DTU Space
National Space Institute

Observer movement detection - ISS Docking procedure analysis

30330 - Image Analysis with microcomputers

Abstract

Docking on the International Space Station (ISS) has become much more important in the last years, allowing a safe interchange of astronauts and cargo with space stations orbiting the Earth. The importance of an efficient and precise docking procedure reflects on the safety of the astronauts on board the Spacecraft (S/C). Different ports have been used in the past and now a standard docking port is in use to allow an even easier docking. Therefore, it's of fundamental importance to be able to furnish the telemetry a visual help, recognizing the docking port from farther away by means of a camera. In addition, the space environment is quite harsh and can present atomic oxygen, different light conditions, and obstructions.

This project addresses the implementation of an algorithm to track the docking port of the Space Station while approaching it.

Course 30330 - Image Analysis with microcomputers

Project Report

December, 2023

This report has been carried out over the fall semester at the Department of Astrophysics and Atmospheric Physics, in everyday speech called DTU Space.

We would like to thank Mathias Benn for the opportunity to carry out this project and a special thank you for the guidance and help throughout the project. We would like to thank DTU Space for giving us the opportunity to access the laboratory to obtain frames closer to the real life situations.

Authors:

Giovanni Orzalesi, s223523

Alessandro Morello, s222688

Pedro Vega de Seoane, s222524

Christos Oikonomou, s220338

Contents

List of Figures

List of Tables

Glossary

HCS Hard Capture System. 4

ICC Initial Contact Conditions. 4

IDD International Definition Document. 3

IDSS International Docking System Standard. 3

ISS International Space Station. i, 1–4, 9, 30, 47

OMD Observer Movement Detection. 1

OpenCV Open Computer Vision Library. 30, 31, 37, 44

RGB Red-Green-Blue. 9, 10, 34

S/C spacecraft. i, 1, 3, 4, 6

SCS Soft Capture System. 4, 5

VS Visual Studio. 30

VSC Visual Studio Code. 30, 31

Contribution table

	Contribution table
Giovanni Orzalesi	28%
Alessandro Morello	28%
Pedro Vega de Seoane	28%
Christos Oikonomou	16%

1 Introduction

The docking procedure stands as a crucial phase of every space mission directed to the International Space Station (ISS) because it ensures the capability of the spacecraft to remain precisely bonded to the docking port of the station. This is crucial in order to ensure the safety of both human occupants and sophisticated equipment on board. Usually spacecraft used to send astronauts to space have limited autonomy and missing the docking more than once can lead to very dangerous situations for the people onboard. It is clear therefore that another location system must be run in parallel to the commonly used telemetry. This is because telemetry can be affected by delaying of the signal and an accuracy lower than the required one to dock the two spacecrafts. On the other side, the implementation of a tool to make the process faster and more precise will increase the possibility of an efficient docking. This is because the analysis of the images acquired by the approaching Spacecraft (S/C) can be easily analyzed onboard, avoiding as a consequence delays due to the transmission of the signal and errors deriving from the transmission itself. This project delves into the intricate realm of Observer Movement Detection systems, specifically tailored to study and eventually reproduce the existing protocols governing the docking operations on the ISS. Central to our investigation is the heightened importance of leveraging image analysis techniques such as edge detection and features extraction to bolster the capabilities of the State of the Art technology in place.

The current State of the Art technology in Observer Movement Detection (OMD) systems has demonstrated commendable efficacy, but there exists a compelling need for refinement to meet the evolving demands of space missions. In the hostile environment of space, variables such as fluctuating lighting conditions, occlusions, mirrors refraction, and intricate spatial dynamics present unique challenges for accurate movement detection. Recognizing these challenges, our research seeks to push the boundaries of movement detection technology, employing image analysis methodologies to try reproduce the accuracy and reliability of movement detection during the docking process. The selected approach involves a meticulous examination of existing detection algorithms, with a focus on identifying and addressing their limitations. This is necessary in order to simplify the problem, which is complex because of all the variables to be considered. The developed algorithm will be able to detect the docking port of the ISS or vice versa of the S/C and will be able to derive the relative position of the considered object with respect to the other one.

2 Problem Statement

First of all it's important to define exactly the problem that will be attempted to be solved and to set the scene that will be analysed.

2.1 Objective

This project is based on docking ring detection based on the recognition of geometrical features typical of On-orbit spacecrafts. As a consequence the project aims to build a tool to identify the docking port of a spacecraft in orbit during the docking procedure. Once identified the docking port, the program will provide the distance to it and the relative position with respect to the reference spacecraft. The designed system uses the differences in contrast between different parts of the docking port in order to recognise the features of the door and locate it in space.

Ideally, the two docking ports will be located exactly in front of the moving spacecraft perpendicular to it, matching the axis of the respective ports and slowly approaching each other. This is usually accounted for during the first phases of the approach, explained in sec.2.2.1. Moreover, in an ideal situation the light conditions are perfect in order to not saturate the CCD of the camera and to still show some features of the ring. The best case scenario will consist also of a clear line of sight, in the sense of no obstruction is in between the approaching spacecraft and the ISS. This is because as in space there is a microgravity environment, the station and the spacecraft can be rotated in different ways in order to point the solar panels towards the sun and to point all the instrument on board to perform science observations. Most likely, robotic arms, solar panels and so on can obstruct the line of sight and therefore it's important for the developed tool to be able to recognise the docking port either ways. Moreover, the tool should be able to adapt to many different situations that can happen in space. For example if the two approaching objects are not aligned and perpendicular to each other, the docking port won't look as a circle but most likely it will be closer to an ellipse and therefore the extraction of the features must take into account different geometries. On the other hand, the tool must be account also for rapid changes of the illumination of the ports. In fact, it's worth to mention that the orbital period of the International Space Station is of about 90 minutes, and as a consequence the light conditions can change during the approach. Nowadays docking ports are equipped with an illumination system to help finding the docking port in very poor light conditions. Those lights however have a limited range and struggle more when used at long distances.

The idea is then to try including all these features in an unique tool that will be able to derive distance and relative orientation of the two objects from a sequence of pictures taken during the approach. The tool will be able to adapt to different situations, allowing for very poor or very rich light conditions, small obstructions and misalignment.

2.2 System Overview

Nowadays the ISS is very popular with different world space agencies. Consequently, it is important that docking ports follow a common standard, which allows the docking of

S/C from different agencies. These docking ports must be able to function at least for one between *docking* and *berthing*. It's possible to speak about docking procedures when the spacecraft can maneuver and attach to the station by itself, while berthing refers to the use of the robotic arm or of an external tool to capture the incoming spacecraft and connect it to the station. As a consequence, a new regulation called International Docking System Standard (IDSS) has been introduced, in order to have standard dimensions and performances for all the different spacecrafts in use. This IDSS International Definition Document (IDD) details the physical geometric mating interface and design loads requirements. The physical geometric interface requirements must be strictly followed to ensure physical spacecraft mating compatibility.

2.2.1 Docking procedure

The docking procedure is a very complex maneuver to perform and usually it starts right after the launch for vessels that are intended to dock on the ISS. Figure 2.2.1 shows the different steps to accomplish in order to dock onto another satellite. The docking procedure is split in different phases, and just in the last phase image analysis tools can be implemented because of the proximity to the target, because of the light conditions and because of the required precision. The process commences with the two vehicles positioned at a remote distance, beyond each other's line of sight [1]. The spacecrafts initiates a rendezvous from two distinct initial orbits. The *chaser* possesses knowledge of the approximate orbits of both itself and the *target*. These orbits can be assumed, determined using a navigation package, or ascertained through ground tracking methods. Leveraging the orbit information, maneuvers for orbit transfer are computed to align the chaser with the target in the same orbit. The chaser is then transferred into a *drift orbit* before the next phases.[1]

After the vessel is moved to this peculiar orbit, it's possible to start the *close-range randezvous* maneuvers, divided into *homing* and *closing*. The homing phase starts when the satellite is in between 3000 to 10000km from the ISS and from this distances it is possible for the chaser to use position information communicated by a cooperative target spacecraft to navigate, or the chaser can use the target as a point of light relative to the background of stars for relative navigation. Here is where image processing tools start to play an important role in the docking procedure. The procedure now enters in the proximity operations phases. These are stratified into two distinct phases predicated on the chaser's distance from the target during execution. The initial phase, termed *closing* conventionally employs relative navigation and/or docking sensor, with the chaser positioned within the range of 1 km to approximately 100 m from the target. At this distance, correction maneuvers can be computed onboard of the spacecraft via image processing tools.[2] Eventually, these tools play a vital role in the last phase of the approach, called the *terminal randez-vous phase* where the proximity to the target is between 10 and 100 meters and therefore docking sensors and cameras are the main instruments used to provide attitude and relative position of the two vessels.

Once the approach is understood, it's important to have all the possible informations about the docking port, as it is the object that has to be detected from the chaser. The IDSS IDD ensures a preliminary phase of pre-docking rendezvous and employs a two-stage docking approach. In the rendezvous phase, an active docking vehicle navigates to the passive docking vehicle, aligning their docking interfaces for the subsequent docking stage. The passive vehicle offers three types of targets to aid the active vehicle in achieving precise alignment necessary for the mechanical interfaces to mesh at the beginning of the docking

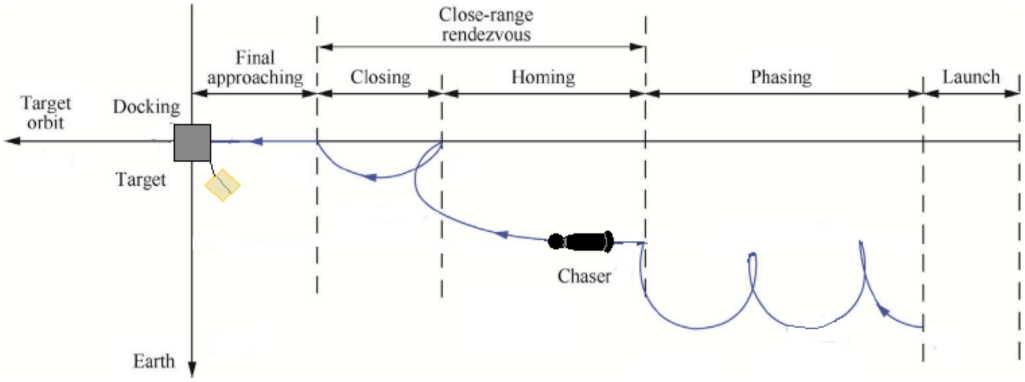
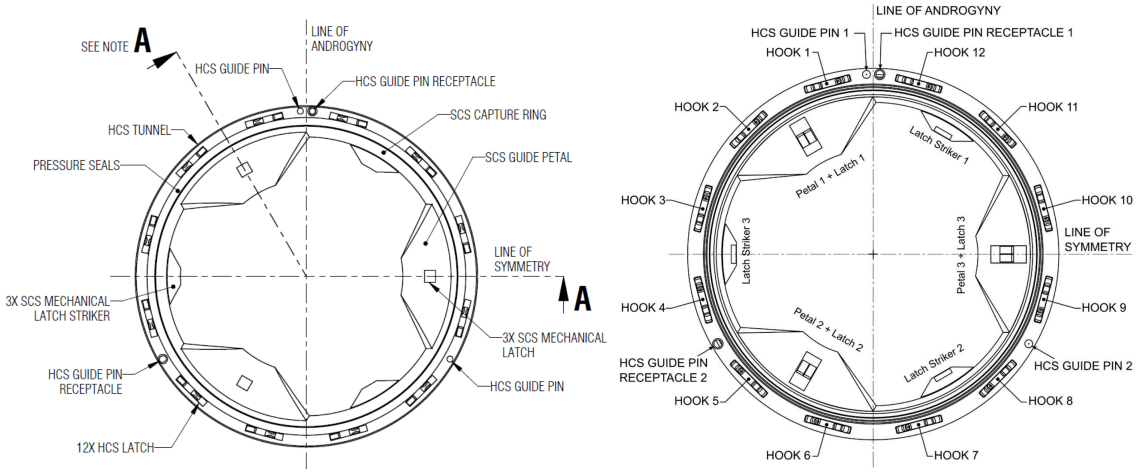


Figure 2.2.1: Scheme of the docking procedure in all its phases[2]

stage. These targets are available for operations ranging from longer to mid-range, as well as for short-range operations when the active vehicle is on the docking axis of the passive vehicle. This allows a precise localisation of the docking port even if during the rendezvouz phase the line of sight over the ISS is covered or without illumination. Furthermore, shorter-range targets are utilized by the active vehicle for alignment within the capture envelope specified by the docking system's Initial Contact Conditions (ICC) requirements, concluding the rendezvous stage. [3]

The initial phase of the terminal randezvous phase establishes the first capture of the approaching vehicles and is executed by the Soft Capture System (SCS). In this capture stage, the SCS of the active docking mechanism aligns and secures itself to the passive docking mechanism, subsequently stabilizing the newly conjoined spacecraft relative to each other. This is done in order to first secure the to spacecraft to each other, maintaining them in position to allow the Hard Capture System (HCS) to start the sealing phase. At this point in fact the HCS engages in structural latching and sealing at the docking interface, facilitating the transfer of structural loads between the spacecraft and creating a transfer tunnel. This tunnel can be pressurized to enable crew and cargo transfer for joint mission operations. These processes are strongly time dependent, therefore a fast and precise approach is required to not incur into safety issues.[3] Another particularity of the docking port is that it's fully androgynous around one axis. This means that both of the interfaces can either working as passive or active, depending on the required function in the specific occasion. As a consequence, during the docking sequence the passive soft capture interface remains retracted, while the active interface controls the soft capture function and all sequences of docking through hard capture. However usually the ISS plays the passive role, while the S/C plays the active one. This is shown in the following figure 2.2.2 where the *line of simmetry* and the *line of androginy* are shown with respect to the docking port. The figure also shows all the systems used by the SCS and the HCS.



(a) Docking port androginy interface

(b) Naming convention for hooks, guide pins, petals, latches and strikers

Figure 2.2.2: Docking port structure from International Docking System Standard [3]

The androgynous SCS interface comprises a capture ring, guide petals, mechanical latches, mechanical latch strikers, sensors, and sensor strikers. The term *striker* denotes the region on the passive side of the mating interface designed to be a contact surface for an active component on the active side of the mating interface. During the soft capture phase of docking, the guide petals are the initial components to establish contact, known as the initial contact. Subsequently, the SCS reacts to rectify lateral and angular mis-alignments between the two opposing interfaces. The soft capture is deemed complete when the two capture rings are in full contact, and the active mechanical capture latches are fully engaged with the mechanical latch strikers on the opposing vehicle.[3]

3 State of the Art technology

The need for image analysis tools arises primarily due to telemetry and attitude data requiring longer transmission times than data acquired on-board. Additionally, these data are analyzed on the ground before being re-transmitted to the spacecraft. In the case of proximity maneuvers, these operations may run the risk of taking longer than anticipated, potentially resulting in a collision between the two vehicles.[4] Considered the complexity of the problem that this project aims to solve, several studies are taken into account to implement our solution.

3.1 Base studies

Given the hostile conditions a S/C has to face in space, the recognition of the docking port is not as easy as it may looks. This issue is further exacerbated by the intricate illumination attributes in a vacuum environment, compounded by the presence of multiple layers of reflective material on the target surface that can cause a saturation of the camera. Consequently, the greyscale image is prone to noise, light spots, and the shadowing of certain discernible features on the docking ring.[4] Currently, algorithms are able to isolate the area around the target to facilitate further processes. The image is turned into greyscale form in order to emphasize high contrasts in the image, sharp edges and general features of the object. The docking port usually presents reflective features, made in order to favour a precise and fast location of it even from further distances as shown in the following figure 3.1.1

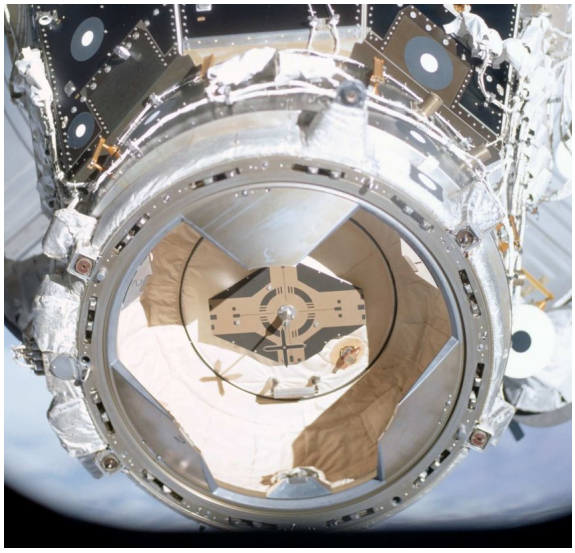
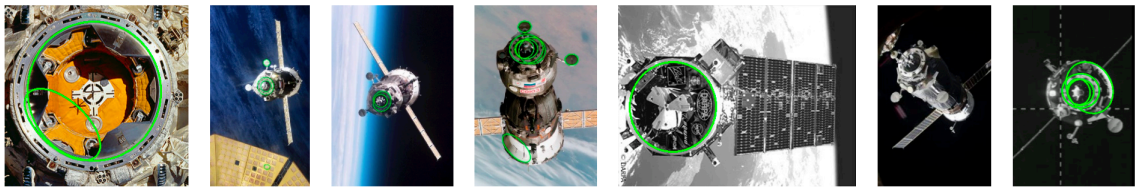


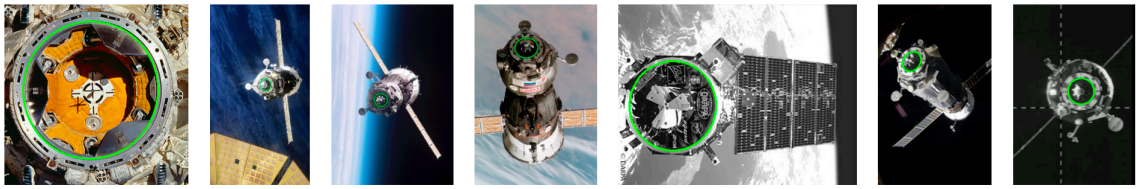
Figure 3.1.1: Reflective material used to facilitate the location of the dock from high distances

Firstly, once the image has been transposed to its greyscale version, a contour extraction is done following high contrasts in the vessel to isolate specific edges. Currently, two algorithms are commonly used: Libuda's algorithm [5] and Zhang's [6]. Those two algorithms are interesting as they have slightly different approaches to the recognition of ellipses from the points. Those two algorithms are the main inspiration to pursue the aim of the project. Libuda's algorithm is a four stage filtering process which uses geometric

features in each stage to synthesize ellipses from binary image data with the help of lines, arcs, and extended arcs. On the other side Zhang's algorithm has been specifically designed to detect geometrical shapes on on-orbit spacecrafts, thus it is very specific and therefore very fast compared to the other algorithms. The two algorithms work in slightly different ways and are presented in section 4.2.2. The result of the application of them is shown in figure 3.1.2



(a) Libuda's recognition algorithm



(b) Zhang's recognition algorithm

Figure 3.1.2: Ellipse detection with two different recognition algorithms[6]

The identified ellipses are shown in green. As it is clear, Zhang's algorithm detects just one ellipse, relative to the inner circle of the docking ring, while Libuda's algorithm detects also some false positives and false negatives ellipses. This is mainly because Zhang's algorithm has been developed to take full advantage of the characteristics of the docking ring and as a consequence the detection is faster than in other algorithms. As it is possible to see in figure 3.1.3, Zhang's method works 10 times better than Libuda's or Fornaciari's methods, and it also allows a higher precision in the recognition of the ellipses.

