POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Politecnico di Torino

Master's Degree Thesis

Software Implementation of Image Processing Techniques for EV Socket Pose Estimation with Neural Network Support

Supervisors

Prof. Marcello CHIABERGE

Dr. Marina MONDIN

Dr. Fereydoun DANESHGARAN

Candidate

Francesco MAULA

October 2024

Abstract

The increasing adoption of electric vehicles (EVs) highlights the growing need for efficient and safe charging systems. A significant challenge in automating these systems is the accurate detection and alignment of the charging socket, especially in high-power environments where manual handling poses safety risks and reduces operational efficiency. This thesis, developed at California State University Los Angeles in collaboration with *InnoTech System LLC*, focuses on the *CCS Type 1* charging socket, a widely adopted standard in North America, with the aim of automating its detection and pose estimation.

The motivation for this work arises from the need to automate the charging process to enhance both safety and efficiency. A pre-trained *YOLOv8* neural network was used to detect the socket and provide an initial estimate of its position, but further refinement was required. Detecting both the position and orientation of the socket is essential to ensure proper plug alignment, a critical aspect for fully automated systems.

The main objective of this thesis is to develop a system capable of detecting not only the position of the socket but also its orientation. The process begins with the *YOLOv8* neural network's estimate, followed by a series of refinement steps using *OpenCV* in Python for image processing. Testing under various environmental conditions ensures robustness and reliability in real-world scenarios, where factors such as lighting and occlusions may affect performance.

This thesis integrates machine learning with image processing to address these challenges, presenting a reliable and adaptable solution for the automation of EV charging systems. The research is structured across key chapters, including a review of the state of the art, an in-depth discussion of the components used, a detailed explanation of the detection and pose estimation methods, and a comprehensive evaluation of the system's performance through extensive testing.

Acknowledgements

After five years of study, it is now time to bid farewell to the *Politecnico di Torino* with this final project. This journey has been filled with challenges, but also abundant rewards, and I extend my deepest gratitude to everyone who supported me along the way.

I would like to begin by reflecting upon the path that led to this moment. My experience in Los Angeles was unforgettable. Although the culture and habits were very different from what I was used to, after just a few weeks I started to feel at home. For this, I am especially grateful to my American supervisors, Dr. Marina Mondin and Dr. Fereydoun Daneshgaran, who offered me both professional and personal support. I also want to express my gratitude to my friend, Taus Enrico, who was an ideal companion throughout this journey. I am equally grateful to the incredible people I met at *California State University*, who created such a positive environment and provided all the resources I needed to complete my research.

Furthermore, I want to extend my heartfelt thanks to my Italian supervisor, Prof. Marcello Chiaberge, for his invaluable support throughout this period. I am also very grateful to my friends and colleagues, who remained with me during good and tough times, offering constant help and encouragement.

Finally, I owe everything to my family, whose endless love and unwavering support made this achievement possible. This thesis is not just the result of hard work, but also the reflection of the immense support I received from so many along the way.

Thank you all.

"Life is beautiful"
To my family

Table of Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence

CCS Combined Charging System

CDF Cumulative Distribution Function

CNN Convolutional Neural Network

CTMA Cluster Template Matching Algorithm

DLT Direct Linear Transform

DoF Degrees of Freedom

EPnP Efficient Perspective-n-Point

EV Electric Vehicle

LED Light Emitting Diode

LS Least Squares

NN Neural Network

PnP Perspective-n-Point

SDK Software Development Kit

TCP Tool Center Point

Chapter 1

Introduction

1.1 Background and Motivation

The advancement of autonomous technologies has significantly impacted various industries, leading to the development of automated systems for tasks that require precision, efficiency, and safety. One such area of development is the automation of charging systems for electric vehicles (EVs). The motivation behind this thesis project addresses the growing demand for electric mobility solutions and the need for an efficient and reliable charging infrastructure.

Automated charging systems are crucial for the future of electric mobility. As the adoption of electric vehicles continues to increase, there is a corresponding need for an automatic charging infrastructure. Manual handling of charging connectors poses risks to human safety, particularly in environments where high-power charging is required. By automating the charging process, these risks can be mitigated, ensuring safer and more efficient operation.

