6%	2.6%	0.0%
05	Rectangular Digital Display	92.3%	5.1%	0.0%	2.6%
Total		67.9%	3.6%	23.8%	4.7%

Table 4.10: MISS01 components results breakdown.

Stop	Type	OKs	KOs Env.	KOs Robot	KOs AI	KOs Focus
01	Mechanical Water Gauge	25.0%	2.3%	4.5%	45.5%	22.7%
02	Mechanical Water Gauge	17.9%	0.0%	0.0%	66.7%	15.4%
03	Mechanical Water Gauge	31.8%	6.8%	2.3%	31.8%	27.3%
06	Mechanical Water Gauge	68.2%	0.0%	2.3%	27.3%	2.3%
07	Square Digital Water Gauge	81.8%	0.0%	4.5%	4.5%	9.1%
08	Square Digital Water Gauge	88.6%	0.0%	4.5%	6.8%	0.0%
09	Square Digital Water Gauge	88.6%	0.0%	4.5%	0.0%	6.8%
10	Round Digital Water Gauge	72.7%	0.0%	2.3%	6.8%	18.2%
11	Round Digital Water Gauge	93.2%	0.0%	2.3%	2.3%	2.3%
12	Square Digital Water Gauge	84.1%	0.0%	4.5%	0.0%	11.4%
13	Square Digital Water Gauge	75.0%	0.0%	4.5%	0.0%	20.5%
14.1	Osmosis Display	84.0%	0.0%	8.0%	8.0%	0.0%
14.2	Osmosis Display	48.0%	0.0%	8.0%	36.0%	8.0%
14.3	Osmosis Display	88.0%	0.0%	8.0%	4.0%	0.0%
17	Mechanical Water Gauge	47.6%	2.4%	9.5%	21.4%	19.0%
18	Rectangular Digital Display	83.3%	0.0%	4.8%	7.1%	4.8%
Total		67.3%	0.8%	4.4%	16.4%	11.1%

The algorithms explained in detail in the previous chapter, “gray round water meter reader” and “liquid level reader”, showed in general good results. Both the algorithms belong to MISS03, that performed worse than the other two missions, probably just because of the “analog gauge” algorithm with a poor 5% of OKs. Liquid level reader scored a success rate of 94.7%, one of the highest, while the two callbacks involving gray round water meter reader scored 80% and 69.4% respectively. As previously said, those two readings were subject not only to the performance of the algorithm itself, but also on the value shown on the display. It is very probable that the statistics would have been higher if one had been sure to always photograph the correct value to be read. As a rule, algorithms that exploited pure computer vision logic have proven to be more effective than those that used an OCR tool. In fact, if the input photo was not optimal in terms of brightness, noise, or similar factors, it is highly likely that the output would have been a failure. By the way, ParseqOCR can be considered an excellent tool to use in applications such as the one under examination, and it was chosen because it has been tested and found to be superior to others. Considering that the errors for liquid level read were related to the pointing system, improvements can be made in that regard to increase the statistics. On the other hand, for gray round meter reader, it is necessary to improve the quality of the input data or adjust the algorithm to work in a wider range of brightness and noise conditions.

Chapter 5

Conclusion and further developments

To summarize, the project concluded with satisfaction and acceptable outcomes thanks to the strong performance of the robot and the majority of the computer vision algorithms. The missions were replayed 141 times, averaging 5.875 replays per day, with a total of 1527 photos captured and 69.95% successfully interpreted by the algorithms. Discarding those images that were uninterpretable due to robotic or camera malfunctions or environmental factors, 1234 photos remained, resulting in an 81.6% interpretation success rate. This pilot has demonstrated the potential of robotic autonomous systems for plant inspections, with the possibility of further improving the algorithms' performance and investigating the reason behind the failure of the Spot PTZ camera's auto-focus feature.

An important topic of debate regarding potential improvements concerns the interest in making inspections fully autonomous. Currently, Spot is assisted by an operator while walking along the designated paths, and assistance is needed to open doors. To address this issue, the team suggested different ideas including utilizing Spot Arm, the twin robot of Spot Enterprise equipped with a manipulator on its back. Although Reply does have Spot Arm, Boston Dynamics is still improving certain routines to ensure smooth operations. While the *open-door* option is available, the success rate varies depending on factors such as the type of handle and the size and location of the door. By the way, the most limiting aspect is that Spot Arm opens doors only in teleoperation mode, for now, and thus an operator still controls its action through the tablet. This makes it impossible to fully automate processes, even if Boston Dynamics has acknowledged this limitation and is working on updates to address it.