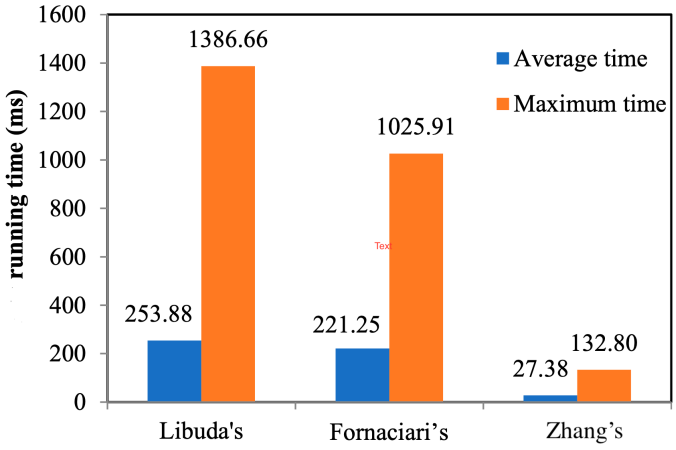


Figure 3.1.3: Time comparison for different algorithms[6]

Zhang's algorithm has been tested in different situations to test its reliability: different light conditions have been tested and the response was rapid and precise. The same

has been found for different backgrounds: the presence of the Earth in the background increases noticeably the complexity of finding contours, as the thresholding of the image is more delicate [6] [7]. The proposed algorithm however has still some difficulties in the identification of the docking port in the case where edges are not well define or when the image presents a lot of details that are difficult to split into arcs and then into ellipses. Eventually, a full sequence is presented, to give a better understanding of what is intended to do during this project. The level of precision and speed is difficult to reach for the time span of the project, but the final result will be similar to the one shown in figure 3.1.4

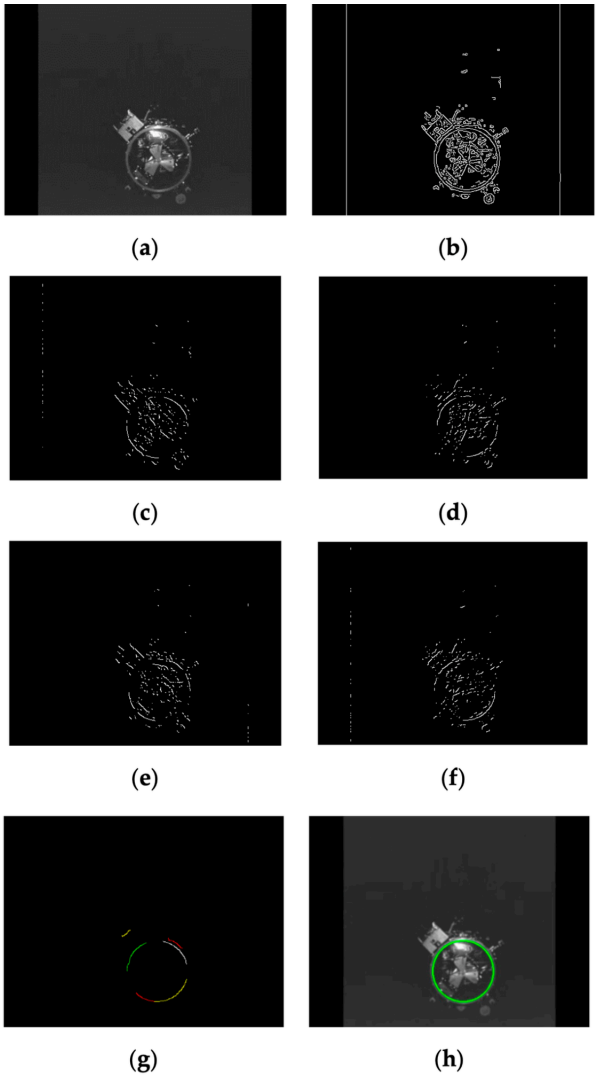


Figure 3.1.4: Final sequence of Zhang Algorithm. **a)** input image, **b)** edge image, **c)-f)** classification of edge points based on gradient orientation (see sec.4.2.2), **g)** selected arcs, **h)** detected ellipse [6]

Those methods are definitely the ones that offers the best performances in terms of detection of the geometrical features and in required run time. However they are quite tedious to implement and need a high computational capacity and memory. A more adoptable solution involves Fitzgibbon's algorithm which is based on the approximation of the ellipse by means of a Least Square solution. [8] Zhang's algorithm is addressed to future works (see sec.8.2) as its implementation is more complex that Fitzgibbon's solution.

4 Theoretical Background

This section will discuss all the theories necessary to perform the detection of the ISS's docking port.

4.1 Basics of Image Processing

4.1.1 Image Representation

An image is essentially a two-dimensional array of numbers, where each number corresponds to the intensity or color of a pixel. In digital representation, this array is organized in rows and columns, forming a matrix. Each element in the matrix represents the pixel value at a specific location in the image. The matrix dimensions correspond to the height and width of the image, and the values in the matrix determine the intensity of each pixel.

Images obtained with digital cameras or that are generated with digital methods are often in color, and different color spaces provide various ways of representing or interpreting this colors.

- **RGB Color:** In this color space, each pixel is represented by three values corresponding to the intensity of Red-Green-Blue (RGB). The combination of these three can generate a wide spectrum of colors.

Figure 4.1.1: RGB image some pixel values. [9]

- **Grayscale:** This is a special case where the image is represented in shades of gray. So each pixel constitutes a single intensity value ranging between 0 (black) and 255 (white). This space is mainly used for simplicity and computational efficiency.

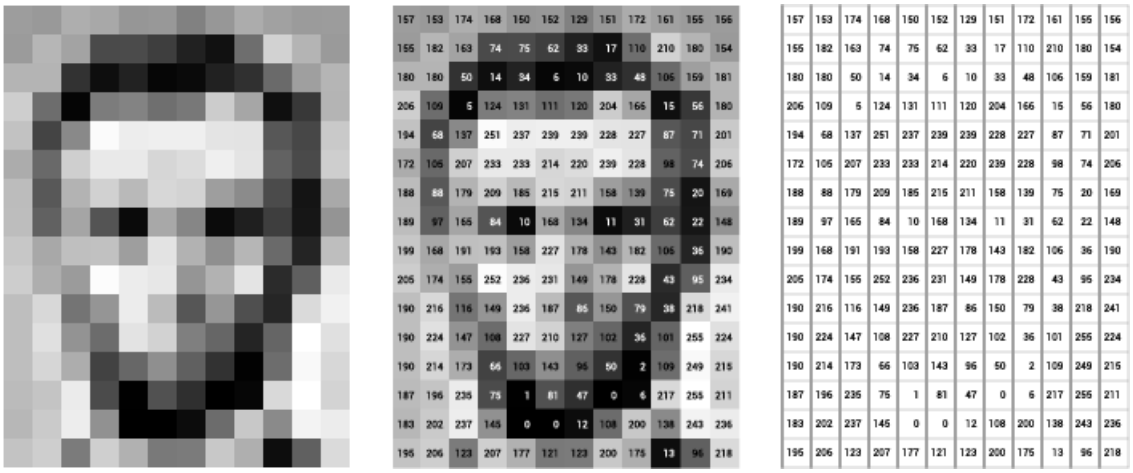


Figure 4.1.2: Grayscale image pixel values. [10]

There is an importance in understanding the significance of color spaces and what fits better depending on the problem to solve. For example, RGB is commonly used for color analysis and manipulation, while grayscale reduces the dimensionality of the image to simplify different processes.

4.1.2 Image enhancement

Image enhancement techniques are employed to enhance image quality by emphasizing specific features or eliminating undesirable characteristics. The application of these techniques is personalized for each image, tailoring the enhancement process to its unique characteristics and requirements. These are some common methods:

- **Contrast adjustment:** Contrast enhancement involves stretching or compressing the range of pixel intensities to make the image visually more appealing. This helps in distinguishing objects and details in the image.
- **Brightness correction:** Adjusting brightness involves uniformly increasing or decreasing the intensity of all pixels. This is crucial for correcting underexposed or overexposed images, ensuring better visibility of details.
- **Histogram Equalization:** Histogram equalization redistributes the intensity values across the entire range, enhancing the contrast of an image. It is particularly effective in improving the visual appearance of images with uneven lighting.

4.1.3 Noise removal

Noise in images pertains to random fluctuations in pixel values that adversely affect image quality. This interference can result from various factors, including inadequate lighting, elevated ISO settings, prolonged exposure times, and other contributing elements. The most common types are the Gaussian noise and the salt and pepper noise. As the name states, the first noise is a random variation that follows a Gaussian distribution. On the other hand, the salt and pepper noise presents itself as randomly occurring bright and dark pixels, resembling grains of salt and pepper.

(a) Noisy image: Gaussian noise with mean = 0.005 and variance = 0.005

(b) Noisy image: Salt and pepper noise with noise density = 0.003

Figure 4.1.3: Difference between Gaussian and Salt-pepper noise. [11]

In order to get rid of this noise, there are the so-called noise reduction methods. But before showing these methods, it is key to understand what a Kernel is.

In image processing, a kernel is a small matrix that is used for convolution operations. Convolution involves sliding the kernel over the image and combining the values of the pixels in the image with the corresponding values in the kernel. The resulting value is placed in the center of the kernel and forms a new pixel in the output image. So basically, considering an image I and a kernel K of size m x n. The convolution operation at a pixel (i, j) in the output image O is given by:

$$O(i,j) = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} I(i+k, j+l) \cdot K(k,l) \tag{4.1.1}$$

Where $I(i+k, j+l)$ represents the intensity value of the pixel in the image at position $(i+k, j+l)$ and $K(k,l)$ is the corresponding value in the kernel.

Once this is explained, let's show the main noise reduction methods:

- **Gaussian Blurring:** This involves convolving the image with a kernel with the following function:

  $$B(i,j) = \sum_{k,l} I(i+k, j+l) \cdot Gaussian(k,l,\sigma) \tag{4.1.2}$$

  where:

  $$Gaussian(k,l,\sigma) = \frac{1}{2\pi\sigma^2} \cdot e^{(} - \frac{k^2+l^2}{2\sigma^2}) \tag{4.1.3}$$

  being σ the standard deviation. This emphasizes nearby pixel contributions while attenuating those farther away, effectively smoothing out high-frequency noise.

- **Median Filtering:** Median filtering is a non-linear operation that replaces the intensity of each pixel with the median intensity value in its local neighborhood. This is

really useful with salt and pepper noise. Given a pixel (i, j) and a neighborhood $N(i, j)$, the median-filtered pixel is calculated as:

$$M(i, j) = \text{Median}I(x, y) : (x, y) \in N(i, j) \tag{4.1.4}$$

- **Wavelet Denoising:** Wavelet denoising involves thresholding the wavelet coefficients of an image to reduce noise. Let W be the wavelet transform of the image, and T be the thresholding function. The denoised image is obtained by applying an inverse wavelet transform:

$$D = \text{InverseWaveletTransform}\{T(W(I))\} \tag{4.1.5}$$

Where T basically sets coefficients below a certain threshold to zero and leaves others unchanged:

$$T(x, \lambda) = \text{sign}(x) \cdot \text{max}(|x| - \lambda, 0) \tag{4.1.6}$$

with λ being the threshold.

4.1.4 Image Transformation

Spatial domain transformations involve direct manipulation of pixel values in the image. These transformations are applied directly to the spatial coordinates of the pixels. There are two common spatial transformations, rotation and scaling. In the case of the rotation, there could be the angle θ, so to compute the rotation the next function should be applied:

$$R(x, y) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

where (x, y) are the original pixel coordinates, and $R(x, y)$ represents the new coordinates after rotation. And for a uniform scaling:

$$S(x, y) = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

4.1.5 Image Segmentation

This is a core aspect of image processing, involving the division of an image into distinct segments or regions based on visual features such as color, intensity, and texture. This process enhances the detailed analysis of image content, facilitating targeted interpretation and manipulation.

This section explores a range of segmentation techniques, from basic methods like thresholding to advanced algorithms such as Canny Edge Detection, active contour models, graph-based segmentation, and more. Leveraging mathematical principles, these techniques enable the identification of boundaries, edges, and clusters of pixels, contributing to a nuanced understanding of visual data.

Figure 4.1.4: Edge detection segmentation example.

For the purpose of this project, only the Canny Edge Detection will be explained. This algorithm enhances edge information, making it a powerful tool in image analysis, computer vision, and feature extraction. The mathematical steps involved ensure the accurate identification of edges in images with varying complexities. These are the main steps:

1. Gaussian Smoothing or Blur: Use a Gaussian kernel to smooth the image and reduce the noise and suppress small, high-frequency details in an image.

$$I_{\text{smooth}}(x, y) = I(x, y) * G(x, y)$$

2. Gradient Magnitude: Calculate the gradient magnitude to highlight the areas with a big intensity change. High gradient magnitude values indicate locations in the image where there is a significant change in intensity, which often corresponds to the presence of an edge.

$$|\nabla I_{\text{smooth}}| = (I_{\text{smooth}} * G_x)^2 + (I_{\text{smooth}} * G_y)^2$$

3. Non-Maximum Suppression: Retain local maxima in the gradient direction. Its primary purpose is to refine the results of gradient-based edge detection by thinning the detected edges to represent the true location of the edges more accurately.

$$I_{\text{thin}}(x, y) = \begin{cases} |\nabla I_{\text{smooth}}(x, y)|, & \text{if it is a local maximum} \\ 0, & \text{otherwise} \end{cases}$$

   This step ensures that only pixels with the highest gradient magnitudes along the gradient direction survive, contributing to a more accurate representation of edges.

4. Edge Tracking by Hysteresis: Classifying pixels as strong, weak, or non-edges and giving back a binary image (Fig. 4.1.4).

$$\text{Final Edge Image}(x, y) = \begin{cases} 1, & \text{if } I_{\text{thin}}(x, y) \geq T_{\text{high}} \text{ (strong edge)} \\ 1, & \text{if } I_{\text{thin}}(x, y) \geq T_{\text{low}} \text{ and connected to a strong edge} \\ 0, & \text{otherwise} \end{cases}$$

   where T_{high} and T_{low} are the high and low gradient thresholds.

In summary, image segmentation is a fundamental aspect of image processing, involving the division of an image into distinct segments based on visual features like color, intensity, and texture. This process enables detailed analysis and targeted interpretation of image content.

4.2 Ellipse Representation and Features

As previously noted, the docking port is characterized by a circular shape. Nevertheless, from almost all perspectives, this shape often appears more akin to an ellipse. Therefore, a more in-depth explanation of the fundamental properties of an ellipse is provided here.

4.2.1 Ellipse Theory

An ellipse is a geometric figure defined by the set of points in a plane such that the sum of the distances from two fixed points, called *foci*, to any point on the ellipse is constant. This constant sum is known as *2a*, where **a** is the *semi-major axis*. The standard form equation of an ellipse in a Cartesian coordinate system with the major axis parallel to the x-axis is given by eq.4.2.1:

$$\frac{(x - x_C)^2}{a^2} + \frac{(y - y_C)^2}{b^2} = 1 \tag{4.2.1}$$

where x_C and y_C represent the center of the ellipse, *a* is the length of the semi-major axis, and *b* is the length of the semi-minor axis. The foci of the ellipse are located at $(x_C \pm c, y_C)$, where the value *c* is the distance from the center to each focus, given by $c = \sqrt{a^2 - b^2}$.

The eccentricity (*e*) of an ellipse is a measure of how ”stretched” or ”flattened” it is and is defined as the ratio of the center-focus distance *c* and the semi-major axis *a*. The eccentricity lies in the range $0 \leq e < 1$, and an ellipse becomes a circle when the eccentricity is equal to zero. The lengths of the major and minor axes determine the overall size and shape of the ellipse. The major axis is the longest diameter, and the minor axis is the shortest diameter. The aspect ratio of the ellipse is given by the ratio of the lengths of the major and minor axes $R = \frac{a}{b}$. This will be used to filter the ellipses following expected shapes and ratios. This is because as it was mentioned in sec.2.1 the expectation is to approach the docking port with relatively small tilt with respect to it. Ellipses also exhibit symmetry about their major and minor axes. The line passing through the center and both foci is the major axis, and the line perpendicular to the major axis passing through the center is the minor axis.[12]

It can be useful for the purposes of this project to use linear algebra to represent conical shapes, this will be of particular utility in the Fitzgibbons algorithm explained later in sec.**??**. Linear algebra suggests that a conical shape can be described by **Q**:

$$Q = \begin{bmatrix} A & B & C & D & E & F \end{bmatrix}^T \tag{4.2.2}$$

The aim will be to find the coefficients (A,B,C,D,E,F) that can be used to draw a shape in the form shown in eq.4.2.3. The entire process is described in the following secitons. [13]

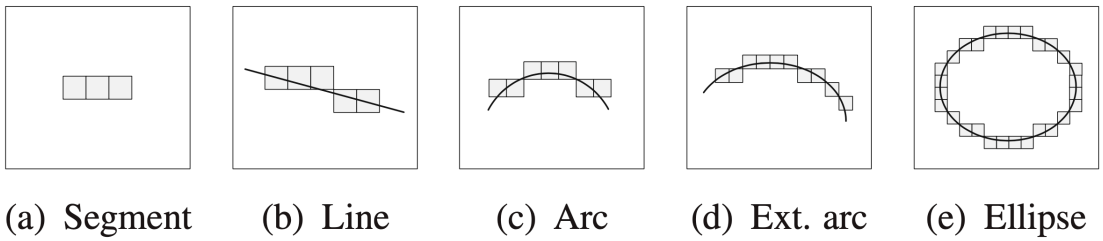
$$P(\mathbf{Q}, \mathbf{x}) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \tag{4.2.3}$$

4.2.2 Ellipse Detection

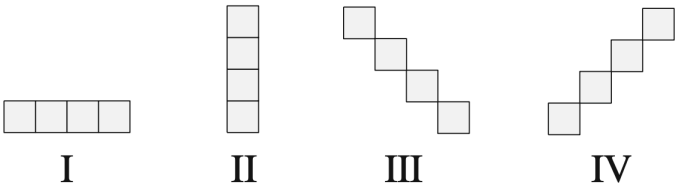
Following what has been introduced before in sec.3.1, the two algorithms studied for this project are Libuda’s algorithm [5] and the one proposed by Fitzgibbon in 1996 as it is the easier to implement on this project.

4.2.2.1 Libuda's algorithm

Libuda's algorithm for ellipse fitting is an iterative method designed to accurately determine the parameters of an ellipse that best fits a set of data points. This algorithm is particularly useful in computer vision, image processing, and pattern recognition applications where the extraction of geometric shapes, such as ellipses, is crucial. The objective is to find the ellipse parameters that minimize the error between the observed data points and the points predicted by the ellipse model. Libuda's algorithm starts with the extraction of singular segments, in order to expand the figure with time. A segment, in this context, comprises a minimum of two contiguous pixels and is categorized within one of the line orientation groups illustrated in Figure 4.2.1.



(a) Extracted objects during the detection of ellipses



(b) Group division

Figure 4.2.1: Ellipse detection steps [5]

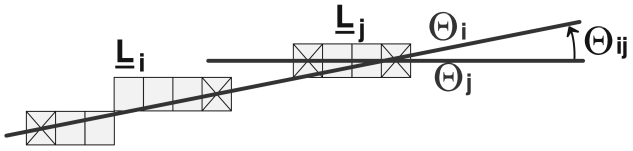
Within each distinct group of segments, lines are synthesized by connecting adjacent segments that do not surpass a predefined quantization error concerning the ideal analog line represented by these segments. The extraction of lines is done by following the algorithm proposed by *Kim et al. 2003* [14] which gives for every line orientation (figure 4.2.1 *bottom*) a set of values such as $\vec{L}_i = (x_{si}, y_{si}, x_{ei}, y_{ei}, x_{Mi}, y_{Mi}, \Theta_i)$ described by the starting position x_{si}, the end position x_{ei}, the midpoint x_{Mi} and the slope Θ_i.[5][14] Those values can be found with the following equations 4.2.4 and 4.2.5:

$$x_{Mi} = \frac{x_{si}+x_{ei}}{2} \quad y_{Mi} = \frac{x_{si}+x_{ei}}{2} \tag{4.2.4}$$

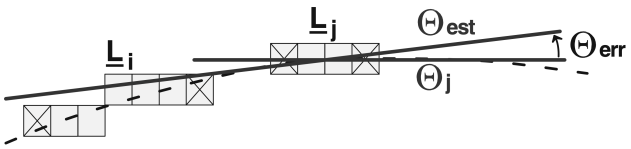
$$\Theta_i = \arctan(\frac{y_{si} - y_{ei}}{x_{ei} - y_{si}}) \tag{4.2.5}$$

Subsequently, an arc is formed by connecting at least two adjacent lines belonging to the same line orientation group (fig.4.2.1 *bottom*). These lines must adhere to a specified error limit in the tangents to an estimated circle that they collectively represent. Different lines are found via a search window that locates adjacent lines that might be part of the

same arc. A computation is then performed in order to verify that the intersection angle Θ_{ij} and the tangent error Θ_{err} are within the expected tolerances (Figure 4.2.2).[5]



(a) Intersection angle



(b) Tangential error

Figure 4.2.2: Condition to be satisfied to connect multiple lines [5]

Throughout the arc extraction process, each line orientation group is bifurcated into arc orientation groups based on the arc's orientation about the ellipse's midpoint. Those groups are defined as half of a quadrant of an ideal circumference, which results in being divided into 8 slices. Extended arcs are then composed of three consecutive arcs originating from sequential arc orientation groups. Ultimately, an ellipse is meticulously constructed by amalgamating one or more extended arcs that precisely delineate the same ellipse, adhering to a predefined tolerance. These arcs collectively cover the circumference of the described ellipse to a predetermined extent[5]

It's then possible to extract the ellipses derived from the arcs and check using some mathematical conditions that ensure the right fit of the ellipse within a defined tolerance The goal of the Libuda algorithm is in fact to find the optimal values for the ellipse parameters to minimize the difference between the given points and the approximated ones. Libuda's algorithm typically employs an iterative optimization technique, such as the Levenberg-Marquardt algorithm, to iteratively update the ellipse parameters until convergence is achieved. The Levenberg-Marquardt algorithm is particularly suitable for nonlinear least squares problems, making it well-suited for optimizing the parameters of the ellipse model, however, it is not treated in this project.