Although the automotive sector is a primary focus, the need for automated charging systems extends beyond electric vehicles for personal use. Heavy industrial environments, such as ports, also face significant challenges due to the physical demands of handling large and heavy cables for high-current charging. In these settings, automated charging solutions are not only a matter of convenience but are essential for worker safety, as cables required to support large-scale machinery and logistics vehicles can be cumbersome and difficult to manage manually. Similar needs arise in the logistics and infrastructure sectors, where efficient charging systems are

critical to maintaining continuous operations and minimizing downtime.

Moreover, current solutions for automated charging systems are often costly, making their widespread adoption challenging for both the automotive and industrial sectors. The motivation behind this research is to develop a more affordable automated charging system while maintaining the reliability and performance expected from high-quality solutions. By reducing the cost of these systems, the research aims to make automated charging technology more accessible to a broader range of applications, including fleet management, public charging stations, and heavy-duty industrial operations.

A key component of an automated charging system is the accurate detection and positioning of the charging socket. This process involves using advanced computer vision techniques and machine learning algorithms to identify the location and orientation of the socket. The detection system typically uses cameras and sensors to capture images and depth information of the charging port area of the vehicle.

Recent advances in neural networks, such as the *YOLOv8* model, have enabled real-time, highly accurate detection of objects, including charging sockets. These models can process visual data to locate the socket with high precision, even in varying lighting conditions and complex environments. Using depth sensors improves the ability of the system to understand the position and shape of objects, which is important for accurately positioning and aligning the plug.

Automating the connection process eliminates direct human contact with the charging interface, significantly reducing the risks associated with manual handling. Furthermore, automated systems can be designed to handle the mechanical stresses and demands of repeated high-power charging cycles more effectively than manual operations.

In addition to safety, automated charging systems offer improved efficiency and convenience. They can operate continuously, and their integration with smart grid technologies can optimize energy use. This is particularly important for fleet operations and public charging infrastructure, where high throughput and reliability are critical.

CCS Type 1 Plug

The project focuses on Electric Vehicle (EV) sockets. As shown in Figure 1.1, the type of EV charging connector varies significantly across different regions and models. One of the most widely adopted standards for EV charging, particularly in North America, is the ***CSS Type 1* plug**. This plug combines a standard AC connector with a high-speed DC connector, allowing both types of charging to be accessed through a single port.

The *Type 1* connector (*SAE J1772*) has five pins and operates with single-phase AC from the mains for charging EV. The *Combined Charging System (CCS)* connector builds on the J1772 charging inlet by adding two additional pins below it. This combination of a Type 1 connector with high-speed charging pins gives *CCS* its name and supports high-power charging (up to 350 kW), making it suitable for a variety of EVs and charging scenarios. In North America, almost every automaker, with the exception of Tesla, has adopted the *CCS* standard.

In the context of automated charging systems, the *CSS Type 1* plug presents unique challenges and opportunities. Its design requires precise detection strategies, alignment, and a secure connection to ensure efficient power transfer and safety.

Focusing on the accurate detection of the pose of *CSS Type 1* socket, this research aims to develop a robust and versatile automated charging solution capable of handling one of the most common standards in the EV industry.