Figure 5.1 shows an example of how Spot Enterprise and Sport Arm may work together.

Figure 5.1: Spot Arm and Spot Enterprise collaborating.

In conclusion, continuing to invest in the developments of robotic technologies will lead to significant benefits in terms of accuracy and efficiency, safety, and cost-effectiveness, when comparing those new methods with respect to traditional inspections. The thesis meant to identify the key factors that are critical to successful similar implementations, and we hope that sharing the story of this project actually shed light on the aspects to be taken into account in future comparable activities.

Bibliography

[1] K. Schwab, “The fourth industrial revolution”. London, England: Portfolio Penguin, 2017.

[2] “Automating Industrial Inspection with Autonomous Mobile Robots.” https://www.energy-robotics.com/post/automating-industrial-inspection-rounds-with-autonomous-mobile-robots

[3] M. Petrova, “Where four-legged robot dogs are finding work in a tight labor market.” https://www.cnbc.com/2021/12/26/robotic-dogs-taking-on-jobs-in-security-inspection-and-public-safety-.html

[4] “Global Inspection Robots Market Size, Share & Industry Trends Analysis Report By Robot Type, By Testing Type (Non-destructive Inspection and Automated Metrology), By Application, By Regional Outlook and Forecast, 2022 - 2028,” 2022. Accessed: Apr. 03, 2023. [Online]. Available: https://www.kbvresearch.com/inspection-robots-market/

[5] D. Lattanzi, G. Miller, “Review of Robotic Infrastructure Inspection Systems,” *Journal of Infrastructure Systems*, vol. 23, no. 3, Sep. 2017.

[6] L. Yu *et al.*, “Inspection robots in oil and gas industry: A review of current solutions and future trends,” in *ICAC 2019 - 2019 25th IEEE International Conference on Automation and Computing*, Institute of Electrical and Electronics Engineers Inc., Sep. 2019.

[7] P. Biswal, P. K. Mohanty, “Development of quadruped walking robots: A review,” *Ain Shams Engineering Journal*, vol. 12, no. 2. Ain Shams University, pp. 2017–2031, Jun. 01, 2021.

[8] “Spot® - The Agile Mobile Robot | Boston Dynamics.” https://www.bostondynamics.com/products/spot

[9] “Spot Specifications.” https://support.bostondynamics.com/s/article/Robot-specifications

[10] “Spot CAM setup and usage.” https://support.bostondynamics.com/s/article/Spot-CAM-Spot-CAM-Spot-CAM-IR

[11] “Spot CORE I/O setup and usage.” https://support.bostondynamics.com/s/article/Spot-CORE-I-O-User-Guide

[12] “Operating Spot.” https://support.bostondynamics.com/s/article/Operating-Spot

[13] “Getting started with Autowalk.” https://support.bostondynamics.com/s/article/Getting-Started-with-Autowalk

[14] J. C. Russ, “The Image Processing Handbook, Fifth Edition,” *The Image Processing Handbook, Fifth Edition*, pp. 1–819, Jan. 2006.

[15] R. C. Gonzalez, R. E. Woods, “Digital image processing”, 3rd ed. 2008.

[16] C. Tomasi, R. Manduchi, “Bilateral filtering for gray and color images,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 839–846, 1998.

[17] S. Al-amri, N. Kalyankar, “Image Segmentation by Using Thershold Techniques,” 2010.

[18] N. Otsu, “Threshold Selection Method from Gray-Level Histograms.,” *IEEE Trans Syst Man Cybern*, vol. SMC-9, no. 1, pp. 62–66, 1979.

[19] “OpenCV: Morphological Transformations.” https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[20] J. Sklansky, “Finding the convex hull of a simple polygon,” 1982.

[21] R. Kundu, “YOLO Algorithm for Object Detection Explained [+Examples].” https://www.v7labs.com/blog/yolo-object-detection

[22] G. Jocher *et al.*, “ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation,” Nov. 2022.

[23] “Spot SDK — Spot 3.2.3 documentation.” https://dev.bostondynamics.com/

[24] “About - OpenCV.” https://opencv.org/about/

[25] “GitHub - heartexlabs/labelImg.” https://github.com/heartexlabs/labelImg

[26] “ROS - Robot Operating System.” https://www.ros.org/

[27] “What is Amazon EC2? - Amazon Elastic Compute Cloud.” https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

[28] “Docker overview.” https://docs.docker.com/get-started/overview/

[29] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” 2019.

[30] nathancy, “HSV color thresholder script - Stack Overflow.” https://stackoverflow.com/questions/57469394/opencv-choosing-hsv-thresholds-for-color-filtering/57469788#57469788