4.2.3 Fitzgibbon's algorithm

Fitzgibbon's algorithm it's a very robust and solid tool to directly fit ellipses from a set of points. It is based on solving equation 4.2.3 with some constraints in order to obtain the ellipse that approximates those points. To describe this method, let's first define $P(\mathbf{Q}, \mathbf{x}_i)$ as the *algebraic distance* of a point (x, y) to the conic $P(\mathbf{Q}, \mathbf{x}) = 0$. The fit of a general conic can be approached by the minimization of the sum of squared algebraic distances $\mathcal{D}_A$ (eq.4.2.6) of the curve to the *N* data points $\mathbf{x}_i$.[8]

$$\mathcal{D}_A = \sum_{i}^{N} P(\mathbf{x}_i)^2 \tag{4.2.6}$$

One of the constraints that can be applied to filter out some of the curves derived from

the least square solution, it's given in equation 4.2.7. Differently from the case of a *circle*, in this case the expectation is that $A \neq C$

$$B^2 - 4AC < 0 \tag{4.2.7}$$

This constraint does not facilitate the solution of the problem, as the Kuhn-Tucker conditions don't guarantee the existence of a solution.[15] The imposition of an inequality can be tedious to handle, however in this case it's possible to arbitrarily scale the parameters in order to implement an *equality quadratic* constraint instead, shown in the following eq.4.2.8.[8]

$$4AC - B^2 = 1 \tag{4.2.8}$$

It's then possible to solve the least square problem to find the coefficients of the general equation of the ellipse. From the general equation it's possible to extract the eigenvalues and the eigenvectors of the figure in order to define the key parameters of the fitted ellipse. Once the coefficients are determined, it's possible to find the determinant and the trace of the design matrix and then find the respective eigenvalues. This is done because of the relationship between the eigenvalues of an ellipse and its dimension. A simplified explanation is reported below. As it has been said before, an ellipse can be seen as a conic section of quadratic form $\mathbf{x}^T\mathbf{A}\mathbf{x} = 1$ where the matrix **A** must be a *positive definite matrix* and it's shown in eq.4.2.9

$$A = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.2.9}$$

It's then possible to factorize **A** in eigenvalues and eigenvectors such as:

$$\mathbf{x}^T \begin{bmatrix} e_1 & \cdots & e_n \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^T \\ \vdots \\ e_n^T \end{bmatrix} \mathbf{x} = 1 \tag{4.2.10}$$

that can be reconducted into the ellipse equation. As a consequence, it is possible to state that the major axis *2a* of the ellipse can be related to the *minor eigenvalue* λ_i found solving the eigenvalues problem. The relation is expressed in the following equation 4.2.11.

$$a = \frac{1}{\sqrt{\lambda_{min}}} \tag{4.2.11}$$

A similar relationship relates the minor axis *2b* to the eigenvalues of the problem but this time the correlation is with the *highest eigenvalue* instead. It's then possible to find the center of the ellipse and its orientation in space by applying the following rules: [12] [13]

$$\begin{cases} x_C = \frac{2CD-BE}{B^2-4AC} \\ y_C = \frac{2AE-BD}{B^2-4AC} \\ \theta = \frac{1}{2}\,\text{arctan}(\frac{B}{A-C}) \end{cases} \tag{4.2.12}$$

4.2.3.1 Singular Value Decomposition

As it has been stated bedore, the solution to the linear system is found by means of Least Square solution. However, a modified least square solution can be applied in order to quantify the sensitivity of a linear system to numerical error or obtain an optimal lower-rank approximation to the matrix.[16] This process is done in order to factorize the *n* x *m* kernel matrix **Q** into the product of three matrices (eq.4.2.13). Here, **U** is a *n* x *n* orthonormal matrix with columns $\mathbf{u}_i$ that span the data space, **V** is a *m* x *m* orthonormal matrix with columns $\mathbf{v}_i$ that span the model space and **S** is a *n* x *m* diagonal matrix with the singular values $\mathbf{s}_i$ in the diagonal. Note that the elements of **S** are non negative. Note also that the columns of **U** are the eigenvectors of the matrix product $\mathbf{QQ}^T$ and that the columns of **V** are the product of $\mathbf{Q}^T\mathbf{Q}$.

$$\mathbf{Q} = \mathbf{USV}^T \tag{4.2.13}$$

This decomposition is particularly interesting because it describes the relationship between the rows and the columns of the matrix, giving this method important properties in terms of dimensionality reduction and data compression. Moreover this factorization gives two orthonormal matrices **U** and **V** and therefore they can be used as new basis for the matrix. As a consequence, these two matrices can be used to perform rotations and reflections on the data.[16]

4.3 2D to 3D Point Conversion

Image analysis extracts vital information from 2D images, converting pixel coordinates to bridge the digital and physical realms. Leveraging known object sizes is crucial for accurate 3D mapping through camera calibration. This process facilitates tasks like object localization, scene reconstruction, and depth estimation, contributing to a comprehensive understanding of spatial relationships. In object localization, knowing 3D coordinates is vital for accurate positioning, tracking object movement, and determining spatial orientation.

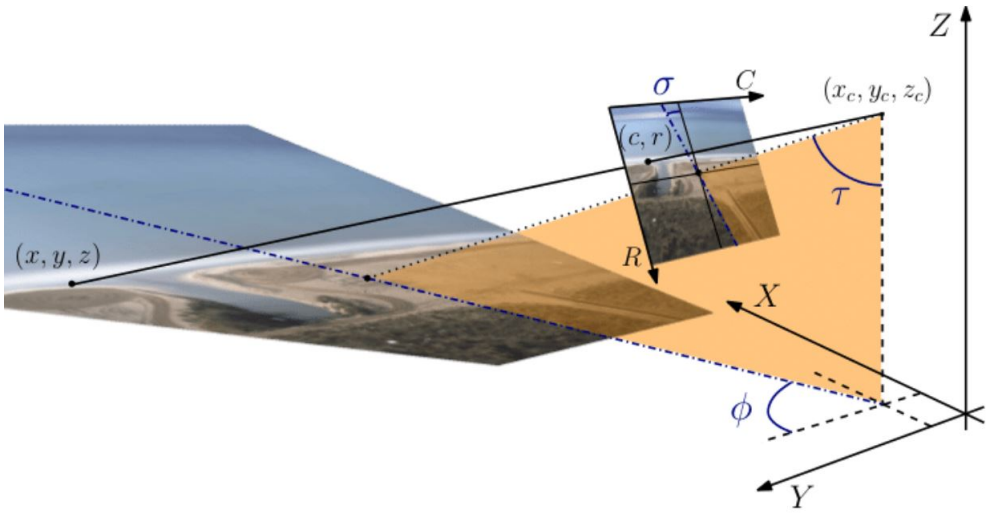


Figure 4.3.1: Pixel (c, r) to Real-world (x, y, z) transformation example. [17]

The conversion supports precise measurements, aiding in tasks like dimension estimation, volume calculation, and scene understanding. Understanding coordinate systems is key, defining a world coordinate system and a camera-centric one, seamlessly mapping pixel coordinates to real-world dimensions. The process concludes with distance calculation in real-world units, utilizing known object dimensions and the focal length. This theoretical foundation forms the bedrock for accurate 2D to 3D point conversion, deepening the comprehension of spatial aspects in digital images.

4.3.1 Pixel Allocation in Image

Pixel allocation in an image involves the systematic assignment of coordinates to individual pixels, forming the foundation for subsequent mathematical transformations and analyses. Mathematically, the image is conceptualized as a discrete grid, with each grid point corresponding to a pixel. The image resolution determines the density of this grid, influencing the level of detail captured.

$$Res = \frac{Lengthoftheimage[cm]}{Nr.pixels} \tag{4.3.1}$$

For example, and as mentioned above, in a grayscale image, each pixel is assigned an intensity value, forming a matrix of pixel values.

In the origin of the pixel coordinate system, each pixel is uniquely identified by its position (Fig. 4.3.2), denoted as (c, r), where 'c' represents the horizontal axis, and 'r' denotes the vertical axis. Whereas the image coordinate system (x,y) is usually placed in the middle of the image.

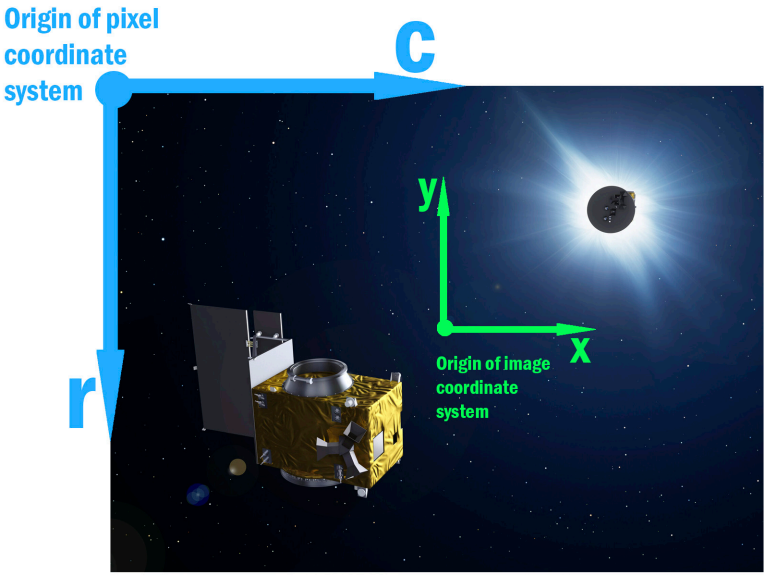


Figure 4.3.2: Pixel Coordinates vs. Image Coordinates.

To transform from the pixel coordinate system, where the origin is defined at the top-left corner (c, r), to the image coordinate system, where the origin is at the middle (x, y), you can use the following mathematical operations:

- **Horizontal Transformation:**

$$x = c - \frac{W}{2}$$

  Where W is the width of the image in pixels.

- **Vertical Transformation:**

$$y = \frac{H}{2} - r$$

  Where H is the height of the image in pixels.

These transformations center the coordinate system by adjusting the horizontal position from the left edge and the vertical position from the top edge to the middle of the image. The subtraction of $W/2$ and $H/2$ ensures that the origin is placed at the center. This process aligns the pixel coordinate system with the image coordinate system, facilitating seamless translation between the two systems for further analysis or manipulation.

4.3.2 Camera Calibration

Camera calibration is the process of determining the intrinsic and extrinsic parameters of a camera to accurately map 2D image points to 3D world coordinates. Intrinsic parameters include the focal length, principal point, and lens distortion coefficients, which are specific to the camera being calibrated. Extrinsic parameters define the camera's position and orientation in the 3D world.

The imaging process of a camera can be described using a pinhole (perspective) camera model and can be expressed as:

$$\alpha \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.3.2}$$

where (X, Y, Z)are the coordinates of a 3D point X. The projection matrix P, representing a pinhole camera model, maps the coordinates (u, v) of point X onto the image plane. The scale factor α is arbitrary. The projection matrix P is given by

$$\mathbf{P} = \mathbf{K}[\mathbf{Rt}] \tag{4.3.3}$$

K is a 3×3 upper triangular matrix known as the camera calibration matrix,R is a 3×3 rotation matrix and t is a 3×1 translation vector. R and t are called the extrinsic parameters of the camera, and they represent the rigid body transformation between the camera and the scene.

The camera calibration matrix K is given by

$$\mathbf{K} = \begin{bmatrix} f_u & \zeta & u_0 \\ 0 & f_\nu & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} af & \zeta & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3.4}$$

The intrinsic parameters of a camera, including the focal length (f), aspect ratio (a), skew (ζ), and principal point (u0, v0), play a crucial role in camera calibration. The focal length determines the distance between the camera lens and the image sensor. The aspect ratio is calculated as the ratio of the focal length in the u-direction (fu) to the focal length in the v-direction (fv). The skew parameter (ς) depends on the angle between the image axes. The principal point (u0, v0) represents the intersection of the optical axis (zc-axis) with the image plane. Camera calibration involves estimating these intrinsic parameters. In many cases, the image axes are assumed to be orthogonal, resulting in a skew value of zero. Additionally, the aspect ratio is often assumed to be one, and the skew is assumed to be zero to simplify the calibration process and improve its stability. The initial estimate of the intrinsic parameters can be improved by relaxing the constraints of unit aspect ratio and zero skew. A camera is considered calibrated when its intrinsic parameters are known. If both the intrinsic and extrinsic parameters of a camera are known, then the camera is considered fully calibrated.

Intrinsic parameters are fundamental properties of a camera that are essential for accurate camera calibration. These parameters include the focal length, principal point, and lens distortion coefficients. The focal length determines the camera's field of view and magnification, while the principal point represents the optical center of the camera's image sensor. Lens distortion coefficients account for any optical distortions that may occur, such as radial or tangential distortions. By accurately estimating and calibrating these intrinsic parameters, the camera's imaging system can be precisely characterized, leading to improved image quality, accurate measurements, and reliable computer vision applications. Proper calibration of the intrinsic parameters is crucial for achieving accurate and consistent results in various fields, including robotics, augmented reality, and 3D reconstruction.

Focal length plays a crucial role in camera calibration. It refers to the distance between the camera's lens and the image sensor when the subject is in focus. The focal length determines the field of view and magnification of the captured image. In camera calibration, accurately determining the focal length is essential for achieving precise measurements and accurate image reproduction. By calibrating the camera's focal length, distortion and other optical aberrations can be corrected, resulting in improved image quality and more reliable measurements. Proper calibration ensures that the camera accurately captures the scene as intended, making it an important aspect of achieving high-quality and accurate imaging results. [18]

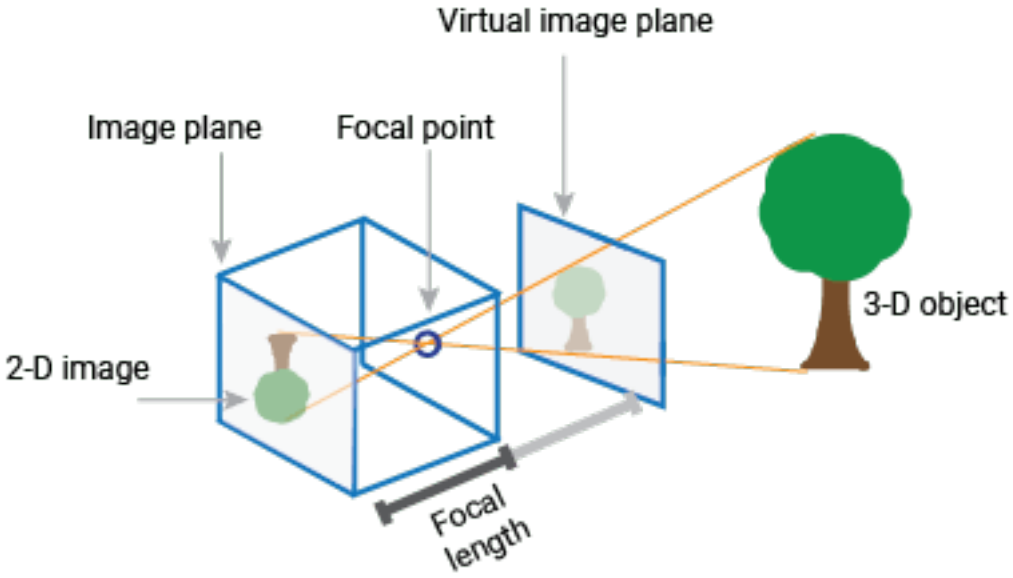


Figure 4.3.3: Inverted image on the opposite side of the camera

Distortion correction is a critical step in camera calibration to ensure accurate measurements. Distortions can occur due to various factors, such as lens imperfections or the camera's position relative to the subject. These distortions can introduce inaccuracies in the captured images, affecting the reliability of measurements and computer vision algorithms. By applying distortion correction techniques, such as radial and tangential distortion models, the distortions can be accurately estimated and corrected. This correction process involves mapping the distorted image points to their undistorted counterparts, based on the estimated distortion parameters. By removing the distortions, the resulting images have improved geometric accuracy, allowing for precise measurements and more reliable computer vision applications. Distortion correction is an essential component of camera calibration, ensuring that the camera accurately captures the scene and enabling accurate and consistent measurements in various fields, including metrology, object tracking, and 3D reconstruction.

Extrinsic parameters are an important component of camera calibration that describe the camera's position and orientation in the world coordinate system. These parameters include the rotation matrix and translation vector, which define the camera's pose relative to a reference frame. By accurately estimating and calibrating the extrinsic parameters, the camera's position and orientation can be precisely determined, allowing for accurate mapping of the captured images to the real-world coordinates. This information is crucial for various applications, such as 3D reconstruction, augmented reality, and camera track-

ing. Proper calibration of the extrinsic parameters ensures that the camera's perspective is accurately aligned with the scene, enabling reliable and accurate spatial measurements and visualizations.

Translation and rotation components are essential elements in camera calibration that describe the camera's position and orientation in three-dimensional space. The translation component represents the displacement of the camera from a reference point or origin, indicating how the camera is positioned relative to the scene being captured. It provides information about the camera's location in terms of its distance and direction from the reference point. On the other hand, the rotation component describes the orientation of the camera in space, specifying how the camera is rotated around its optical axis. It provides information about the camera's tilt, pan, and roll angles. Accurate estimation and calibration of these translation and rotation components are crucial for various computer vision applications, such as 3D reconstruction, object tracking, and augmented reality. By precisely determining the camera's position and orientation, these components enable accurate mapping of the captured images to the real-world coordinates, facilitating reliable spatial measurements and visualizations.

In camera calibration, determining the relationship between the camera and world coordinate systems is a crucial step. This relationship is established through the estimation of extrinsic parameters, which define the camera's position and orientation in the world coordinate system. By capturing images of known calibration patterns from different viewpoints, the correspondence between the image points and their corresponding real-world points can be established. Using this correspondence, the rotation matrix and translation vector can be computed, which describe the camera's pose relative to the world coordinate system. This information allows for accurate mapping of the image coordinates to the real-world coordinates, enabling precise measurements and spatial understanding. Establishing the relationship between the camera and world coordinate systems is essential for various applications, including 3D reconstruction, object tracking, and augmented reality, where accurate alignment between the virtual and real-world objects is required.

4.3.3 Pixel to Real World Transformation

In the development of this section, it was assumed that the rotation around the Z axis is neglected. This is because in this project the dimension difference is so big, that the error obtained taking into account that rotation is minimal.

With the intrinsic and extrinsic parameters from camera calibration and the conversion from the pixel coordinate system to the image coordinate system, the image-to-real-world transformation becomes more precise.

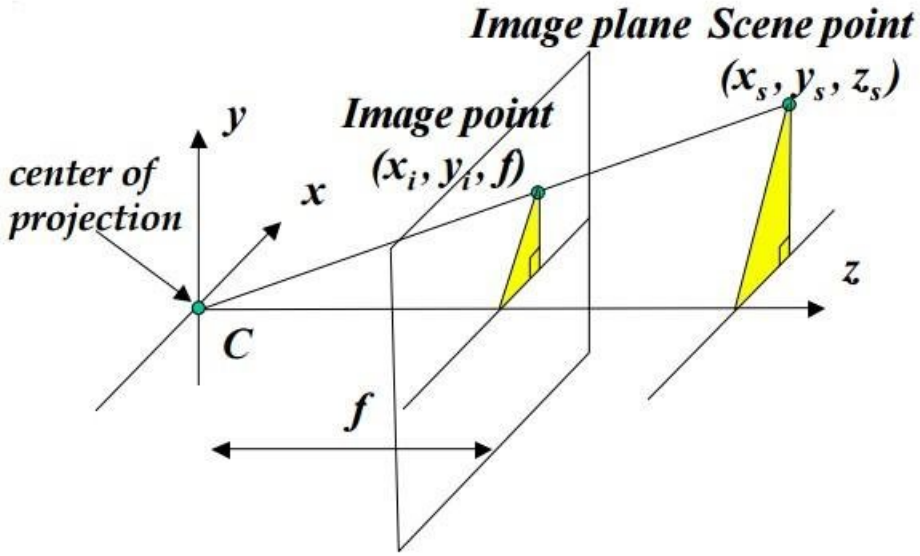


Figure 4.3.4: Projection from the image plane to real-world coordinate.

For the horizontal and vertical coordinates, the transformation is given by:

$$X = \frac{x \cdot S_x}{f_x}$$

$$Y = \frac{y \cdot S_y}{f_y}$$

Where:

- X and Y are real-world coordinates.
- x and y are coordinates based on the image coordinate system.
- S_x and S_y are the scale factors in the horizontal and vertical directions.
- f_x and f_y are the focal lengths in the horizontal and vertical directions, respectively.

These equations involve the scale factors, which are influenced by the camera calibration and the physical dimensions of the camera sensor.

On the other hand, the depth (Z) comes into play when wanting to convert pixel coordinates to 3D coordinates in the real world. In such cases, the depth information will be used to calculate the corresponding 3D coordinates.

$$Z = \frac{D_{\text{image}}}{f \cdot D_{\text{object}}}$$

Here, D_{object} is the known physical dimension of the object, and D_{image} is the apparent dimension of the object in the image.