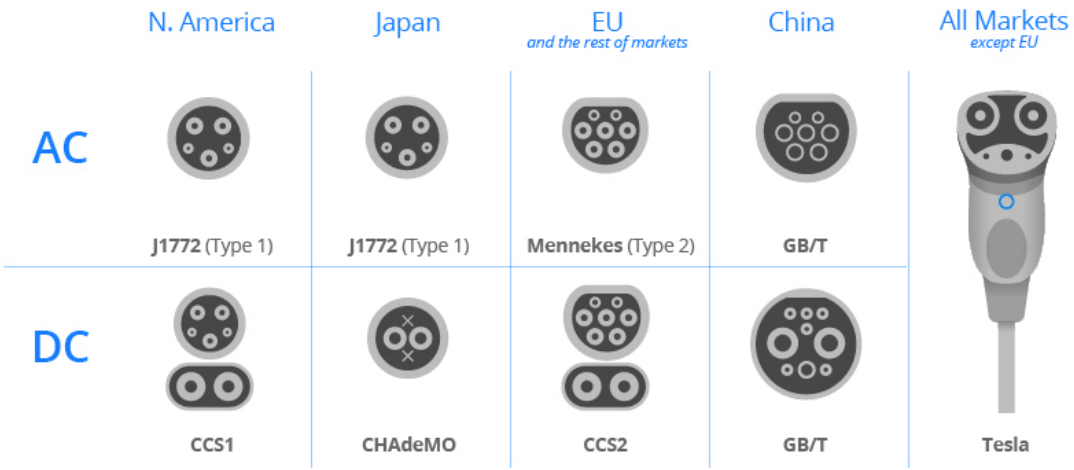


Figure 1.1: EV Connector Types across the world

1.2 Problem Statement and Objectives

The primary goal of this project is to develop a system that accurately detects the position and orientation of *CSS Type 1* charging socket. Initial detection of the plug's position is accomplished using a pre-trained neural network with stereo camera images. The objective is to refine this estimate using additional camera data to accurately determine the position and orientation of the socket. To achieve this goal, the following objectives are identified:

- *Initial Detection*: Utilize the pre-trained neural network and stereo camera images to detect the plug and provide an initial estimate of the position of the center.
- *Orientation Retrieval*: Develop and implement algorithms to further refine the position of the socket and to estimate precisely the orientation, using the stereo camera data.
- *Integration with 5 DoF Robot*: Adapt the system for a 5 Degrees of Freedom robotic arm.
- *Validation and Testing*: Test the system in various conditions to ensure robust performance, validating the accuracy of both the position and orientation estimations facilitated by the stereo camera setup.

1.3 Research Hypotheses

In developing this system, the following research hypotheses are formulated to guide the project:

- *Hypothesis 1*: The neural network, using stereo camera images, provides a valid and reliable initial detection of the socket, which serves as a basis for subsequent position refinement and orientation determination.
- *Hypothesis 2*: By aligning the plug in the roll orientation relative to the 5 Degrees of Freedom (DoF) robotic arm, one degree of freedom can be effectively excluded. This alignment simplifies movement operations without compromising the accuracy of position and orientation determination for EV socket interaction.

1.4 Methodology Overview

This research employs a systematic approach to achieve precise detection and pose estimation of the *CCS Type 1* charging socket. The methodology integrates both hardware and software components to ensure robustness and precision. Key aspects include:

- *System Integration*: Utilizing a robotic arm for automated plug alignment.
- *Initial Detection*: Using a pre-trained neural network to provide initial position estimates from stereo camera images.
- *3D Perception*: Using the *ZED Mini* stereo camera to capture depth information essential for pose estimation.
- *Image Processing*: Applying techniques to enhance image quality for accurate feature extraction.
- *Feature Detection and Classification*: Detecting and classifying features to refine initial estimates.
- *Pose Estimation*: Employing a pose estimation algorithm for precise pose determination.
- *Optimization and Validation*: Tuning parameters and validating performance through extensive testing.

This methodology ensures a comprehensive and effective solution for detecting and positioning the *CCS Type 1* plug.

1.5 Thesis Structure

The thesis is organized as follows:

- *Chapter 2: State of the Art*
  This chapter reviews recent developments in the field of automatic EV charging robots and provides a detailed examination of the methodologies used to detect and retrieve poses of objects, focusing on current image processing techniques.

- *Chapter 3: Components and Technical Background*
  This chapter introduces the main components used in the study, including the robotic arm and the stereo camera. It discusses their technical specifications, features, and theoretical foundations relevant to their use in the project.

- *Chapter 4: Socket Detection and Pose Estimation*
  This chapter details the proposed approach for socket detection and pose estimation. Covers the neural network architecture, image processing techniques, and algorithms used for estimating the socket's position and orientation.

- *Chapter 5: Test and Results*
  This chapter presents the results of the tests conducted to validate the system. It includes a detailed analysis of the system's performance in terms of circle detection, and the accuracy of the position and orientation determinations.

- *Chapter 6: Conclusion*
  The final chapter summarizes the findings of the research, discusses the implications of the results, and suggests directions for future work.

Chapter 2

State of the Art

The first part of this chapter presents a general overview of recent developments in automatic EV charging robots. The second part offers a more in-depth examination of the methodologies used for detecting and retrieving object poses, focusing on the current state of the art in image processing techniques.

2.1 Recent Developments

This section provides a general review of recent developments in automatic charging robots, highlighting their main features.

Volkswagen Mobile Charging Robot

One of the initial developments identified in the research is the *Volkswagen Group* mobile charging robot. The primary function of this prototype is to autonomously charge vehicles in confined parking areas, such as underground car parks. Activated via an app or Car-to-X communication, the charging robot operates entirely autonomously. Navigates to the vehicle, communicates with it, opens the charging socket flap, connects the plug, and later disconnects it, all without human intervention. To charge multiple vehicles simultaneously, the robot transports a mobile energy storage unit to the vehicle, connects it, and uses it to charge the vehicle's battery. The storage unit remains with the vehicle throughout the charging process, while the robot moves on to charge other vehicles. After charging is complete,

the robot retrieves the energy storage unit and returns it to the central charging station [1].

Siemens Autonomous Charging System

The *Siemens Autonomous Charging System* is an advanced solution for automatic charging of EVs with standardized CCS connectors. Presented at IAA Mobility in Munich, this system uses a robot to connect to a vehicle charging port in a minute. The robot moves on all spatial axes to connect to the vehicle charging port. The system provides up to 300 kW of power, with plans to achieve 1 MW for heavy-load transport vehicles. Artificial intelligence and optical sensors determine the precise position and orientation of the charging port, even in adverse weather conditions. This technology is suitable for autonomous vehicles, standard EVs, and heavy-load transport vehicles, handling variations in vehicle types, port placements, and parking positions. A prototype in close production has been tested under real conditions [2].

Rocsys Autonomous Charging System

Figure 2.1: *RocSys ROC-1* Autonomous Charging System

A commercially available product is the *Rocsys Autonomous Charging System*, which was showcased at the *Advanced Clean Transportation Expo (ACT)* in Las Vegas in May 2024. The *ROC-1* robot, shown in Figure 2.1, automates the entire

process of connecting the charging connector to the electric vehicle. Communicates with the vehicle throughout the docking process, from the initial connection to the end of the charge cycle. Rocsys' hands-free charging solutions use Artificial Intelligence (AI)-based computer vision, patented soft robotics, and remote services. The advanced vision system captures 3D information using a single camera, enabling the robot to navigate the plug to the socket with precision. Deep-learning algorithms ensure functionality in various weather conditions. Integrated LED lighting allows for continuous operation. In addition, soft robotics technology enables safe plug connection in demanding environments, supporting ongoing customer activities. Robotics can handle sudden shocks, ensuring reliable performance without interrupting customer operations [3].

2.2 Methodological Approaches

The aim of this section is to provide a comprehensive overview of the methodologies, algorithms, and technologies developed and utilized to accurately detect and determine the pose of the sockets.

Since one of the requirements of the project was the use of a stereo camera to detect and estimate the pose of the electrical socket, one of the first recent implementations analyzed was the one presented by *Tadic* [4]. In the project, a new approach for automatic charging socket detection using a *ZED 2i* depth sensor was proposed. Moreover, some common image processing techniques were used. The proposed algorithm utilizes both RGB and depth images captured by the *ZED 2i* depth sensor. The process involves the following steps:

1. *Preprocessing*: Converts the image to grayscale and applies the intensity transformation using the gamma function and contrast stretching to enhance the image.

2. *Noise Reduction*: Applies filtering to remove noise.

3. *Morphological Operations*: Performs a series of morphological operations to obtain a binary mask of the socket area and determine the center coordinates.

4. *Socket Detection*: Uses logical operations to combine the binary mask with the thresholded and binarized RGB images to detect the socket region.

5. *Depth Map Analysis*: Extracts the socket area from the original depth map using logical operations.

6. *Pose Estimation*: Determines the tilt angles of the socket in the XY, XZ, and YZ planes using the depth information and binary mask.

Another recent development in this area is presented by *Pan et al.* [5]. Their project involves the use of a 6 DoF robotic arm and a single lens camera, with a particularly interesting strategy. The proposed technique is characterized by the following steps:

1. *Recognition*: Uses a Convolutional Neural Network (CNN) to identify the charging port. The CNN model is trained with images of charging ports under various lighting conditions to achieve high recognition accuracy.