In summary, leveraging intrinsic and extrinsic parameters from camera calibration enhances the precision of image-to-real-world transformations. The transformation equations for horizontal (X) and vertical (Y) coordinates involve scale factors (S_x and S_y) and focal lengths (f_x and f_y). These factors, influenced by camera calibration and sensor dimensions, contribute to accurate mapping. Additionally, considering depth (Z) becomes crucial for converting pixel coordinates to 3D coordinates in the real world.

4.4 Feature Matching

4.4.1 Global and Local Features

In the field of image processing and computer vision, it is necessary to represent an image using extracted features. While a raw image may contain all the information visible to the human eye, computer algorithms require a different approach.

Global feature representation involves representing the entire image with a single multi-dimensional feature vector. This vector captures various aspects of the image such as color, texture, or shape. By comparing the feature vectors of two images, similarities or differences can be identified. For example, a global descriptor of color can distinguish between images of a sea (blue) and a forest (green) by producing different feature vectors for each category. Global features can be seen as a property of the image that involves all pixels, such as color histograms, texture, edges, or specific descriptors extracted from filters applied to the image.

On the other hand, local feature representation aims to represent the image based on distinct regions that remain invariant to changes in viewpoint and illumination. These regions, known as interest regions or keypoints, are used to extract local feature descriptors that represent the image's local structures. Most local features focus on representing texture within specific image patches.

Overall, the text highlights the importance of both global and local feature representations in image processing and computer vision tasks.

Figure 4.4.1: Global and local image features representation.

The choice of features to use often depends on the specific applications at hand. Developers typically prefer features that are highly discriminative. The choice of features depends on the specific task at hand, with developers favoring discriminative features. Global features, which capture the overall pattern or characteristics of an image, are useful in scenarios such as web-scale image indexing or when a rough segmentation of the object is available. They are faster, compact, and require less memory. However, global features have limitations, as they are not invariant to significant transformations and can be affected by clutter and occlusion. On the other hand, local features, which focus on specific regions or structures within an image, offer superior performance in applications like copy detection, image matching, and object recognition. They are more distinctive and stable but require more memory due to the potentially large number of local features. Researchers propose aggregating local image descriptors into a compact vector representation and optimizing dimensionality reduction to address this issue.

4.4.2 Image Feature Detectors

Feature detectors can be categorized into three main types: single-scale detectors, multi-scale detectors, and affine invariant detectors. Single-scale detectors use a single representation for features or object contours based on the detector's internal parameters. These detectors are invariant to certain image transformations such as rotation, translation, changes in illumination, and noise. However, they are unable to handle the scaling problem. When faced with two images of the same scene that differ in scale, it is important to determine if the same interest points can be detected. Therefore, it becomes necessary to develop multi-scale detectors that can reliably extract distinctive features even when there are scale changes. The details of single-scale detectors, multi-scale detectors, and affine invariant detectors are discussed in the following sections.

Harris and Stephens introduced a combined corner and edge detector. This detector addresses the issue by calculating the variation of auto-correlation (intensity variation) across different orientations. As a result, it offers improved detection and repeatability rates. The detector based on the auto-correlation matrix is widely used in practice. The auto-correlation matrix, which is a 2x2 symmetric matrix, is utilized for detecting image features and describing their local structures and it is displayed as

$$M(x,y) = \sum_{u,v} w(u,v) * \begin{bmatrix} I_x^2(x,y) & I_xI_y(x,y) \\ I_xI_y(x,y) & I_y^2(x,y) \end{bmatrix} \tag{4.4.1}$$

The local image derivatives in the x and y directions are represented by Ix and Iy respectively. The weighting window over the area (u,v) is denoted by w(u,v). When using a circular window such as a Gaussian, the response will be isotropic, with values being weighted more heavily near the center. In order to find interest points, the eigenvalues of the matrix M are computed for each pixel. If both eigenvalues are large, it indicates the presence of a corner at that location. A diagram illustrating the classification of the detected points is shown in Figure 4. The response map is constructed by calculating the cornerness measure C(x,y) for each pixel (x,y) using

$$C(x,y) = det(M) - K(trace(M))^2 \tag{4.4.2}$$

where

$$det(M) = \lambda_1 * \lambda_2, and trace(M) = \lambda_1 + \lambda_2 \tag{4.4.3}$$

The parameter K is an adjustment parameter, and λ1 and λ2 represent the eigenvalues of the auto-correlation matrix. Computing the exact eigenvalues is computationally expensive as it involves square root calculations. To address this, Harris proposed using a corner measure that combines the two eigenvalues into a single measure. Non-maximum suppression should be applied to identify local maxima, and the remaining non-zero points in the corner map represent the detected corners.

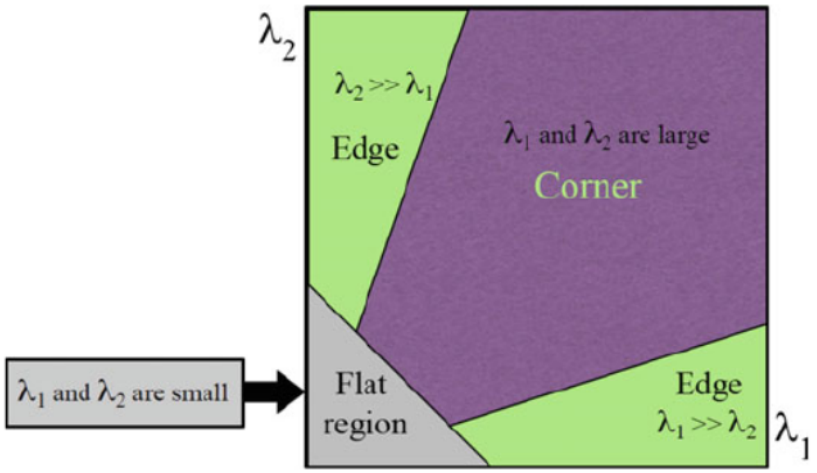


Figure 4.4.2: Classification of image points based on the eigenvalues of the autocorrelation matrix M

4.4.3 Features Matching

Image matching, which is a crucial component in various computer vision applications like image registration, camera calibration, and object recognition, involves establishing correspondences between two images of the same scene or object. A typical approach to image matching involves detecting a set of interest points in each image and associating them with image descriptors. Once the features and their descriptors have been extracted from two or more images, the next step is to establish initial feature matches between these images, as shown in Figure 14.

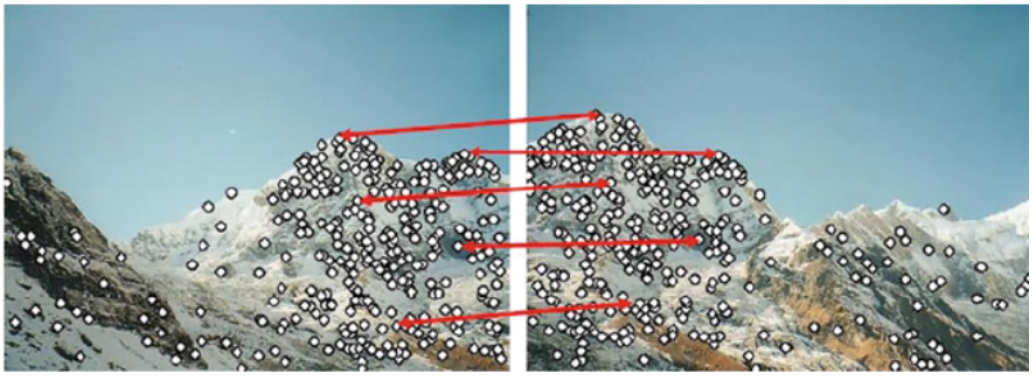


Figure 4.4.3: Matching image regions based on their local feature descriptors

In a general sense, the problem of image matching can be described as follows: Let p be a point that is detected by a detector in an image, and let it be associated with a descriptor:

$$\Phi(p) = \phi_k(P)|k = 1, 2, ..., K \tag{4.4.4}$$

where, for all K, the feature vector provided by the k-th descriptor is

$$\phi_k(P) = (f_{1p}^k, f_{2p}^k, ..., f_{n_kp}^k) \tag{4.4.5}$$

The objective is to identify the most suitable match q in another image from a set of N interest points Q=q1, q2, ..., qN by comparing the feature vector φk(p) with the feature vectors of the points in set Q. In order to achieve this, a distance measure can be defined between the descriptors φk(p) and φk(q) of the interest points, as shown below:

$$d_k(p,q) = |\phi_k(p) - \phi_k(q)| \tag{4.4.6}$$

Based on the distance dk, the points of Qare sorted in ascending order independently for each descriptor creating the sets

$$\Psi(p,Q) = \psi_k(p,Q)|k = 1,2,...,K \tag{4.4.7}$$

such that

$$\psi_k(p,Q) = (\psi_k^1, \psi_k^2, ..., \psi_k^N) \in Q|d_k(p,\psi_k^i) \leq d_k(p,\psi_k^j), \forall i > j \tag{4.4.8}$$

The acceptance of a match between two interest points (p, q) is determined by two conditions:

1. p being the best match for q compared to all other points in the first image
2. q being the best match for p compared to all other points in the second image.

It is crucial to develop an efficient algorithm to perform this matching process quickly. For matching vector-based features, the nearest-neighbor matching in the feature space of image descriptors using Euclidean norm can be utilized. However, the optimal nearest neighbor algorithm and its parameters depend on the characteristics of the dataset. Additionally, to eliminate potentially ambiguous matches, the ratio between the distances to the nearest and next nearest image descriptor must be below a certain threshold. In the case of matching high-dimensional features, the most efficient algorithms found are the randomized k-d forest and the fast library for approximate nearest neighbors (FLANN).

However, these algorithms are not appropriate for binary features such as FREAK or BRISK. Binary features are compared using the Hamming distance, which is calculated by performing a bitwise XOR operation followed by a bit count on the result. This process involves quick bit manipulation operations. When dealing with large datasets, the common approach is to replace linear search with an approximate matching algorithm. This algorithm can significantly speed up the matching process by several orders of magnitude compared to linear search. However, it comes with the trade-off that some of the nearest neighbors returned may be approximate neighbors, although they are usually close in distance to the exact neighbors.

The effectiveness of matching methods based on interest points relies on both the characteristics of the interest points themselves and the choice of associated image descriptors. Therefore, it is essential to use detectors and descriptors that are appropriate for the content of the images in various applications. For instance, when dealing with images containing bacteria cells, a blob detector should be employed instead of a corner detector. On the other hand, for aerial views of cities, a corner detector is suitable for identifying man-made structures. Additionally, selecting a detector and descriptor that can handle image degradation is crucial. For example, if there is no scale change in the image, a corner detector that does not consider scale would be highly desirable. However, if the

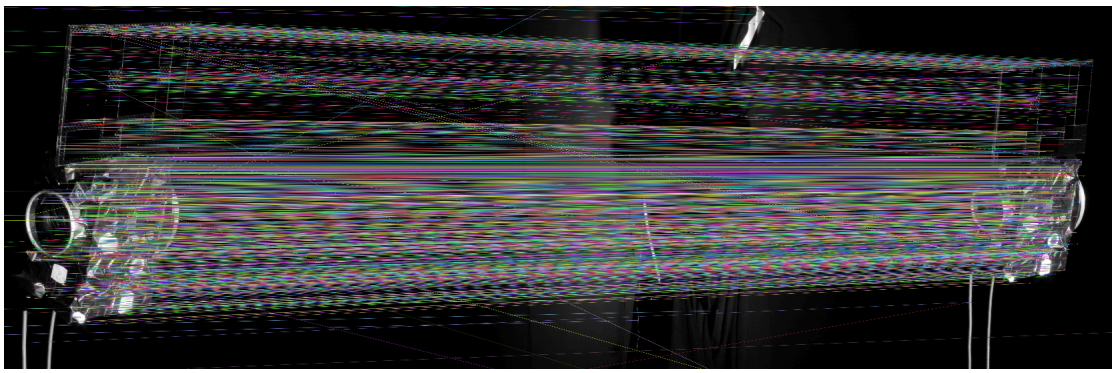


Figure 4.4.4: Feature matching applied

image exhibits significant distortion such as scale and rotation, a more computationally intensive descriptor like SURF feature detector and descriptor would be more appropriate.

To achieve higher accuracy, it is advisable to utilize multiple detectors and descriptors simultaneously. In the field of feature matching, it is important to note that binary descriptors (such as FREAK or BRISK) are generally faster and commonly used for finding point correspondences between images. However, they are less precise compared to vector-based descriptors. To filter out outliers in matched feature sets and estimate geometric transformations or fundamental matrices, statistically robust methods like RANSAC can be employed. These methods are particularly useful in feature matching for applications such as image registration and object recognition.[19]

For the project, this process resulted in being computationally heavy and not adapting to the requirements. As shown in figure 4.4.4, there are too many points detected and many are completely off.

For these reasons, this solution was abandoned and was pursued a more intuitive and faster algorithm.

5 Implementation

This section will discuss all the various tools essential for the practical implementation of the previously explained theory, aimed at detecting the docking port of the ISS. These tools are the basic structure of the operational framework.

5.1 Programming Language and Tools

5.1.1 C++ and Visual Studio

The use of an appropriate programming language holds paramount significance. Of all the possible languages that could have been used, C++ emerges as the optimal choice. Its flexibility and versatility make it suited for complex tasks such as feature extraction and other image analysis algorithms.

This language allows also the use of a robust set of libraries, giving the basic functions for the development process.

To use this language it was indispensable to download the IDE: Visual Studio (VS) or Visual Studio Code (VSC). This platform provided a good development experience not only with the coding section but also with the debugging one, where it was possible to optimize the code.

Moreover, thanks to the C++ language it was also possible to use the commonly wide library *iostream*. This library proved to be fundamental for operations such as reading from and writing to files or terminals, it also increments the overall versatility and effectiveness of the implemented solution.

5.1.2 OpenCV

Apart from the C++ language, Open Computer Vision Library (OpenCV) plays a pivotal and indispensable role in the success of this project. As the largest open-source computer vision library, OpenCV stands as a cornerstone for image processing and computer vision applications. Its extensive repository houses over 2500 algorithms, showcasing its versatility and making it an invaluable resource for researchers, developers, and engineers in the field.

OpenCV excels in both real-time and offline scenarios, making it suitable for a wide range of applications. From fundamental tasks like feature extraction through edge detection and image filtering to more advanced functionalities such as camera calibration and intricate 3D reconstruction, OpenCV offers a comprehensive toolkit. Its robust set of features empowers developers to implement complex computer vision pipelines efficiently and with high performance.

In addition to its algorithmic richness, OpenCV provides bindings for various programming languages, including C++, Python, and Java, enhancing its accessibility and integration into diverse projects. In the case of this program, it will be used in C++ as mentioned above. OpenCV's influence extends beyond traditional computer vision applications. With machine learning becoming increasingly integral to computer vision, OpenCV has embraced this shift, incorporating machine learning capabilities and deep neural networks into its framework.

In conclusion, OpenCV's impact on the project is immeasurable. Its comprehensive suite of algorithms, commitment to open-source collaboration, and adaptability to evolving tech-

nologies make it an indispensable tool for the correct development in image processing, computer vision, and beyond regarding this project.[20]

5.1.3 Matlab

In addition to C++ and OpenCV, another important tool for this project is Matlab. Developed by MathWorks, Matlab is a strong powerful language that can be combined with the wide tools that could be implemented and that offer wide versatility. Noteworthy for its visualization capabilities and its versatility and flexibility can be combined with other Software. In this particular case, Matlab serves to read the data coming from VSC providing a visual representation of the intricate analysis previously conducted using C++. It is important to highlight the synergy between these tools that can be combined to give comprehensive and insightful help to the realization of the project.

5.2 Experimental setup

The test of the tool it's fundamental to ensure the accuracy of the algorithm when it comes to real cases. To do so, a facility in DTU Space has been used. The facility houses a robotic arm which is useful as it is precise in the movements, in holding positions and it's accurate in the relative movements between different positions.

5.2.1 Image Data Collection

Of particular relevance to our project is the PROBA-3 replica setup, a component of the PROBA satellite series developed for launch by the European Space Agency. The PROBA-3 series focuses on validating novel spacecraft technologies and concepts. The intention was to use a model similar to a real one to increase the difficulty in the recognition of the features from the pictures. This increases the challenges in the extraction of the features because of the complexity of the shape of the satellite and because of its sorroundings that may show different contrasts and illumination conditions.

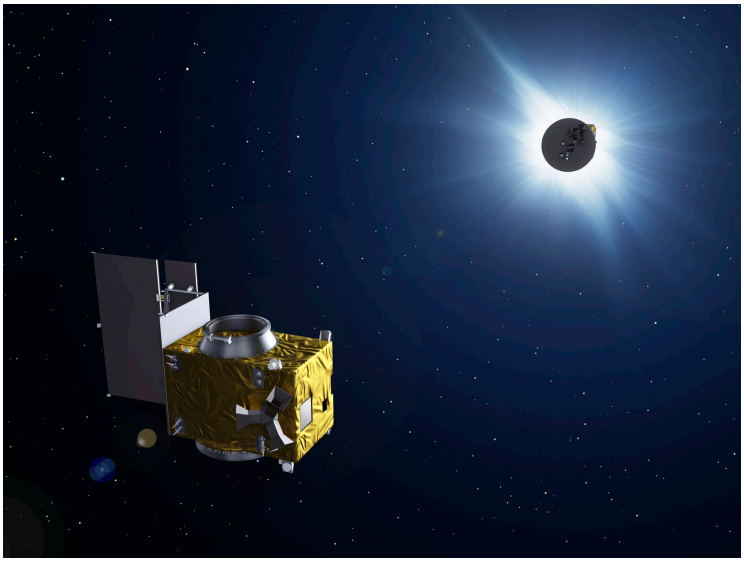


Figure 5.2.1: Proba-3 rendering representation.

While distinct from the ISS, this satellite shows a notable circular feature, similar to the docking port that is part of the ISS. This is the feature used for the purpose of this project. Additionally, the setup replicates environmental conditions akin to outer space. This controlled environment facilitated the acquisition of a multitude of images from various per-

spectives and under diverse lighting conditions, laying the foundation for subsequent processing.

5.2.1.1 DSLR Camera

A DSLR camera has been selected to acquire the images to test the algorithm. The selected camera has been a Canon EOS 2000D, with APS-C sensor and 18-55mm focal length lens (Figure 5.2.2). The F-number of the lens has been set to the highest possible in order to allow the camera to maintain the focus window wider and therefore to keep in focus the docking ring at different position. This is because usually cameras in space have fixed focus and therefore a big focus window is necessary to cover different distances.

(a) Canon EOS 2000D and Canon EFS 18-55mm lens

(b) Lens settings

Figure 5.2.2: Camera setup for the acuqisitions

Moreover the focal length of the camera has been fixed for all the acquisitions, as shown in figure 5.2.2. It's worth to mention that the use of an APS-C sensor leads to a crop factor in the image of a factor 1.5. This must be taken into account during the retrieving of the distances, as it affects the size of the features in the images.

Due to time constraints, it was only possible to be present in the lab for a single session, which posed challenges during the testing phase. Despite this limitation, a diverse range of lighting conditions and angles were deliberately incorporated into the image capture process. The setups were intentionally arranged in a random fashion to foster authenticity and ensure a representative outcome.

While the limited time in the lab presented challenges, the intentional randomness in the arrangement of setups was intended to mimic real-world scenarios more closely. This approach aimed to capture a wide spectrum of conditions that the system might encounter in practical applications, ultimately contributing to the effectiveness and adaptability of the testing phase.

The distances between the camera and the target model used to test the algorithm are reported in table 5.2.1. The recorded distance has been the one on the Z axis as it defined the major component of the distance. Because of safety reasons in fact the robotic arm couldn't be moved close to the model and therefore a fine alignment could not been tested. As a consequence the more relevant component to be analysed and therefore noted was the relative distance between the camera and the target. Furthermore, be-

Position ID	Relative distance [mm]
1	2450
2	3100
3	3750
4	4200

Table 5.2.1: Relative position of the images used for the test of the algorithm

cause of technical issues of the arm, the camera could not be pointed perpendicular to the docing ring and the perpendicular to the focal plane of the camera was tilted with respect to the satellite model of 2 degrees.

5.3 Code Architecture

The code structure is divided into two distinct files. The primary file, often denoted as the main file, encapsulates the central logic and fundamental structure of the program. Meanwhile, the functionalities and operations are implemented in a separate file named $function.cpp$. In this secondary file, it is possible to find the comprehensive definitions of all the functions utilized in the program. This modular approach enhances code organization, readability, and maintainability, allowing for a clearer distinction between the core program logic and the specific functions driving its implementation.

Towards the conclusion of the project, the algorithm underwent a transition from being applied to individual or a limited number of pictures to being systematically implemented on the entire dataset. This shift in approach is often a crucial step in validating the algorithm's robustness across diverse images.

Applying the algorithm to the entire dataset allows for a comprehensive evaluation of its performance under varied conditions, lighting scenarios, and object configurations. This application helps identify potential challenges and areas for improvement, ensuring that the algorithm is not overly tailored to specific images but demonstrates consistent effectiveness across the entire dataset.