2. *Location*: Employs a pose measurement algorithm based on circle features. This method involves preprocessing the image for brightness adjustment and denoising, followed by segmentation, edge detection, and ellipse fitting to determine the precise pose of the charging port.

Quan et al. [6] proposed a different method divided into two main stages:

1. *Rough Positioning*: Uses *Hough* circle and *Hough* line algorithms to locate the charging port by identifying circular and linear features. This stage provides an approximate position of the charging port.

2. *Precise Positioning*: Uses the *Canny* operator to extract contour information from both the original and gradient images. The contours are fitted into ellipses using the *Quadratic Curve Standardization (QCS)* method. The *Perspective-n-Point (PnP)* algorithm is then employed to determine the exact position of the charging port.

Moreover, *Quan et al.* [7] proposed a furher method for recognizing and localizing an EV charging port based on a *Cluster Template Matching Algorithm (CTMA)* and divided into two phases:

1. *Search Phase*: In the search phase, images are collected, converted to grayscale, and undergo bilateral filtering. Contours are detected using the *Canny* algorithm and smaller contours are eliminated. The feature circle algorithm fits

the remaining contours to ellipses, and irrelevant ellipses are discarded. Once six or more feature points are identified, their pixel coordinates are converted to a position matrix, and the *Efficient Perspective-n-Point (EPnP)* algorithm calculates the pose of the socket relative to the camera.

2. *Aiming Phase*: In the aiming phase, images are taken in front of the charging port and templates are created using feature and gradient extraction software. Bilateral filtering is applied to these images, and contours are extracted with the *Canny* operator. The *CTMA* matches the feature points by evaluating the contour information, reducing the matching time, and improving the robustness. Finally, the *EPnP* algorithm refines the pose of the socket, guiding the robot to complete the plug-in accurately.

Table 2.1 groups the principal characteristics of each analyzed method, highlighting their main advantages and disadvantages.

Methods	**Advantages**	**Disadvantages**
Tadic [4]: image processing techniques from a stereo camera.	High detection accuracy (94%). Robust to noise.	Sensitive to inadequate lighting.
Pan et al. [5]: CNN for socket recognition followed by image processing and ellipse fitting for pose estimation.	High recognition accuracy (98.9%). Precise location (position error within 1.4 mm, angle error within 1.6°).	Requires complex preprocessing and precise calibration.
Quan et al. [6]: *Hough* circle algorithm followed by precise contour extraction and fitting.	High recognition accuracy (97.9% for rough positioning, 94.8% for the entire system). Precise location (displacement error: 0.60 mm, 0.83 mm, 1.23 mm; angular errors: 1.19°, 0.97°, 0.50°). Robust in various environments.	Requires precise camera and hand-eye calibration. Sensitivity to strong light conditions.

Continue on the next page

Methods	**Advantages**	**Disadvantages**
Quan et al. [7]: Image processing with contour extraction, *CTMA* and *EPnP* algorithm.	High efficiency and accuracy. Effective in various light and environmental conditions. High success rate (95%).	Multi-step process with sophisticated preprocessing. Sensitivity to extreme lighting conditions.

Table 2.1: Methods comparison, advantages and disadvantages

Chapter 3

Components and Technical Background

In this chapter, a detailed overview of the main components used in the research is provided, specifically focusing on the robot and the stereo camera system. The theoretical principles underlying each component are also introduced to provide a comprehensive understanding of their functionalities and applications.

3.1 The Robotic Arm

For the project, the *UFactory xArm 5* robotic arm was utilized. The robot, shown in Figure 3.1, is an anthropomorphic arm of 5 DoF from *Shenzen UFactory Company*.

The following sections provide an introduction to the main theoretical concepts in robotics and a discussion of the key features of the *UFactory* robotic arm used in this project.

3.1.1 Reference Systems and Transformation Matrices

Reference systems and transformation matrices are fundamental concepts in robotics. A reference system, or coordinate frame, provides a structured way to define the position and orientation of objects in space. In robotics, multiple reference frames are often used, such as the base frame, joint frames, and the end-effector frame.

Figure 3.1: *UFactory xArm 5-DoF* Robotic Arm

Transformation matrices are mathematical tools that are used to convert coordinates from one reference frame to another. These matrices encapsulate both rotation and translation information, allowing for seamless transitions between different frames of reference.

A typical transformation matrix is a homogeneous 4x4 matrix, which includes a 3x3 rotation matrix R and a translation vector $\mathbf{t} = [t_x, t_y, t_z]^T$, as shown in Equation 3.1.

$$\mathbf{T} = \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

The transformation matrix $\mathbf{T}$ can be used to relate the coordinates of a point $\mathbf{P}$ in one frame to its coordinates in another frame. If $\mathbf{P}_A$ represents the coordinates of the point in the frame A and $\mathbf{P}_B$ represents the coordinates in the frame B, the relationship can be expressed as shown in Equation 3.2.

$$\mathbf{P}_B = \mathbf{T}_{A \to B} \mathbf{P}_A \tag{3.2}$$

Here, $\mathbf{P}_A$ and $\mathbf{P}_B$ are expressed in homogeneous coordinates, an extension of Cartesian coordinates that enables more convenient mathematical manipulations of transformations. A point in a 3D space with Cartesian coordinates (x, y, z) can be represented in homogeneous coordinates as $(x, y, z, 1)$.

Euler angles are a method to describe the orientation of a rigid body with respect to a fixed coordinate system. They consist of three angles, typically called roll

(ϕ), pitch (θ), and yaw (ψ). These angles represent rotations on the axes of the coordinate system and can be used to construct the rotation matrix R, as shown in Equation 3.3. $R_x(\phi)$, $R_y(\theta)$, and $R_z(\psi)$ are the rotation matrices around the x, y, and z axes, respectively.

$$R = R_z(\psi)R_y(\theta)R_x(\phi) \tag{3.3}$$

The *UFactory xArm 5* has two main coordinate systems, as shown in Figure 3.2:

- A: *Base Coordinate System*, based on the mounting base of the robotic arm.
- B: *Tool Coordinate System*, which consists of the Tool Center Point (TCP) and coordinate orientation.

Moreover, the user can define additional reference systems, called user coordinate systems (C).

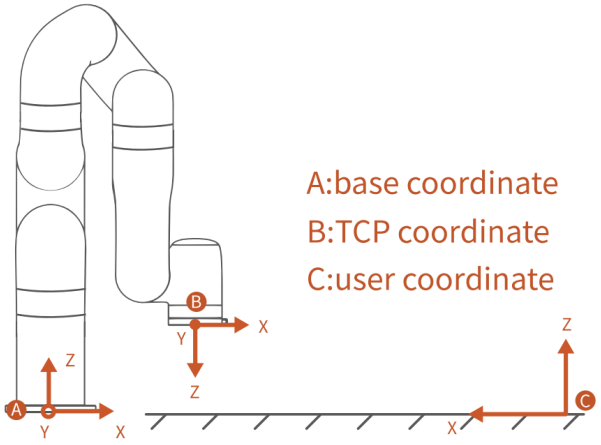
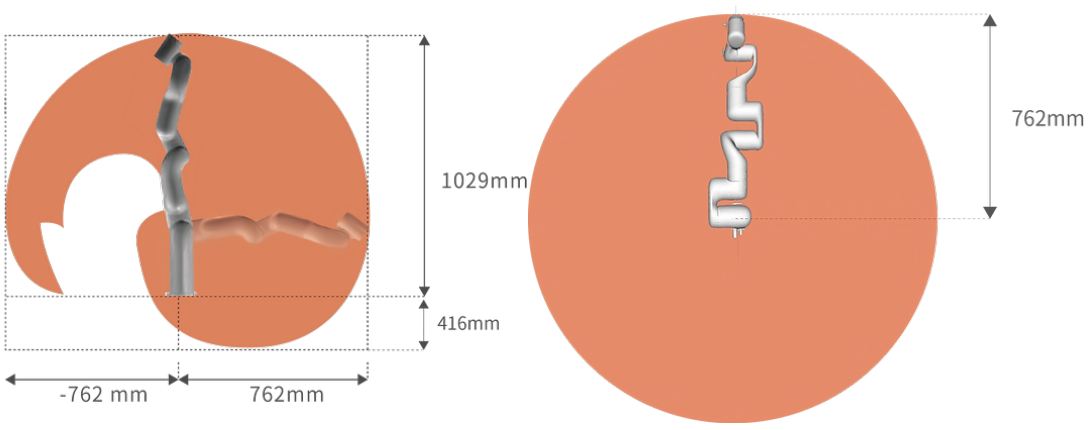