During this phase, it's common to encounter and address issues related to outliers, variations in image quality, and unexpected anomalies. Fine-tuning parameters, adjusting thresholds, and optimizing the algorithm for a broader range of images contribute to enhancing its reliability and applicability.

5.4 Code

5.4.1 Glob

The sequential analysis of all images is a critical aspect of the testing phase. The incorporation of the *glob* function becomes imperative. The glob function is used for pattern-matching file names. It takes a pattern as input and returns a list of file names that match the specified pattern. This function, specialized in pattern-matching file names, proves indispensable in creating a list of file names that align with the specified pattern. In essence, *glob* facilitates the systematic retrieval of image filenames.

Subsequently, the next step involves coding a for loop to iterate through the generated list of image filenames. This iterative process ensures the application of the algorithm to each image sequentially. By integrating *glob* and the for loop, the code enables the systematic and efficient analysis of an entire sequence of images.

5.4.2 Reading an image

Following the iteration through the list of file names generated by the previously defined loop, a crucial step involves reading these images. This task is realized by using the *imread* function. This function is potent and not only gives a chance to load the image but also can perform the initial transformation, like loading the image in a grey scale. The loaded image is then stored in a *Mat* structure (matrix), providing a structured format that allows convenient access to individual pixels and their corresponding values. This stage of the code sets the foundation for subsequent image processing and analysis tasks.

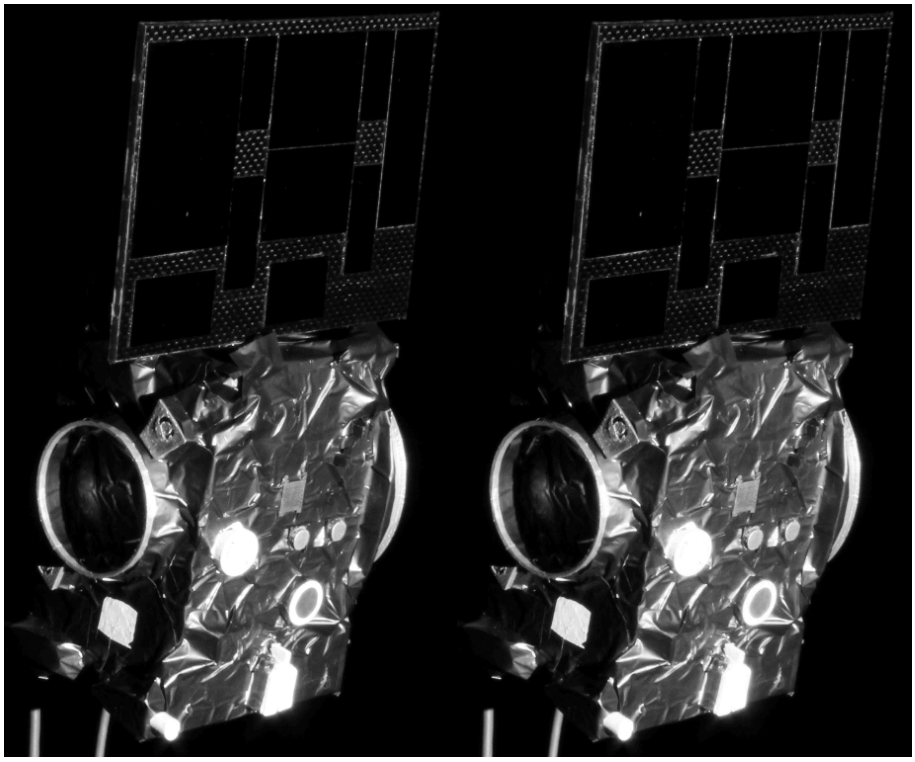
5.4.3 Grey image

Working with grayscale images in space projects is better for color ones, especially when focusing on shape or feature extraction. Gray images are particularly advantageous for their reduced computational demands. The theory behind the grey images is explained in section 4.1.1, while this chapter elucidates the practical implementation of the grayscale transformation.

After having the image file name, two primary approaches are considered. The first method implies the use of the function *imread* with the flag *IMREAD_GRAYSCALE*, allowing for a direct transformation during image loading. Alternatively, a custom function can be crafted, traversing through each pixel of the image to achieve the grayscale transformation manually.

```
1     Mat grey_image(Mat img)
```

During the iteration through the image pixels, the process involves extracting the three values from the RGB channels, and the resulting average is assigned as the final value for the corresponding pixel in the grayscale image. As illustrated in the figure 5.4.1, a visual comparison between the two approaches reveals no discernible difference in the output. Consequently, a conscious decision was made to opt for the hand-written function, given its equivalent performance and the advantage of greater control and customization in the grayscale transformation process.



(a) Built-in Grey image (b) Function Grey image

Figure 5.4.1: Difference between built-in and created function transformation

This strategic choice underscores the importance of tailoring the implementation to specific project requirements and preferences, ensuring optimal performance and adherence to the desired outcomes.

5.4.4 Gaussian Blur

Following the acquisition of the grayscale image, the subsequent task is to mitigate noise, as elucidated in Section 4.1.3. To achieve this, the initial approach involves the implementation of hand-written functions that incorporate the Gaussian kernel into the images. The process commences with the generation of a kernel using the function

```
1     Mat generateGaussianKernel(int size, double sigma)
```

where the size parameter denotes the matrix size, and the sigma signifies the standard deviation. Once the kernel is computed, the following step is to perform the convolution between the matrix and the image. This convolution process is facilitated by the function

```
1 Mat filter_2D(Mat img, Mat kernel)
```

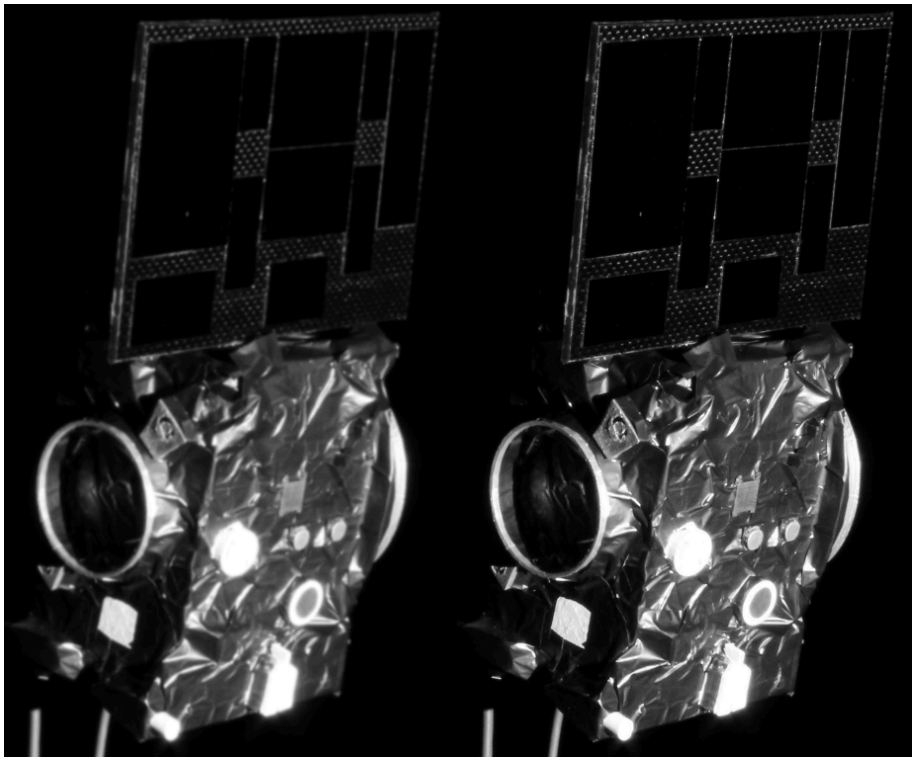
which takes the image and kernel as inputs. This function resembles the built-in function *filter2D*. This function closely resembles the built-in *filter2D* function. The output of this convolution operation yields a blurred image, effectively reducing noise and enhancing the overall image quality.

Subsequently, it became evident that the initial approach, relying on hand-written functions for Gaussian blurring, lacked the precision necessary for consistent performance across all images. The methodology initially seemed suitable for manual optimization, but then it proved to be inadequate for accommodating the varied light conditions present in the project's image dataset. The diversity in lighting across the images necessitated a more robust and adaptive blurring strategy to ensure effective noise reduction uniformly.

Recognizing these issues leads to the exploration of alternative techniques capable of addressing the challenges posed by varying light conditions in the project's image set.

The alternative solution adopts a similar concept but distinguishes itself by incorporating adaptive values that dynamically adjust for each image, providing a more responsive approach. In contrast to the previous static parameter approach, this method leverages built-in functions for enhanced adaptability. The initial step involves computing the kernel, with the size determined by calculating the standard deviation of the image using the *meanStdDev* function. Subsequently, the *GaussianBlur* function is employed to calculate the final filtered image, with parameters dynamically adapted to the specific characteristics of each image.

As depicted in Figure 5.4.2, a visual inspection unequivocally demonstrates the superior performance of the adaptive values approach. The adaptability of this method proves instrumental in achieving optimal blurring results across images with varying light conditions.



(a) Adaptive values Blur (b) Fix values Blur

Figure 5.4.2: Difference between Blur approaches

5.4.5 Canny

The subsequent project step involves performing the Canny edge detection. The theoretical foundation for this step is explained in section number 4.1.5. This phase is crucial for detecting and accentuating the borders within the images. Due to the algorithm's intricacy and time constraints, developing a custom function for Canny detection proved challenging, leading to the utilization of the built-in functions provided by OpenCV.

In the initial attempt at feature extraction, a standard Canny approach was employed. However, the diverse lighting conditions and other camera properties posed a challenge in manually configuring the parameters for each image. Consequently, an algorithm was developed to dynamically adjust the parameters, streamlining the process and enhancing the efficiency of feature extraction.

```
1     Canny(blur, canny, 50, 100);
```

In the second attempt, two additional functions,

```
1     calculateAdaptiveLowThreshold(blur);
2     calculateAdaptiveHighThreshold(blur);
```

were implemented. These functions take the blurred image as input and generate lower and higher threshold values, respectively, which are then used as parameters for the Canny edge detection function.

For the estimation of the low threshold, the algorithm employs a combination of Gaussian blur and median blur. As described in section 4.1.3, this dual-blur approach is crucial for further noise reduction. Following the application of Gaussian blur, the algorithm computes the median value and applies a constant factor to derive the final low threshold.

Similarly, in the function for estimating the higher threshold, the same dual-blur approach is utilized with a higher constant factor. This adaptive threshold technique aims to dynamically adjust the threshold values based on the characteristics of the blurred image, enhancing the robustness of the Canny edge detection process under varying conditions.

By incorporating these adaptive threshold functions, the algorithm demonstrates an ability to adapt to different images and conditions, potentially improving the detection of edges while minimizing the impact of noise. Fine-tuning these constants was necessary to achieve optimal results.

In Figure 5.4.3, it is evident that the use of adaptive threshold results in highlighting only the most significant borders. Notably, certain details, such as the points from the panels mounted on top of the spacecraft, are no longer detected. The adaptive threshold approach prioritizes the prominent features while potentially neglecting finer details in the image.

(a) Adaptive values of threshold (b) Fix values of threshold

Figure 5.4.3: Difference between Canny approaches

5.4.6 Find Contours

In the current stage of the image processing pipeline, the search for contours is a pivotal step. The accuracy and effectiveness of this process are critical for detecting the correct edges and avoiding the detection of extraneous details. The implementation involves using the findContours function, which takes the Canny edge-detected image (canny) as input. The function then populates a vector of vectors of points (vector<vector<Point> > contours) with the identified contours. Contours can be explained simply s curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.[20]

```
1     vector<vector<Point>> contours;
2     findContours(canny, contours, RETR_TREE, CHAIN_APPROX_TC89_KCOS);
```

The RETR_TREE parameter specifies the contour retrieval mode, indicating a hierarchical structure that captures contours within contours. This mode can be beneficial when dealing with complex structures or nested shapes.

The CHAIN_APPROX_TC89_KCOS parameter denotes the contour approximation method, specifically using the Teh-Chin chain approximation algorithm with the K cosines algorithm. This method helps in reducing the number of points in the contours while preserving the essential shape characteristics. It can be particularly useful for smoothing out the contours and simplifying their representation.

5.4.7 Ellipses fitting

Once the contour points are found it's possible to use them to fit the ellipse. At least 5 points are needed in order to find an ellipse, as a consequence the first thing to do is to check the vector containing the contour in order to be sure that enough points are present to fit the ellipse.

```
      if (numPoints < 5)
      {throw invalid_argument("Insufficient points for ellipse fitting");}
```

Two matrices **A** and **b** are then created to solve the Least Square Problem and a *for loop* has been implemented to associate the elements of **A** to the coefficients of the general equation of the conic (eq:4.2.3)

```
    for (int i = 0; i < numPoints; ++i) {
       double x = points[i].x;
       double y = points[i].y;

       A.at<double>(i, 0) = x * x;
       A.at<double>(i, 1) = x * y;
       A.at<double>(i, 2) = y * y;
       A.at<double>(i, 3) = x;
       A.at<double>(i, 4) = y;
       A.at<double>(i, 5) = 1;

       bTerm.at<double>(i) = -1;
    }
```

The least-square problem is then solved utilizing *Singular Value Decomposition* (see sec.4.2.3.1) and the result is given as a *rectangle* in which the ellipse is circumscribed. This has been done using the built-in *solve* function which solves the given inverse problem of finding a new parameters vector **x**. Therefore, the geometrical parameters of the ellipse need to be determined to compare the detected dimensions with the known dimensions of the dock and thus infer the distance to it. First of all, it's important to ensure that the considered matrix is *positive definite*, and therefore its determinant must be positive. Once this is done it's possible to solve the eigen-problem for the considered matrix. It's worth mentioning that all the eigenvalues associated with a positive definite matrix are positive and real and this ensures real values for the ellipse dimensions. Eventually, employing equation 4.2.11 and 4.2.12 it's possible to determine the 6 ellipse parameters necessary to describe unequivocally its shape and orientation in space.

5.4.8 Parameters Extraction

In the final stages of the project, the extraction of crucial parameters is paramount for further analysis and interpretation. The process begins by determining the diameter of the docking port, accounting for the camera's perspective effect on the elliptical appearance of the port. The major axis of the ellipse is obtained using the *max* function with the two sizes of the ellipse.

Once the major axis is calculated, the next step involves estimating the distance from the camera to the docking port using the calculateDistance function. This function takes the major axis as input, along with other fixed camera values such as focal length (*focal_length*), the pixel size in centimeters (*pix_val*), and the known real diameter of the docking port.

```
1    calculateDistance(majorAxis);
```

The distance is calculated using the formula:

```
1    distance = focal_length * realDiameter / (imgDiameter * pix_val);
```

This distance estimation leverages the camera calibration details provided in section 4.3.2.

After obtaining the distance, the focus shifts to understanding the x and y translations of the docking port. The translationCamera function is employed, taking the center coordinates of the ellipse as input. Inside this function, essential camera values such as horizontal and vertical field of view (*Hfov* and *Vfov*), image dimensions (6,000 x 4,000), and the focal length with the APS parameter are considered.

An active-pixel sensor (APS) is an image sensor where each picture element (”pixel”) has a photodetector and an active amplifier. The APS parameter is likely used to account for the size of the camera sensor.[21]

```
    translationCamera(center.x, center.y);
```

The function involves moving the reference system from the top-left corner to the center of the image, followed by calculating the final position of the docking port center. The coordinates (cx and cy) are adjusted using the following equations:

```
    cx = cx * Hfov / (2*focal_length);
    cy = cy * Vfov / (2*focal_length);
```

These calculations help establish the precise position of the docking port center within the camera’s coordinate system. The entire parameter extraction process integrates theoretical concepts outlined in various sections, including camera calibration (section 4.3.2), real-world measurements (section 4.4.3), and reference system adjustment (section 4.3.1).

5.4.9 Parameter’s file

In the conclusion of the project, saving the obtained parameters in an organized manner is crucial for documentation and future reference. The use of a *.csv file* provides an efficient way to store structured data.

```
    ofstream outputFile(pos);
```

This line of code opens a file stream (outputFile) with the specified file path (pos). The *ofstream* class in C++ is used for writing to files.

Following the file stream instantiation, the parameters can be written to the file using the output stream operator ($<<$):

```
    outputFile << translation[0] << "," << -translation[1] << "," << distance
    << "," << 6.9 << endl;
```

The files are saved in order, this means that in the first column, there is always the value of the x translation, then the y translation, the distance, and in the end the real diameter of the docking port.

After writing the required data, it’s important to close the file stream, ensuring that the file is properly closed, freeing up system resources, and finalizing the writing process.

By following this approach, the project’s parameters are effectively stored in a structured format, making it easy to analyze, share, and utilize the results in future works.

5.5 Matlab code

In this section, the MATLAB code is designed to visualize the results obtained from the previous section where critical parameters, such as x-translation, y-translation, and distance, were calculated and saved in a CSV file. The focus here is on creating informative

3D plots that illustrate the spatial relationships of the calculated points in relation to the docking port.

The initial task involves reading the parameters from the CSV file and assigning them to variables. The function *readmatrix* is employed to read the file and than the obtained values are assigned to variables.

Once the data is loaded, the scatter3 function is utilized to generate a 3D scatter plot. This function allows for the visualization of points in three-dimensional space. The x, y, and z coordinates are represented by xTranslation, yTranslation, and distance, respectively. The points are visually differentiated by color and marker style for enhanced clarity.

In order to have better understanding of the system and how the camera moved in the space, aside the 3D plot it is realized also a 2D plots between the three distances.

The results and the plots will be discussed and shown in section 6.

5.6 Camera Calibration

5.6.1 Chessboard Setup

Defining the size of the chessboard pattern in terms of inner corners is a crucial step in camera calibration using OpenCV. The chessboard pattern serves as a reference for extracting 3D object points and their corresponding 2D image points, and the accuracy of this process relies on correctly specifying the dimensions of the chessboard. The size is typically defined by the number of inner corners along each row and column of the chessboard. For instance, if a chessboard has 13x10 inner corners, it implies 12x9 squares. These inner corners play a pivotal role in the calibration algorithm, providing precise information about the spatial layout of the calibration target. Ensuring an accurate specification of the chessboard size enables the calibration process to accurately estimate the camera's intrinsic parameters and distortion coefficients, leading to more reliable and precise results in subsequent computer vision applications.

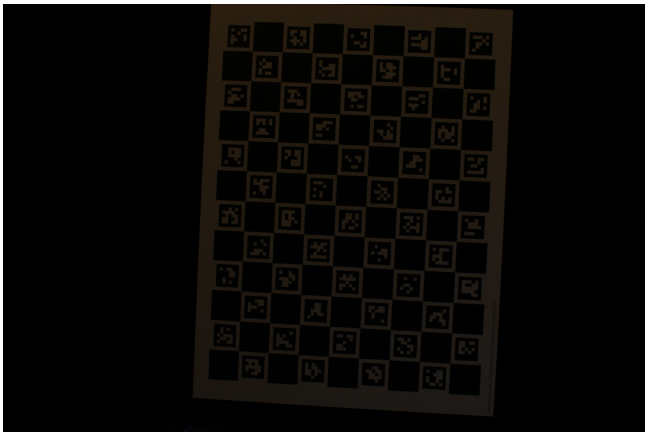


Figure 5.6.1: Chessboard

5.6.2 Object Points

In the process of camera calibration, creating a vector of 3D object points to represent the corners of a chessboard in the real world is a fundamental step. The objective is to establish a correspondence between the detected 2D image points on the chessboard pattern and their corresponding 3D coordinates in the physical world. The vector, typically named objPoints, is generated by iterating over the rows and columns of the chessboard

and assigning each corner a unique 3D coordinate. These coordinates are usually defined in a plane parallel to the chessboard's surface, with the z-coordinate set to zero. By associating these 3D object points with the detected 2D image points during the calibration process, the algorithm can accurately estimate the intrinsic parameters of the camera, such as focal length and lens distortion, facilitating precise geometric transformations in subsequent image analyses and applications.

5.6.3 Image Points and Object Points Collection

Camera calibration is a crucial step in computer vision and image processing, enabling accurate measurement and analysis of real-world scenes. The OpenCV library provides a powerful tool for camera calibration through the cv::findChessboardCorners function. This function identifies the corners of a chessboard pattern in a series of images, allowing for the extraction of 3D object points and their corresponding 2D image points. The chessboard pattern serves as a known calibration target with precisely defined corners in 3D space. Once the corners are detected in the images, the algorithm computes the transformation between the 3D world coordinates and their corresponding 2D image coordinates. The resulting calibration data is then used to correct distortion and obtain accurate camera parameters, such as intrinsic and extrinsic matrices, which are essential for various computer vision applications, including 3D reconstruction, augmented reality, and object tracking.