Figure 3.2: *UFactory xArm 5* Coordinate Systems

3.1.2 Main Features and Performances

The *UF xArm 5* is made up of five rotational joints, each of which allows a single degree of motion. An index of robot performance is the **workspace**, which is the region described by the origin of the end-effector frame when all the manipulator joints execute all possible motions. Figure 3.3 reports the robot workspace in terms of side view and top view.



(a) Side view **(b)** Top view

Figure 3.3: *UFactory xArm 5* Workspace

Another significant performance metric for robots is **repeatability**, which measures the manipulator's ability to return to a previously reached position. It relies on the mechanical structure's characteristics, the resolution of transducers, and the control strategy implemented by the robot's software. The *UF xArm 5* has a repeatability of ± 0.1 mm, indicating its ability to reliably return to previously reached positions.

3.2 Stereo Camera

A stereo camera is a type of camera constituted by two or more lenses. In the project, the *ZED Mini* camera of *Stereolabs* has been used. It is equipped with two lenses. The camera configuration allows it to simulate human binocular vision, giving it the ability to capture three-dimensional images. In the following paragraph, a more in-depth study of the camera model and stereo photography will be conducted.

3.2.1 Pinhole Camera Model

The pinhole camera model is characterized by equations which describe the formation of an image through the projection of 3D points onto the image plane. The geometry of the model consists of the following elements:

- The center of projection C, also known as the *optical center*, where the origin

of a Euclidean coordinate system is defined.

- The line from the camera center perpendicular to the image plane, called the *principal axis* or *principal ray*. The Z axis of the coordinate system is parallel to this line.

- The plane $Z = f$, known as the *image plane* or the *optical plane*.

- The plane passing through the center of the camera parallel to the image plane, known as the *principal plane*.

- The point of intersection between the principal axis and the image plane, called the *principal point*.

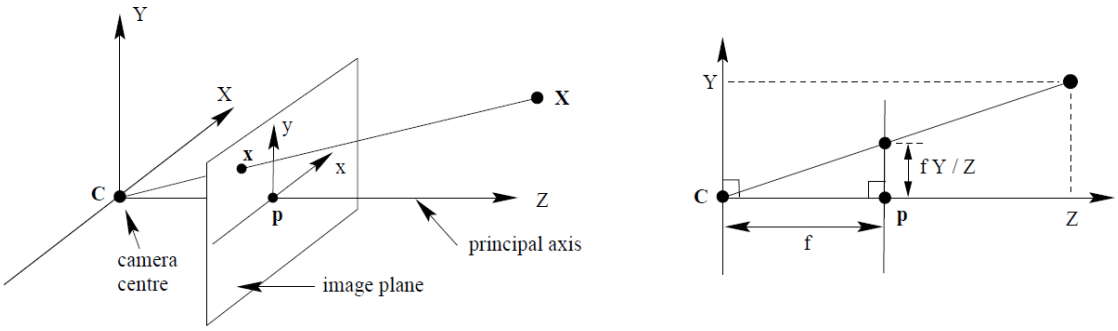


Figure 3.4: Pinhole camera model geometry

As depicted in Figure 3.4, using triangle proportions, a point in space with coordinates $[X, Y, Z]^T$ is mapped onto the image plane as $[\frac{fX}{Z}, \frac{fY}{Z}, f]^T$. The central projection can be expressed as a linear mapping between the world homogeneous coordinates and the image ones, using matrix multiplication, as shown in Equation 3.4. This means that in the 2D coordinate system placed in the image plane with origin at p, the point $x = \frac{fX}{Z}$ and $y = \frac{fY}{Z}$.