5.6.4 Camera Calibration

Camera calibration is a fundamental process in computer vision, and OpenCV provides a robust solution with the cv::calibrateCamera function. This function takes as input a set of 3D object points and their corresponding 2D image points, acquired through techniques like the cv::findChessboardCorners. The goal is to estimate the intrinsic parameters of the camera, represented by the camera matrix (cameraMatrix), distortion coefficients (distortionCoefficients), rotation vectors (rvecs), and translation vectors (tvecs). The camera matrix encapsulates essential information about the camera's focal length and principal point, while distortion coefficients account for lens imperfections. The rotation and translation vectors define the camera's pose in the 3D world. The calibration process optimizes these parameters to minimize the difference between the observed image points and the projected 3D points. Accurate calibration facilitates precise geometric transformations, enabling applications such as image rectification, object measurement, and 3D reconstruction.

5.6.5 Undistorted Image

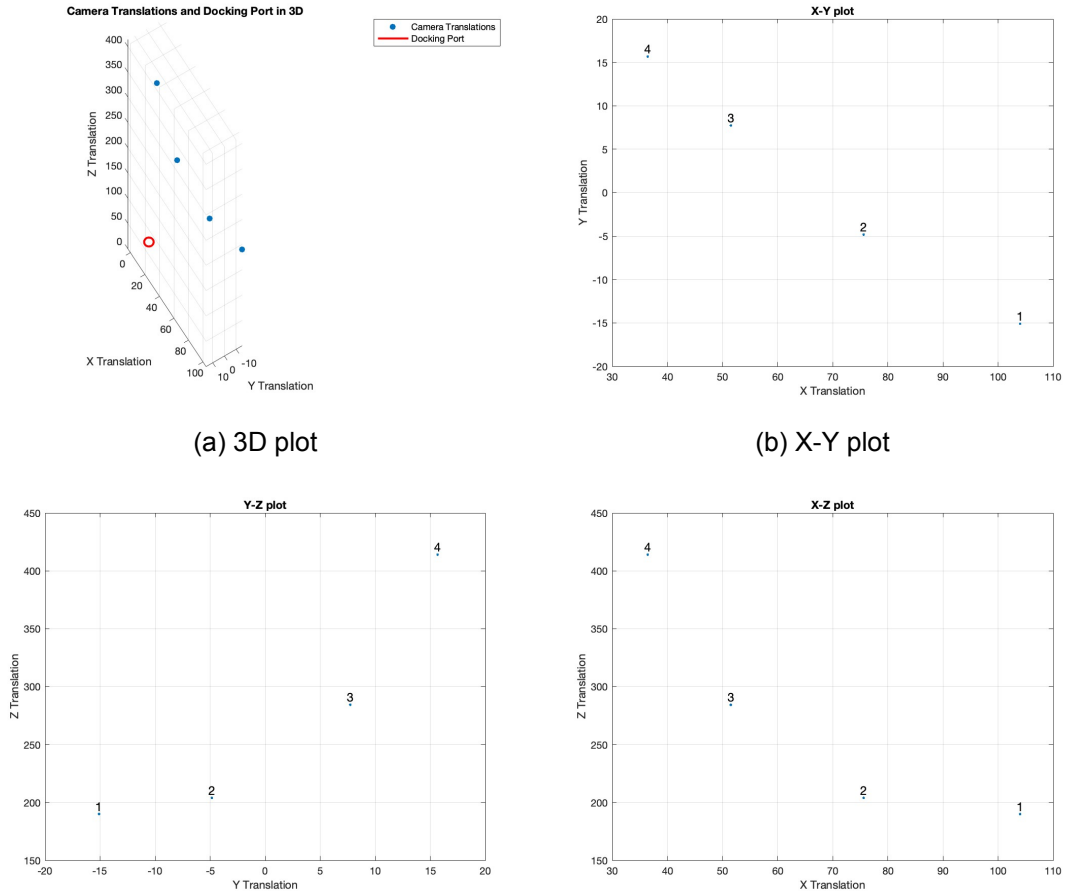
Once camera calibration parameters, including the camera matrix and distortion coefficients, are obtained using methods like cv::calibrateCamera, the cv::undistort function in OpenCV proves invaluable for rectifying images and correcting lens distortions. This function utilizes the calibration parameters to transform distorted images into undistorted representations. By providing the original image along with the camera matrix and distortion coefficients as input, cv::undistort performs geometric corrections, removing radial and tangential distortions caused by the camera lens. The result is a visually improved image with corrected straight lines and more accurate geometric properties. This undistorted image is crucial for subsequent computer vision tasks, ensuring that measurements, object recognition, and image analysis are performed on accurate representations of the scene, ultimately enhancing the reliability and precision of computer vision applications.

6 Results and Analysis

In this chapter, it will be presented the conclusive plots encapsulating the outcomes of the image processing. The results are showcased through visual representations, depicted in both MATLAB plots and modified images facilitated by OpenCV.

The MATLAB plots serve as insightful visualizations of key metrics, providing a comprehensive overview of the algorithm's performance.

This comprehensive presentation of results not only validates the effectiveness of the adaptive blurring approach but also serves as a foundation for further insights and potential optimizations in the broader context of space-based image processing applications.



(a) 3D plot

(b) X-Y plot

(c) Y-Z plot

(d) X-Z plot

Figure 6.0.1: Results

From a static 3D figure could be hard to understand all the features. However, from figure 6.0.1 subfigure (a), the docking port is highlighted in red, serving as a reference point amidst a constellation of blue dots representing distinct camera positions in space. This visualization offers a view of the spatial relationships and orientations.

In subfigure (b) of figure 6.0.1 shows in details the camera positions, particularly in the

X-Y plane (width-height). This perspective enhances the comprehension of the camera's movement, illustrating a trajectory from right to left and simultaneously from bottom to top. Each point is meticulously numbered, a crucial step to establish a clear association between each point and its corresponding image. This numbering system is instrumental in preventing confusion and maintaining accuracy throughout the analysis.

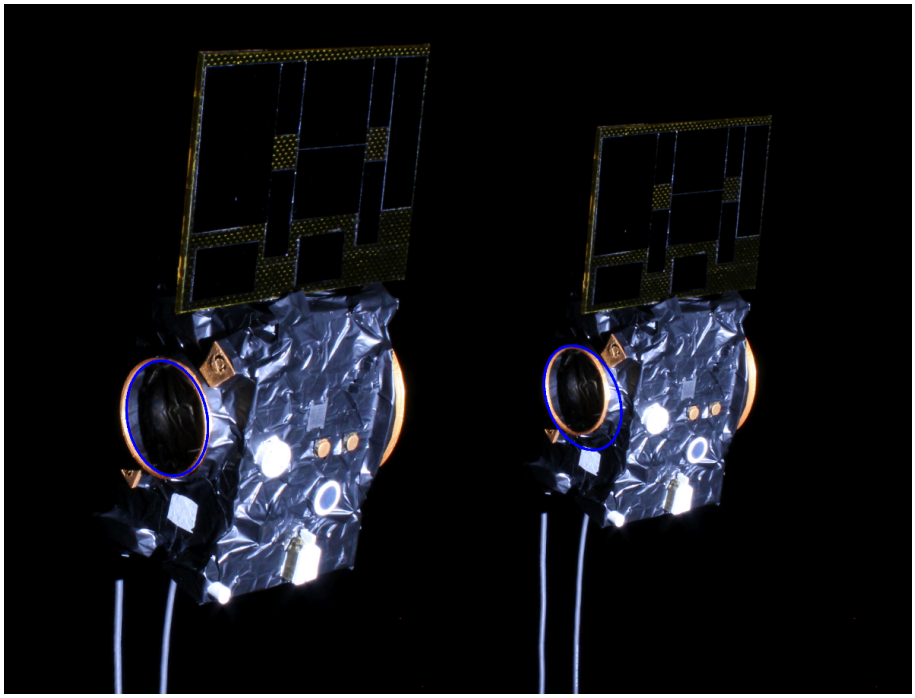
Subfigure (c) provides a side view, depicting the movement on the height-distance plane. The observed trajectory shows a movement from left to right, indicative of an increase in height as the camera moves further away from the docking port.

In subfigure (d), the movement is showcased in the width-distance plane, providing a perspective from the top. As the camera recedes, the trajectory unfolds from right to left.

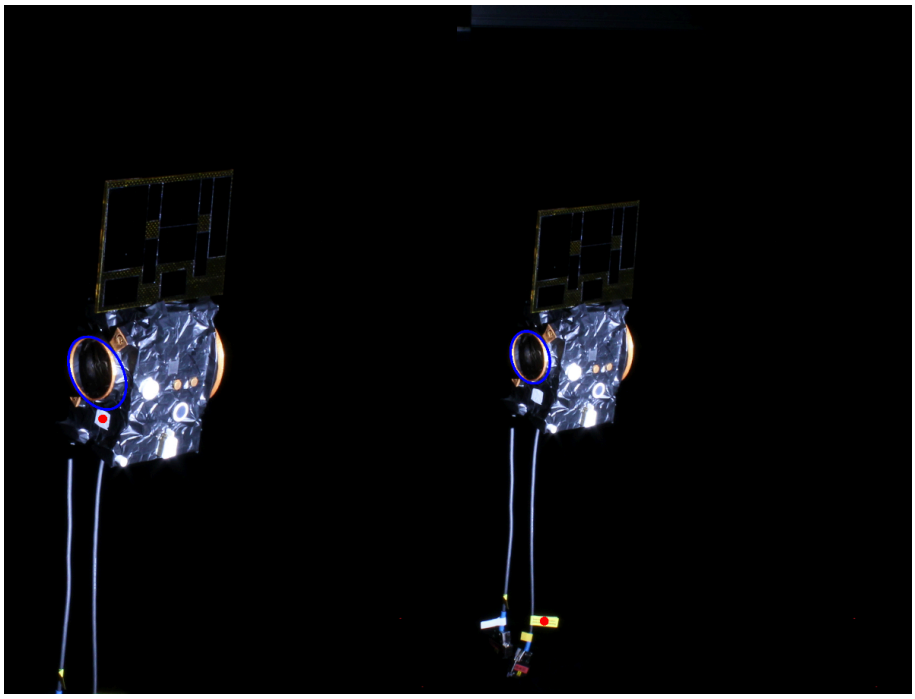
Figure 6.0.2 demonstrates the algorithm's proficiency in consistently detecting the docking port across each frame. The inclusion of a blue ellipse superimposed on the original frame effectively showcases the accuracy of this detection process, affirming the algorithm's reliability in identifying the port's location. Despite it's accuracy in the detection of the ellipses in the figure, the algorithm still lacks of precision in the fitting of the ellipse. A possible reason of that can may be found in the selection of the points used to fit the ellipse. A selection of more spread out points would have led to a more accurate constraint to fit the correct ellipse to the image. A perfect fit in fact is obtained only in the first figure of the set, while the second and the third are too elongated to match the ellipse in the figure. The last figure present an accurate fit, even if not exact.

Furthermore, in the third frame of Figure 6.0.2, a red dot is visible on the white square, positioned alongside the docking port. This feature was intended to facilitate the detection of the camera's rotations, as detailed in Section 8.1. Unfortunately, due to time constraints, further refinement of this aspect was not feasible. The algorithm's capability to detect the white square alongside the docking port was limited to just one frame, indicating a need for refinement in detecting rotational movements. Despite this limitation, the algorithm's adeptness in consistently identifying the docking port stands as a testament to its overall functionality.

The comparison of the figures in 6.0.2 (a) to (d) with the plot results in 6.0.1 provides valuable insights into the spatial dynamics of the camera. The progression from the first frame to the last clearly illustrates the camera's consistent movement further away from the docking port, characterized by a right-upward trajectory. These observations affirm the accuracy and reliability of the algorithm's ability to discern the camera movements throughout the sequence.



(a) First frame (b) Second frame



(c) Third frame (d) Forth frame

Figure 6.0.2: Frame results

6.1 Results discussion

To draw a comprehensive conclusion for the project, it is imperative to assess the accuracy with which the algorithm detects these camera movements relative to the docking port. Precision in this aspect is crucial for ensuring the reliability of the entire image processing pipeline and its applicability to space projects. Further analysis and evaluation metrics

Frame ID	True relative distance [mm]	Detected relative distance [mm]
1	2450	1899
2	3100	2039
3	3750	2842
4	4200	4140

Table 6.1.1: Results of the distances detected with the algorithm developed during the project

are essential to quantify and validate the accuracy of the algorithm's output in capturing the nuanced movements in space.

The data acquired in the DTU Space facility are used to test the proficiency of the algorithm developed for this project. The results are reported in table 6.1.1 with a comparison with the real measurements and are discussed below.

The data presented in Table 6.1.1 outlines the input values for the robotic arm on the left and the corresponding analysis-derived values on the right. It's crucial to note that the distances measured for the robot are relative to its origin, whereas the analysis provides distances from the ISS. The calculation of accuracy error becomes paramount in assessing the precision of the robotic arm movements.

The accuracy error is determined by comparing the actual distance covered by the robot with the distance calculated by the algorithm. A negative accuracy indicates an overestimation, implying that the robot is farther to the ISS than predicted. Conversely, a positive accuracy implies an underestimation, suggesting that the robot is closer than calculated.

Examining the first movement, there is a notable difference between the actual distance of $650mm$ and the algorithm's calculated distance of $140mm$, resulting in a significant negative accuracy of $-510mm$. In the second movement, despite the spacecraft covering the same distance, the algorithm portrayed it as further away, resulting in a positive accuracy of $+153mm$.

In the final movement section, with a reduced movement of $450mm$, the algorithm recorded a substantial movement of $1298mm$, yielding a positive accuracy of $848mm$. While these accuracy values may initially seem considerable, it is crucial to emphasize that these differences are measured in millimeters.

Upon a closer examination of the data and the accuracy analysis, it becomes evident that the absolute differences in movements are relatively small. The fact that these accuracies are in millimeters suggests that, despite variations, the algorithm performance is reasonably accurate. Therefore, while the numerical differences may appear significant, in the context of millimeter-scale movements, the variations are not as substantial as they might initially seem.

6.2 Challenges Faced

The project proved to be a fascinating and intellectually stimulating endeavor, although it presented several formidable challenges that the group had to navigate. One primary challenge was due to the limited dataset available for experimentation. A more expansive dataset would have offered a greater opportunity to refine and rigorously test the accuracy of the algorithm. The constraints imposed by the small dataset underscored the importance of robust testing environments for comprehensive evaluations. Further-

more, it could have been useful save the movements information to check the accuracy. However, in the only one lab session it was stored just the relative movement of distance.

Another significant challenge was the intricacy of code implementations. While OpenCV provides a vast array of functions, the complexity of certain algorithms, coupled with the computational demands, made manual implementations arduous. Balancing the trade-off between manual coding for customization and utilizing built-in functions for efficiency posed a notable coding challenge.

In conclusion, the project was inherently complex, exacerbated by the scarcity of comprehensive literature available for reference. Few researchers from other universities had explored related approaches, but their implementations often lacked essential details, making it challenging to derive valuable insights. Moreover, the absence of articles from space agencies posed a significant obstacle, particularly in the development of effective and reliable ideas. Despite these challenges, the project offered valuable learning experiences and underscored the need for collaborative efforts and knowledge sharing in the realm of space-based image processing.

7 Conclusion

In this comprehensive report, we have delved into the development of the ”Observer Movement Detection - ISS Docking Procedure Analysis” project. Extensive research has led to successful testing of many aspects, while others remain in the realm of theory.

The project’s approach involved initially crafting a code tested on a single image and subsequently adapting it for multiple images. A feature-matching approach was initially pursued but proved ineffective due to the lack of noticeable features on the docking port. Instead, an adaptive code was employed, adjusting functions based on the unique characteristics of each image.

To achieve success in the whole project, a variety of methods and tests were employed. Notably, bespoke functions were crafted for most algorithms. While these were rigorously tested alongside existing OpenCV functions, it was observed that the custom functions, although yielding nearly identical results, proved less computationally efficient than their built-in counterparts. Consequently, during testing, the project predominantly relied on the efficiency of OpenCV’s native functions.

Several methods and algorithms were explored but not ultimately utilized. First, attempts to implement a Gaussian Blur function, where the Gaussian kernel convolved the grayscale image, proved unsuccessful as it couldn’t match the precision achieved by OpenCV’s built-in Gaussian Blur. Similarly, the Ridge Detection algorithm for contour detection, while implemented, was discarded due to its computational intensity and the availability of a superior solution, such as the actual one. Moreover, an attempt was made to implement an algorithm for the rotation of the docking port. This required detecting a significant feature on its surface to exactly know the orientation of it. Despite various attempts, time constraints prevented successful implementation.

I regard the functional code works as expected, processing images one by one and dynamically adjusting each function according to the image’s characteristics. Upon detecting the docking port and calculating its central point, the 3D coordinates are saved for each image in relation to the camera. The concluding step involves displacing the camera’s position in a 3D space, offering insights into its movements concerning the docking port.

While the current implementation demonstrates robust performance for different camera arrangements, it has only been tested on a limited set of images. Future work should involve thorough testing with a larger dataset encompassing diverse camera perspectives. Additionally, implementing rotation functionality for the camera will further broaden its applicability. Accuracy testing should be performed to ensure the reliability of the results across various scenarios and conditions.

8 Future Work

As for the future development of the project, a deeper insight in many areas is needed before a real size prototype of the system is made. Many conclusions have been met about aspects of the project that should be improved. Many of the improvements and constraints have already been mentioned in the pages of the present report.

8.1 Rotation detection

Understanding the rotation of the docking port is crucial for its accurate placement in 3D space. Upon detecting the ellipse and its central point, two possible normal vectors of the plane formed by the ellipse can be identified, as illustrated in Figure 8.1.1.

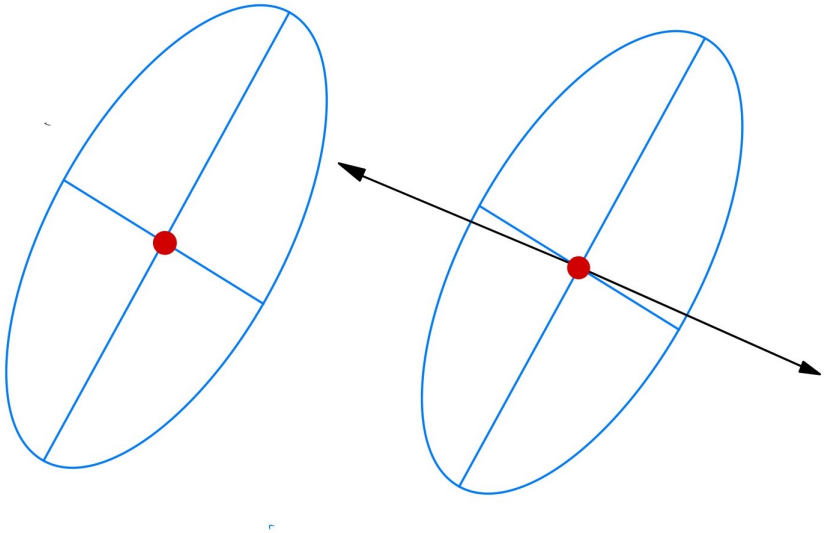


Figure 8.1.1: Normal vectors of the plane formed by an ellipse.

Similar to the challenge of obtaining depth information for the central point of the ellipse, the current algorithm lacks sufficient information to determine the 3D positioning of the ellipse coordinate system with respect to the camera coordinate system. While the algorithm can obtain the directions of the major and minor axes of the ellipse (corresponding to the ellipse coordinate system, depicted as black vectors in Figure 8.1.2), only the projection in the X, Y plane of the image coordinate system is available (red vectors in Figure 8.1.2). Consequently, the missing components of the black x, y vectors in the red Z direction are needed to align the ellipse coordinate system with the image coordinate system.

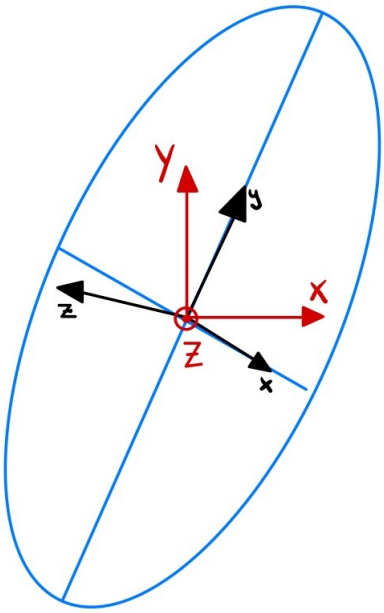


Figure 8.1.2: Normal vectors of the plane formed by an ellipse.

To obtain this missing Z component, an additional reference point within the docking port is required. This reference point will facilitate the placement of the coordinate system in 3D space.

Once this is achieved, the normal vector of the ellipse (black z vector in Figure 8.1.2) can be determined by performing the cross product of the x, y vectors from the ellipse.

To compute the orientation of the normal vector, a feature from the docking port must be selected to establish a reference point. Among the various points that could be chosen, one particularly distinctive feature stands out – the one marked in yellow in figure 8.1.3.

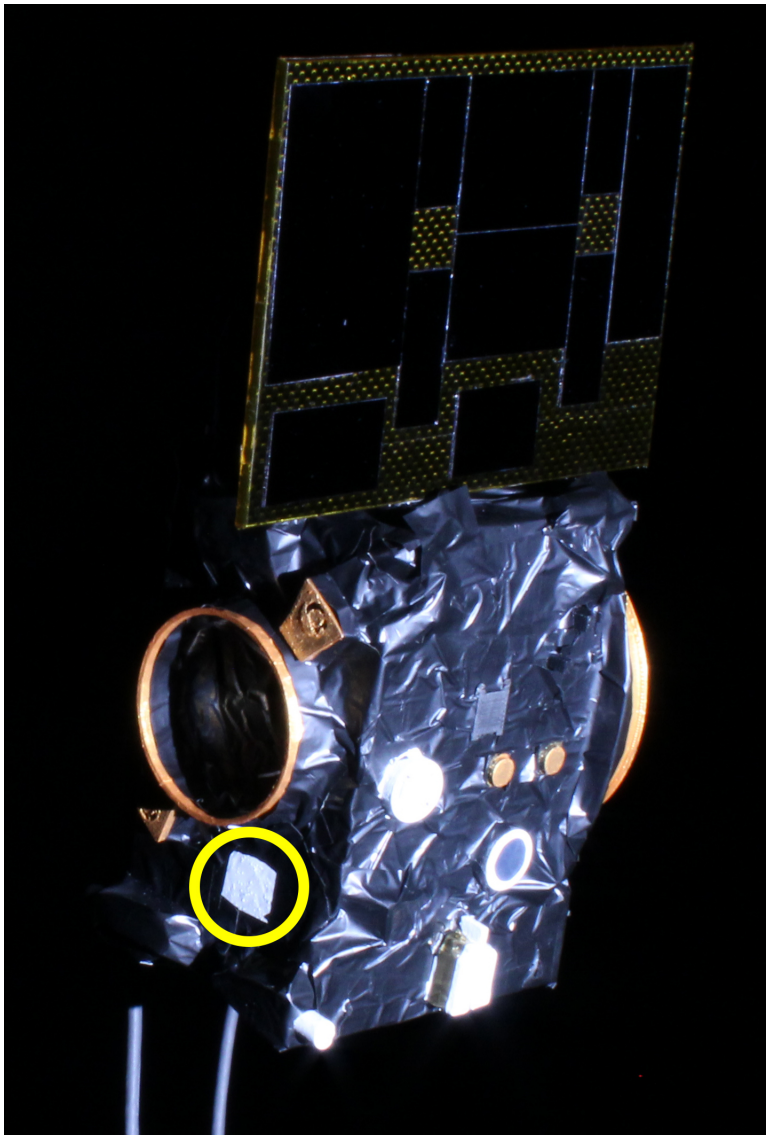


Figure 8.1.3: Reference feature for rotation.

For the detection of this feature, a procedure similar to the ellipse detection is applied. The program adapts image enhancement functions to each image, and instead of detecting an ellipse, it identifies a white square. Once the feature is detected, the center of the square is computed.

Now, knowing the position of the major and minor axis plus the center of the feature, we can define four different quadrants (Figure 8.1.4).

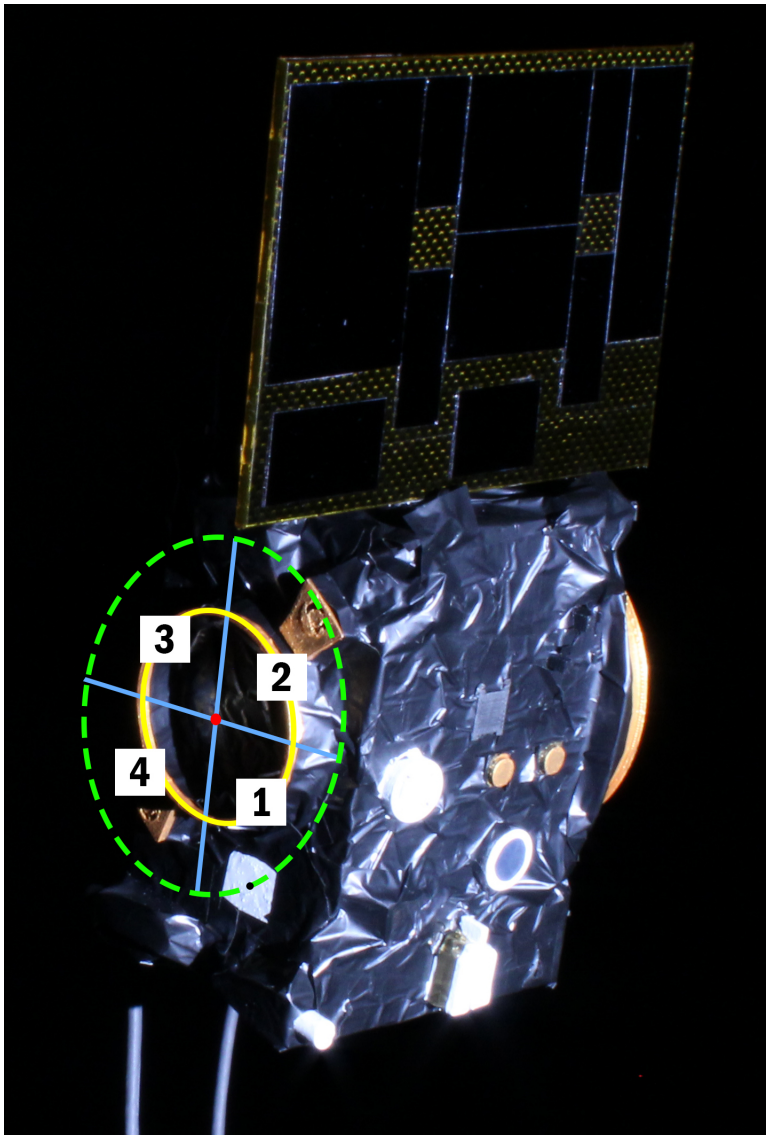


Figure 8.1.4: Quadrants for the reference feature for rotation.

It can be seen that in the arrangement shown in Fig. 8.1.4, when the dock is facing the camera, the center of the feature is placed in a specific position regarding the ellipse coordinate system. This is, having the major axis to the left, and the minor on top, regarding the ellipse coordinate system, so quadrant 1 in Fig. 8.1.4. Knowing this, and assuming that the center of the feature is in the same plane as the ellipse, the vector from the center of the ellipse to the center of the feature can be computed. This vector will be named $V_{center-feature}$. Also, the x, y, z of the ellipse coordinates should be placed as shown in figure 8.1.5.

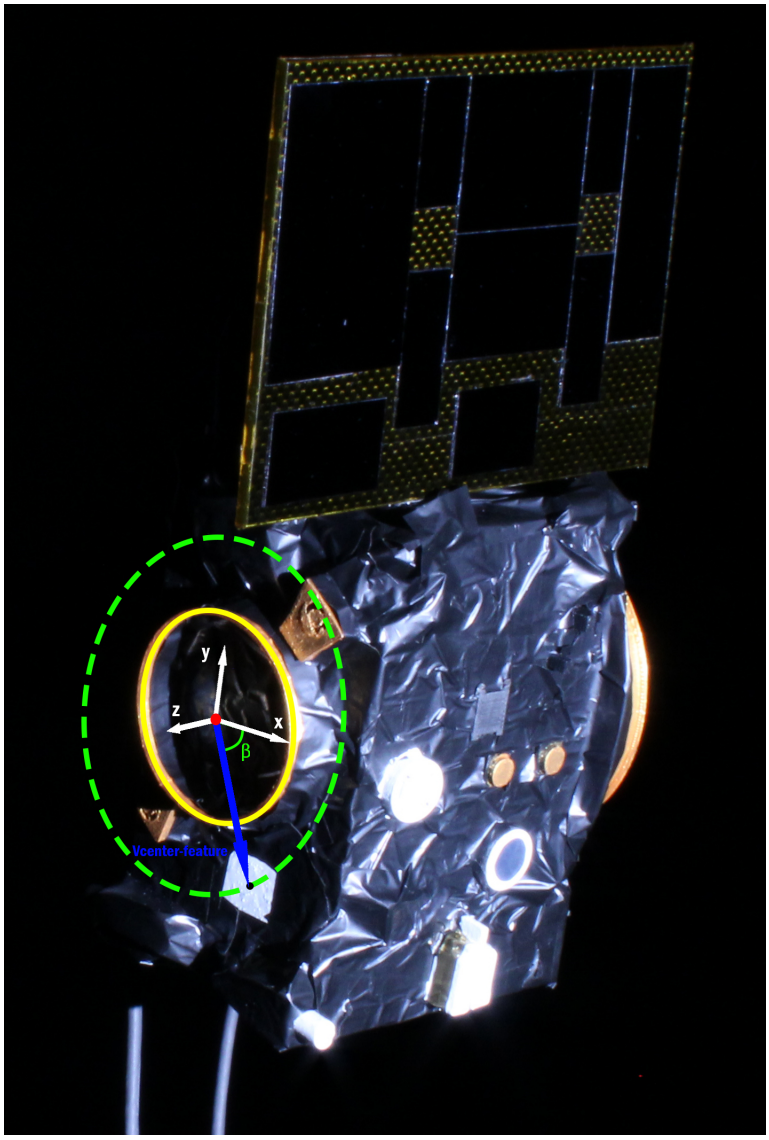


Figure 8.1.5: Quadrants for the reference feature for rotation.

It has to be taken into account that the β angle is lying on the x, y plane and that it is defined as the minimum angle between the $V_{center-feature}$ vector and the minor axis of the ellipse. The placement of the x, y, z coordinate system is computed in the following way:

$$\begin{cases} \textbf{x:} cos(\beta) \\ \textbf{y:} -sen(\beta) \\ \textbf{z:} V_{c-f-unit} \times x \end{cases} \tag{8.1.1}$$

Where $V_{c-f-unit}$ is the unit vector of $V_{center-feature}$.

In summary, after defining the center of the feature and assuming it is in the plane of the ellipse, then the vector connecting the center of the ellipse and the one from the feature has to be computed. With the unit vector of this vector called $V_{center-feature}$, the obtention of the coordinate system from the ellipse can be defined as shown in equations 8.1.1.

And with the last equation, the one for z the normal vector is obtained. Where the cross product of the unit vector of $V_{center-feature}$ and x is computed.

This ensures the orientation of the dock port is always correct and the rotation of it is always in the right place.

From figure 6.0.2 in the fame (c) it is possible to detect a red spot over the white square used as feature. This means that the algorithm of recognition was implemented in the project, but it wasn't tested and its reliability was not good enough to be taken in account. However, that could have been a starting point to add the rx, ry and rx rotations in the project.

8.2 Ellipse fitting

As it has been mentioned before in sec.3.1 a more efficient way to select the points to fit with the ellipses in the image must be implemented in order to reduce the computation time and to increase the precision of the detected shape. A good way to do so would be the implementation of cutting-edge algorithms such as the one proposed by *Zhang et al.2019*. Its working principle is similar to Libuda's algorithm but the time efficiency and the accuracy are higher. After the edge detection, points are classified according to the *gradient* direction of the edges (e_i). This is done because usually the outer edge of the ring has more contrast of the inner ring, because of the dark background outside the first ring. The classification used is associated to the quadrants of an ideal circumference and is reported below:

$$D(e_i) \begin{cases} I, dx > 0 \wedge dy > 0 \\ II, dx < 0 \wedge dy > 0 \\ III, dx < 0 \wedge dy < 0 \\ IV, dx > 0 \wedge dy < 0 \end{cases} \tag{8.2.1}$$

Edge points in the same quadrant are then connected following the *8-edge connectiveness* by fixing the allowed gradient within a certain range to form and arc α_i and eventually a pixel number treshold is used to discard short arcs. As an ellipse presents arcs in all the four quadrants, is then necessary to select arcs from different quadrants to fit the final shape. This is done by considering the convexity of the found arcs with the assumption that arcs in the first and second quadrant are considered convex, while arcs in the other two quadrants are considered concave. This property is measured by looking at the position of the arc with respect to the line that connects its starting and ending points. The followed criteria is shown below, here the symbol "+" is used to identify convex arcs, while "-" is used to identify concave arcs. Note that P^s is the arc starting point, P^e the ending point and P a point contained in the arc.

$$C(\alpha_i) \begin{cases} +, (P^sP^e \times P^sP) \cdot (0,0,1) > 0 \\ -, (P^sP^e \times P^sP) \cdot (0,0,1) < 0 \end{cases} \tag{8.2.2}$$

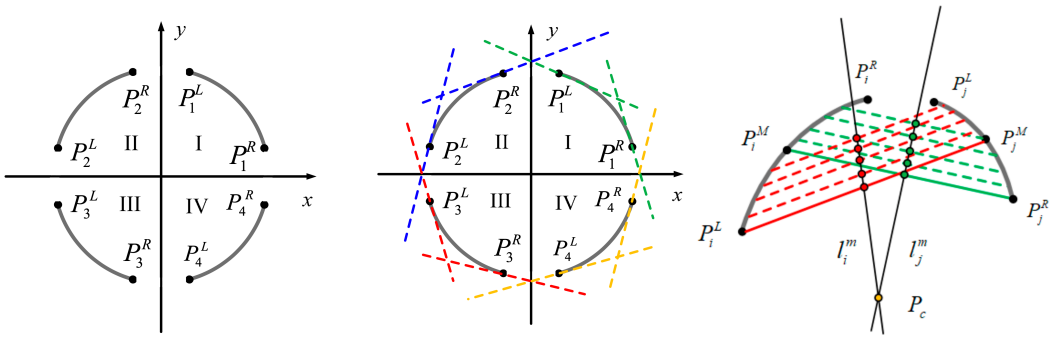
The classification of the arc segment can then be written by the combination eq.8.2.1 and eq.8.2.3 to give the final classification structure used to collect the arcs:

$$Q(\alpha_i)\begin{cases} I, D(e_i) = I \wedge C(\alpha_i) = + \\ II, D(e_i) = II \wedge C(\alpha_i) = + \\ III, D(e_i) = III \wedge C(\alpha_i) = - \\ IV, D(e_i) = IV \wedge C(\alpha_i) = - \end{cases} \tag{8.2.3}$$

This step breaks the spatially connectivity of the contour with gradient direction changing sharply, and removes some arcs that do not belong to the ellipse.[6] In order to fit an ellipse, at least 3 arcs in 3 different quadrants must be found. To do so, a routine is implemented to couple two arcs of the same ellipse first (figure 8.2.1) and then a third arc is added in a following step. For the selection of the first two arcs, the following criteria are given:

1. Quadrant constraint: the two arcs must be located in adjacent quadrants
2. Position constraint: assuming the two arcs are part of the same ellipse, the endpoints of the arcs are constrained by their relative position in the image
3. Tangent constraint: tangent lines are drew from the endpoints of the arcs perpendicular to the end direction of the gradient of the endpoint. Mathematical conditions must be verified to select the proper arc.[6]

After two arcs have been selected parallel lines are used to define two lines l_i^m and l_j^m are used to find the center of the ellipse as it's shown in figure 8.2.1.



(a) Quadrant constraint and position constraint

(b) Tangent constraint

(c) Center computation

Figure 8.2.1: Graphical representation of arcs selection criteria. Adapted from *Zhang et al.2019*

The selection of the third arc needs to be done after the first two are coupled. This is done by considering couple of arcs that meet the desired conditions in adjacent quadrants and then considering that all the arcs part of the same ellipse share the same center. The parameters of the ellipse are then found with the same method used in this project but what really makes the difference is the selection parameter used to filter the ellipses which most likely describe the docking port. The parameter used to estimate the goodness of the fit is the distance *d* between the edge point on the arc and the fitted ellipse E_i.

$$d(P_i)\begin{cases} = 0, P_i \text{ on the boundary of } E_i \\ < 0, P_i \text{ outside } E_i \\ > 0, P_i \text{ inside } E_i \end{cases} \tag{8.2.4}$$

After the definition of a set *K* of of edge points close to the elliptic boundary, the score $s \in [0, 1]$ defines how well the three arcs forming E_i fit the ellipse following $s = \frac{K}{K_i+K_j+K_k}$ where $K_{i,j,k}$ represents the number of pixels in each of the three arcs. A graphical representation of this process is given in the following figure 8.2.2

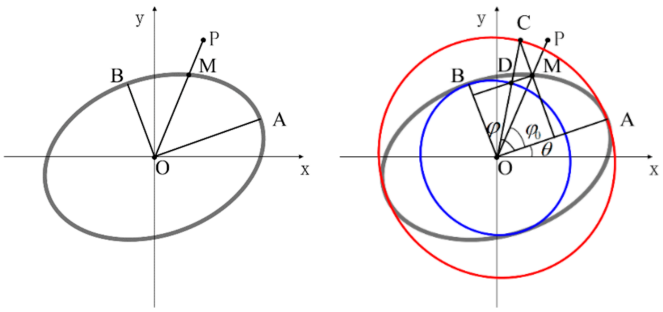


Figure 8.2.2: Distance ellipse-points determination. On the *left* the definition of ellipse is shown while on the *right* the distance computation is represented.[6]

This method of ellipse fitting has been proven to be very reliable and around 10 times faster than Libuda's algorithm, however it's implementation is far more complicated and is reccomended for future implementations.

8.3 Testing and accuracy estimation

Throughout this document, we've emphasized that the project was executed with a limited number of test images. Unfortunately, due to availability constraints, accessing the DTU Space's laboratory for additional testing sessions proved challenging. A more extensive set of images, featuring diverse camera positions in 3D space, would have significantly enhanced the thoroughness of our code testing. Such a dataset would provide a more comprehensive understanding of the algorithm's robustness in detecting both the ellipse and the distinctive feature.

If additional time had been available in the laboratory, obtaining more reference points from the spacecraft would have been a priority. This approach could have further improved the precision of positioning the ellipse in 3D space. As previously mentioned, the current information available is insufficient for computing all three components (x, y, z) of the ellipse coordinate system accurately.

Moreover, due to similar constraints in availability and time, we couldn't conduct measurements for the real placement of the camera in 3D space. This type of data would be invaluable for assessing the robustness of the program. A comparison between the code's output regarding the position of the ellipse's center in 3D space and its real-life position would provide valuable insights.

In summary, an expanded dataset with diverse camera arrangements, additional reference points, and real-world camera placement measurements would not only bolster the code's robustness but also contribute to a more accurate evaluation of its performance.

These considerations remain crucial for future iterations of the project and would undoubtedly enhance the algorithm’s reliability and applicability in various scenarios.

9 Main

In the code below it is shown the project flow of the main. It declares the steps of the algorithm.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include "functions.hpp"

using namespace cv;
using namespace std;

#define CV_8U   0
#define CV_8S   1
#define CV_16U  2
#define CV_16S  3
#define CV_32S  4
#define CV_32F  5
#define CV_64F  6
#define MAX_RAND 100

/*
-----------------------------------------------------------------------
----------- STEP 1: READ ALL THE IMAGES FROM THE FOLDER -----------
----------- STEP 2: READ THE IMAGE WITH THE GRAY        -----------
----------- STEP 3: FILTER TO OBTAIN THE THE ELLIPSES   -----------
----------- STEP 4: DRAW THE ELLIPSES ON THE IMAGE      -----------
----------- STEP 5: DETECT AND DRAW THE COLOSE SQUARE   -----------
----------- STEP 6: CALCULATE THE DISTANCE              -----------
----------- STEP 7: CALCULATE THE X,Y POSITION          -----------
----------- STEP 8: SAVE THE VALUES IN A CSV FILE       -----------
----------- STEP 9: ON MATLAB DISPLAY THE POSITIONS     -----------
-----------------------------------------------------------------------
*/

// IMAGES PATH
string img40 = "/Users/giovanniorzalesi/Desktop/IA proj/set/Set1_00040.jpg";
string folder = "/Users/giovanniorzalesi/Desktop/IA proj/set";
string pos = "/Users/giovanniorzalesi/Desktop/IA proj/pos.csv";

int main() {
    // Save the parameters into a file .csv
    ofstream outputFile(pos);
    // IMAGE FOLDER
    vector<String> filenames;
    glob(folder, filenames, false);

    // LOOP THROUGHT THE IMAGES
    for(auto const &f : filenames){
        cout << string(f) << endl;
        // Load the images

        Mat img = imread(f);
        Mat img_grey = imread(f, IMREAD_GRAYSCALE);

        // Adaptive Gaussian Blur
        int kernel = calculateAdaptiveBlurKernelSize(img);
```

```
        Mat blur;
        GaussianBlur(img_grey, blur, Size(kernel, kernel), 0, 0);

        // Adaptive Canny Edge detection
        int lowThreshold = calculateAdaptiveLowThreshold(blur);
        int highThreshold = calculateAdaptiveHighThreshold(blur);
        Mat canny;
        Canny(blur, canny, lowThreshold, highThreshold);

        // Find contours
        vector<vector<Point>> contours;
        findContours(canny, contours, RETR_TREE, CHAIN_APPROX_TC89_KCOS);

        // Ellipse fitting
        vector<RotatedRect> ellipses = filterEllipses(contours);
        drawEllipses(img, ellipses);

        // Calculate the diameter of the ellipse to estimate the distance
        // Extract major axes
        float majorAxis = max(ellipses[0].size.width, ellipses[0].size.height)
    ;

        // Distance extraction
        double distance = calculateDistance(majorAxis);

        // Translation extraction
        Point2f center = ellipses[0].center;
        vector<double> translation = translationCamera(center.x, center.y);

        // DECTECT THE CENTRE OF THE RECTANGLE
        Mat squares;
        threshold(img_grey, squares, 200, 255, THRESH_BINARY);
        vector<vector<Point>> lines;
        findContours(squares, lines, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
        Point center_rectangle = detectWhiteSquares(lines, img);
        circle(img, center_rectangle, 5, Scalar(0, 0, 255), 20);

        // DISPLAY
        imshow("img", img);
        waitKey(0);

        if(outputFile.is_open()){
            cout << "input" << endl;
            // Write x,y,z,rotation in case

            outputFile << translation[0] << "," << -translation[1] << "," <<
    distance << "," << 6.9 << endl;


        } else cerr << "couldn't";
    }
    // Close the file
    outputFile.close();
    return 0;
}
```

10 Functions

In the code below are shown all the functions used. Each one of them is associated in the main.

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <cmath>
#include <fstream>
#include <opencv2/calib3d.hpp>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/imgproc.hpp>

using namespace cv;
using namespace std;

/*
-----------------------------------------------
----------- FEATURE TRANSFORMATION -----------
-----------------------------------------------
*/

// GREY IMAGE
Mat grey_image(Mat img){
    int width = img.rows;
    int height = img.cols;
    Mat grey(width, height, CV_8UC1);   // Creating a new image with one
    channel

    int x, y, z;

    for(int i = 0; i<width;i++){
        for(int j = 0;j<height;j++){
            x = img.at<Vec3b>(i, j)[0];
            y = img.at<Vec3b>(i, j)[1];
            z = img.at<Vec3b>(i, j)[2];

            grey.at<uchar>(i,j) = (x + y + z) / 3;
        }
    }
    return grey;
}

// RETURN HISTOGRAM GRAY
Mat createHistrogramGray(Mat img){
  unsigned char value = 0; // index value for the histogram (not really needed
    )
  int histogram[256]; // histogram array - remember to set to zero initially
  int width = img.cols; // say, 320
  int height = img.rows; // say, 240

  int k = 256;
  while (k-- > 0)
    histogram[k] = 0; // reset histogram entry
  for (int i = 0; i < height; i++) {
```

```
    for (int j = 0; j < width; j++) {
      value = img.at<uchar>(i, j);
      histogram[value] += 1;
    }
  }

  Mat histo(200, 256, CV_8UC3, Scalar(0, 0, 0));
  Point p1, p2;
  k = 256;
  while (k-- > 0) {
    p1.x = k;
    p1.y = histo.rows - 1;
    p2.x = k;
    p2.y = histo.rows - 1 - histogram[k] / 1000;
    line(histo, p1, p2, Scalar(255, 255, 255), 2, LINE_4);
  }
  return histo;
}

// RETURN BINARY IMAGE
Mat binary(Mat img, unsigned char threshold) {
    Mat bin = img.clone();
    unsigned char value = 0;
    int width = bin.cols;
    int height = bin.rows;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            value = bin.at<uchar>(i, j);
            if (value > threshold) {
                bin.at<uchar>(i, j) = 255;
            }
            else {
                bin.at<uchar>(i, j) = 0;
            }
        }
    }
    return bin;
}

/*
-----------------------------------------------
----------- ELLIPSE EXTRACTION     -----------
-----------------------------------------------
*/

// CALCULATE KERNEL
int calculateAdaptiveBlurKernelSize(const Mat& image) {
    double scaleFactor = 0.75;
    Mat gradX, gradY;
    Sobel(image, gradX, CV_32F, 1, 0);
    Sobel(image, gradY, CV_32F, 0, 1);

    Mat gradientMag;
    magnitude(gradX, gradY, gradientMag);

    Scalar mean, stddev;
    meanStdDev(gradientMag, mean, stddev);

    // Adjust the kernel size based on the standard deviation
    int blurKernelSize = static_cast<int>(stddev.val[0] * scaleFactor);

```

```
115     // Ensure the kernel size is an odd number
116     blurKernelSize |= 1;
117 
118     return blurKernelSize;
119 }
120 
121 // BLUR LOW THRESHOLD (ADAPTIVE)
122 int calculateAdaptiveLowThreshold(const Mat& image) {
123     Mat blurred;
124     medianBlur(image, blurred, 7); // Use an appropriate kernel size, for
        example, 5
125 
126     // Calculate the median pixel value of the blurred image
127     Scalar medianValue = mean(blurred);
128 
129     // Set the low threshold as a fraction (e.g., 0.5) of the median value
130     int lowThreshold = static_cast<int>(medianValue[0] * 11);
131 
132     return lowThreshold;
133 }
134 
135 // BLUR HIGH THRESHOLD (ADAPTIVE)
136 int calculateAdaptiveHighThreshold(const Mat& image) {
137     double highToLowRatio = 1.4;
138     // Calculate the high threshold as a multiple of the low threshold
139     int lowThreshold = calculateAdaptiveLowThreshold(image);
140     int highThreshold = static_cast<int>(lowThreshold * highToLowRatio);
141 
142     return highThreshold;
143 }
144 
145 // Function to calculate eigenvalues from trace and determinant
146 vector<double> calculateEigenvalues(double discriminant, double trace) {
147     vector<double> eigenvalues;
148 
149     // Calculate the eigenvalues
150     double lambda1 = (trace + sqrt(discriminant)) / 2.0;
151     double lambda2 = (trace - sqrt(discriminant)) / 2.0;
152 
153     // Store the eigenvalues in the vector
154     eigenvalues.push_back(lambda1);
155     eigenvalues.push_back(lambda2);
156 
157     return eigenvalues;
158 }
159 
160 RotatedRect fitEllipse2(const vector<Point>& points) {
161     //number of points to fit
162     int numPoints = points.size();
163 
164     // Ensure we have enough points to fit an ellipse
165     if (numPoints < 5) {
166         throw invalid_argument("Insufficient points for ellipse fitting");
167     }
168 
169     // Formulate the least squares problem Ax = b for the ellipse parameters
170     Mat A(numPoints, 5, CV_64F);
171     Mat bTerm(numPoints, 1, CV_64F);
172 
173     for (int i = 0; i < numPoints; ++i) {
174         double x = points[i].x;
175         double y = points[i].y;
```

```
176
177         A.at<double>(i, 0) = x * x;
178         A.at<double>(i, 1) = x * y;
179         A.at<double>(i, 2) = y * y;
180         A.at<double>(i, 3) = x;
181         A.at<double>(i, 4) = y;
182         A.at<double>(i, 5) = 1;
183
184         bTerm.at<double>(i) = -1;  // Assuming the equation of the ellipse is
    Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0
185     }
186
187     // Solve the least squares problem to obtain ellipse parameters -> used
    SVD decomposition
188     Mat x;
189     solve(A, bTerm, x, DECOMP_SVD);
190
191     // Extract ellipse parameters
192     double A_val = x.at<double>(0);
193     double B_val = x.at<double>(1);
194     double C_val = x.at<double>(2);
195     double D_val = x.at<double>(3);
196     double E_val = x.at<double>(4);
197     double F_val = x.at<double>(5);
198
199     // Calculate trace and determinant of A_val
200     double trace_A = A_val + C_val;    //+F???   Tr(matrix)=A+C+F
201     // double det_A = A_val * C_val - B_val * B_val; TEST 1
202    // Det(matrix)= ACF + 2BDE - AE^2 - BD^2 - CF^2
203     double det_A = A_val * C_val * F_val + 2 * B_val * D_val * E_val - A_val *
     pow(E_val, 2) - B_val * pow(D_val, 2) - C_val * pow(F_val, 2);
204
205     // Calculate the discriminant
206     double discriminant = trace_A * trace_A - 4 * det_A;
207
208     RotatedRect ellipse(Point2f(0, 0), Size2f(0, 0), 0);
209 ;
210     if(discriminant >= 0){
211         // Calculate eigenvalues
212         vector<double> lambdaval = calculateEigenvalues(discriminant, trace_A)
    ;
213
214         // Calculate semimajor axis and semiminor axis from eigenvalues
215         double a = sqrt(1.0 / lambdaval[0]);
216         double b = sqrt(1.0 / lambdaval[1]);
217
218         // Calculate other ellipse parameters
219         double cx = (2 * C_val * D_val - B_val * E_val) / (B_val * B_val - 4 *
     A_val * C_val);
220         double cy = (2 * A_val * E_val - B_val * D_val) / (B_val * B_val - 4 *
     A_val * C_val);
221
222         double theta = 0.5 * atan2(B_val, A_val - C_val);
223
224         //Find semi major axis of the ellipse
225         double major_axis = max(a, b);
226
227         // Convert ellipse parameters to OpenCV's RotatedRect format
228
229         ellipse.center = Point2f(static_cast<float>(cx), static_cast<float>(cy
    ));
230         ellipse.size = Size2f(static_cast<float>(a * 2), static_cast<float>(b
```

```
    * 2)); // Use 'ellipse_b'
        ellipse.angle = static_cast<float>(theta * 180.0 / CV_PI);

        // Save the major axis of the ellipse
        cout << "Major axis (2*a) = " << ellipse.size.width << "\n" << endl;
        cout << "Angle of rotation = " << ellipse.angle << "\n" << endl;

        // Draw the ellipses on top of the previous image

        return ellipse;
    } else return ellipse;


}

// RECOGNISE ELLIPSES
vector<RotatedRect> filterEllipses(const vector<vector<Point>>& contours){
    vector<RotatedRect> ellipses;

    for (const auto& contour : contours) {
        // Keep track of the maximum aspect ratio
        double maxAspectRatio = 0.0;
        // Check if the contour has enough points to fit an ellipse
        if (contour.size() >= 100) {
            // Fit an ellipse to the contour
            RotatedRect ellipse = fitEllipse(contour);

            // if(ellipse.center == Point2f(0, 0) && ellipse.size == Size2f(0,
     0) && ellipse.angle == 0){
            //     break;
            // }

            // Filter based on certain criteria
            // For example, you can filter by aspect ratio, size, or any other
     relevant property
            double aspectRatio = ellipse.size.width / ellipse.size.height;

            if (aspectRatio > 0.65 && aspectRatio < 0.9) {
                // Add the ellipse to the list if it meets the criteria
                ellipses.push_back(ellipse);
                // If the current aspect ratio is greater than the maximum
    found so far
                if (aspectRatio > maxAspectRatio) {
                    // Update the maximum aspect ratio
                    maxAspectRatio = aspectRatio;

                    // Clear the existing ellipses and add the current one
                    ellipses.clear();
                    ellipses.push_back(ellipse);
                }
            }
        }
    }

    return ellipses;
}

// DROWING ELLIPSE
void drawEllipses(Mat& image, const vector<RotatedRect>& ellipses){
    for(const auto& ellipse : ellipses){
        // Draw the ellipse on the image in blue
        cv::ellipse(image, ellipse, Scalar(255, 0, 0), 7);
```

```
    }
}

/*
-----------------------------------------------
----------- SQUARE EXTRACTION      -----------
-----------------------------------------------
*/

// RECOGNISE SQUARES
Point detectWhiteSquares(const vector<vector<Point>>& contours, Mat image) {
    // Define a minimum area threshold for contours to filter out noise
    // Calculate the median contour area
    vector<double> contourAreas;
    for (const auto& contour : contours) {
        double area = contourArea(contour);
        contourAreas.push_back(area);
    }

    // Calculate the median contour area
    double medianContourArea = 0.0;
    if (!contourAreas.empty()) {
        size_t size = contourAreas.size();
        nth_element(contourAreas.begin(), contourAreas.begin() + size / 2,
    contourAreas.end());
        medianContourArea = contourAreas[size / 2];
    }

    // Use the median contour area to set adaptive thresholds
    const double maxContourArea = medianContourArea * 350;
    const double minContourArea = medianContourArea * 250;
    Point center;
    center = Point(-1, -1);

    for (const auto& contour : contours) {
        // Calculate the area of the contour
        double area = contourArea(contour);

        // Check if the contour is larger than the threshold
        if (area > minContourArea && area < maxContourArea) {
            // Approximate the contour to a polygon
            vector<Point> approx;
            approxPolyDP(contour, approx, arcLength(contour, true) * 0.03,
    true);

            // Check if the polygon has 4 corners (a square)
            if (approx.size() == 4) {
                // Calculate the center of the square
                Moments mu = moments(contour, false);
                center = Point(mu.m10 / mu.m00, mu.m01 / mu.m00);
            }
        }
    }
    return center;
}

/*
-----------------------------------------------
----------- X,Y,Z CALCULATION      -----------
-----------------------------------------------
*/

```

```
// RETURN THE PARAMETER DISTANCE OF THE CAMERA
double calculateDistance(double imgDiameter) {
    // Define the parameters
    double realDiameter = 6.9;  // cm
    double focal_length = 2.5 * 1.6; // cm  *APS parameter
    float pix_val = 3.71 * pow(10,-4);  // cm/pixel

    // Distance calculation
    double distance = focal_length * realDiameter / (imgDiameter * pix_val);

    return distance;
}

// RETURN THE TRANSLATION OF THE CAMERA
vector<double> translationCamera(double cxDock, double cyDock){
    // Define the parameters
    double Hfov = 1.78;
    double Vfov = 1.19;
    double xpixels = 6000;
    double ypixels = 4000;
    double focal_length = 2.5 * 1.6; // cm *APS parameter

    // Define the final vector
    vector<double> translation;

    // Translation from the centre image in pixels
    double cx = cxDock - xpixels/2;
    double cy = cyDock - ypixels/2;

    cx = cx * Hfov / (2*focal_length);
    cy = cy * Vfov / (2*focal_length);

    translation.push_back(cx);
    translation.push_back(cy);

    return translation;
}
```

11 Matlab

In the code below it's shown the Matlab code apt to plot the graphs for the project final section. With this code it is possible to see the results of the camera movements.

```
% Read the CSV file containing camera translations
data = readmatrix('/Users/giovanniorzalesi/Desktop/IA proj/pos.csv');

% Extract x, y, and z columns
x = data(:, 1);
y = data(:, 2);
z = data(:, 3);
d = data(1, 4);

% Create a 3D scatter plot
figure();
scatter3(x, y, z, 'filled', 'DisplayName', 'Camera Translations');
hold on;

% Plot the docking port as a circle with diameter 5 at (0, 0, 0) in red
radius = d/2;  % Half of the diameter
theta = linspace(0, 2*pi, 100);
circle_x = radius * cos(theta);
circle_y = radius * sin(theta);
circle_z = zeros(size(theta));  % Z-coordinate for the circle

plot3(circle_x, circle_y, circle_z, 'r', 'LineWidth', 2, 'DisplayName', '
    Docking Port');

title('Camera Translations and Docking Port in 3D');
xlabel('X Translation');
ylabel('Y Translation');
zlabel('Z Translation');
grid on;
legend;

% Set axis aspect ratio to be equal
axis equal;
hold off;
figure();
plot(x,y, '.');
title('X-Y plot');
xlabel('X Translation');
ylabel('Y Translation');
% Add numbers to each point
for i = 1:length(x)
    text(x(i), y(i), num2str(i), 'FontSize', 12, 'HorizontalAlignment', '
    center', 'VerticalAlignment', 'bottom');
end

grid on;
figure();
plot(x,z, '.');
title('X-Z plot');
xlabel('X Translation');
ylabel('Z Translation');
% Add numbers to each point
for i = 1:length(x)
```

```
    text(x(i), z(i), num2str(i), 'FontSize', 12, 'HorizontalAlignment', '
    center', 'VerticalAlignment', 'bottom');
end

grid on;
figure();
plot(y,z, '.');
title('Y-Z plot');
xlabel('Y Translation');
ylabel('Z Translation');
% Add numbers to each point
for i = 1:length(x)
    text(y(i), z(i), num2str(i), 'FontSize', 12, 'HorizontalAlignment', '
    center', 'VerticalAlignment', 'bottom');
end

grid on;
```

Bibliography

[1] James R. Wertz and Robert Bell. “Autonomous rendezvous and docking technologies: status and prospects”. In: *Space Systems Technology and Operations*. Ed. by Jr. Tchoryk Peter and James Shoemaker. Vol. 5088. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. Aug. 2003, pp. 20–30. DOI: 10.1117/12.498121.

[2] Caroline Specht. “Tube-based model predictive control for the approach maneuver of a spacecraft to a free-tumbling target satellite”. In: (Jan. 2016).

[3] NASA. *International Docking System Standard (IDSS) Interface Definition Document (IDD) , Revision F*. 2022.

[4] Shengyong Zhang et al. “Research on docking ring pose estimation method based on point cloud grayscale image”. In: *Advances in Space Research* 70.11 (2022), pp. 3466–3477. ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2022.08.015. URL: https://www.sciencedirect.com/science/article/pii/S0273117722007347.

[5] Lars Libuda, Ingo Grothues, and Karl-Friedrich Kraiss. “Ellipse Detection in Digital Image Data Using Geometric Features”. In: vol. 4. Jan. 2006, pp. 175–180. ISBN: 978-3-540-75272-1. DOI: 10.1007/978-3-540-75274-5_15.

[6] Zhang, Pan, and Jiaying Ma. “Real-Time Docking Ring Detection Based on the Geometrical Shape for an On-Orbit Spacecraft”. In: *Sensors* 19 (Nov. 2019), p. 5243. DOI: 10.3390/s19235243.

[7] Richard E Woods and Rafael C Gonzalez. *Digital image processing*. 2008.

[8] Andrew Fitzgibbon, M. Pilu, and Robert Fisher. “Direct Least-squares fitting of ellipses”. In: vol. 21. Sept. 1996, 253–257 vol.1. ISBN: 0-8186-7282-X. DOI: 10.1109/ICPR.1996.546029.

[9] Shahrokh Heidari et al. “Quantum red–green–blue image steganography”. In: *International Journal of Quantum Information* 15 (Aug. 2017), p. 1750039. DOI: 10.1142/S0219749917500393.

[10] Sukumar Katamreddy et al. “Visual Udder Detection with Deep Neural Networks”. In: Dec. 2018, pp. 166–171. DOI: 10.1109/ICSensT.2018.8603625.

[11] Gurmeet Kaur and Rupinder Kaur. “Image De-Noising using Wavelet Transform and Various Filters”. In: *International Journal of Research in Computer Science* 2 (Feb. 2012). DOI: 10.7815/ijorcs.22.2012.017.

[12] Arkadiusz Kobiera. “Ellipse Hyperbola and Their Conjunction”. In: (May 2018).

[13] Alan Edelman and Yuyang Wang. “The GSVD: Where are the Ellipses?, Matrix Trigonometry, and More”. In: *SIAM Journal on Matrix Analysis and Applications* 41.4 (2020), pp. 1826–1856. ISSN: 1095-7162. DOI: 10.1137/18m1234412. URL: http://dx.doi.org/10.1137/18M1234412.

[14] Euijin Kim, Miki Haseyama, and Hideo Kitajima. “Fast line extraction from digital images using line segments”. In: *Systems and Computers in Japan* 34 (Sept. 2003), pp. 76–89. DOI: 10.1002/scj.10124.

[15] R. H. Gallagher. “Optimization: Theory and applications, S. S. Rao, Wiley Eastern Ltd. No. of pages: 711”. In: *International Journal for Numerical Methods in Engineering* 14.11 (1979), pp. 1734–1734. DOI: https://doi.org/10.1002/nme.1620141118. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620141118. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620141118.

[16] Liao Chengwang. “Chapter 22 - Singular Value Decomposition in Active Monitoring Data Analysis”. In: *Active Geophysical Monitoring*. Vol. 40. Handbook of Geophysical Exploration: Seismic Exploration. Pergamon, 2010, pp. 421–430. DOI: https://doi.org/10.1016/S0950-1401(10)04027-9. URL: https://www.sciencedirect.com/science/article/pii/S0950140110040279.

[17] Gonzalo Simarro, Daniel Calvete Manrique, and Paola Souto-Ceccon. “UCalib: Cameras Autocalibration on Coastal Video Monitoring Systems”. In: *Remote Sensing* 13 (July 2021), p. 2795. DOI: 10.3390/rs13142795.

[18] KYK Wong. “Camera calibration from surfaces of revolution”. In: *Ieee Transactions On Pattern Analysis And Machine Intelligence, 2003, v. 25 n. 2, p. 147-161* (2003).

[19] Mahmoud Hassaballah, Abdelmgeid Ali, and Hammam Alshazly. “Image Features Detection, Description and Matching”. In: vol. 630. Feb. 2016, pp. 11–45. ISBN: ISBN 978-3-319-28852-9. DOI: 10.1007/978-3-319-28854-3_2.

[20] *https://opencv.org*.

[21] H. Huang. “Active Pixel Sensor”. In: *Encyclopedia* (). URL: https://encyclopedia.pub/entry/36045.