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{\text{cam}} \\ Y_{\text{cam}} \\ Z_{\text{cam}} \\ 1 \end{bmatrix} \tag{3.4}$$

3.2.2 Intrinsic and Extrinsic Camera Parameters

In the pinhole model, the optical center in the image plane is assumed to be at (0,0). In reality, the camera has a different principal point (c_x, c_y) which differs from the point p. Moreover, the focal lengths (f_x, f_y) are defined in terms of pixel dimensions for the x-axis and the y-axis, respectively. These characteristics are included in the **intrinsic camera parameters** and define the *intrinsic camera matrix*, also called the *calibration matrix*. Thus, Equation 3.4 is modified with the new parameters to obtain Equation 3.5, where K represents the calibration matrix.

$$\begin{bmatrix} f_x X + Z c_x \\ f_y Y + Z c_y \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{\text{cam}} \\ Y_{\text{cam}} \\ Z_{\text{cam}} \\ 1 \end{bmatrix} = K \begin{bmatrix} X_{\text{cam}} \\ Y_{\text{cam}} \\ Z_{\text{cam}} \\ 1 \end{bmatrix} \tag{3.5}$$

The previous equations are referred to as the camera coordinate system. The **extrinsic (or external) parameters** relate the world coordinate system to the camera coordinate system. This relation is represented by the *extrinsic camera matrix*. It consists of a matrix 3×4 formed by combining a 3×3 rotation matrix R and a 3×1 translation vector t. This matrix transforms points from world coordinates to camera coordinates. Let $\mathbf{X}_w = [X_w, Y_w, Z_w, 1]^T$ be the coordinates of the point in the world coordinates, $\mathbf{X}_{\text{cam}}$ is given by Equation 3.6.

$$\mathbf{X}_{\text{cam}} = [R|t]\mathbf{X}_w = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \mathbf{X}_w \tag{3.6}$$

3.2.3 Stereoscopic Vision

Stereo vision allows to estimate the depth of a point object from the camera using two cameras. It replies the 3D perception of the human vision, which is based on the concept of triangulation of rays from multiple viewpoints. Firstly, it is important to analyze the geometrical components of the system. In order to simplify the explanation and the calculation, a parallel stereo system is considered. This means that the two cameras have their optical axis aligned horizontally, as in human vision.

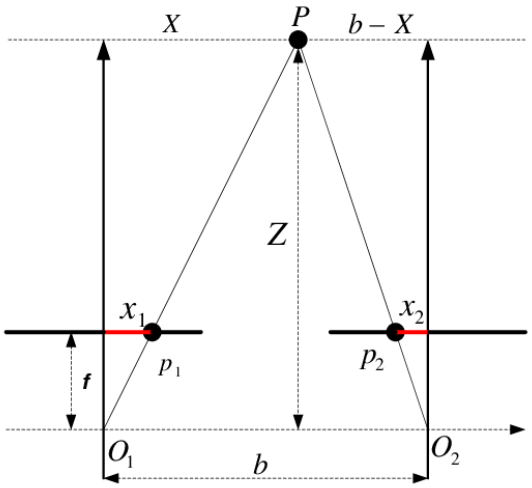


Figure 3.5: Stereoscopic vision geometry

As shown in Figure 3.5, the distance between the optical centers O_1 and O_2 of the left and right cameras is called the *baseline* b. The perception of depth is based on the concept of disparity, which is the difference in image location of the same 3D point when projected under perspective to two different cameras. Since a parallel stereo system is considered, the two images differ only in the x coordinates of their respective image plane. Consequently, the disparity d can be expressed as $d = x_2 - x_1$, where x_1 and x_2 are the coordinates of the point considered in the left and right images, respectively.

Using the proportion of triangles, it is possible to construct the system of equation 3.7.

$$\begin{cases} X : x_1 = (Z - f) : f \\ (b - X) : x_2 = (Z - f) : f \end{cases} \tag{3.7}$$

Solving the system by isolating the unknown depth Z, the result obtained is shown in Equation 3.8.

$$Z = \frac{f \cdot b}{x_2 - x_1} = \frac{f \cdot b}{d} \tag{3.8}$$

The method of determining depth from disparity is called **triangulation**. The focal length f and the baseline b are typically obtained from the stereo camera datasheet or through the camera calibration process. The objective of triangulation is to establish correspondences between points in the left and right images to compute the disparity.

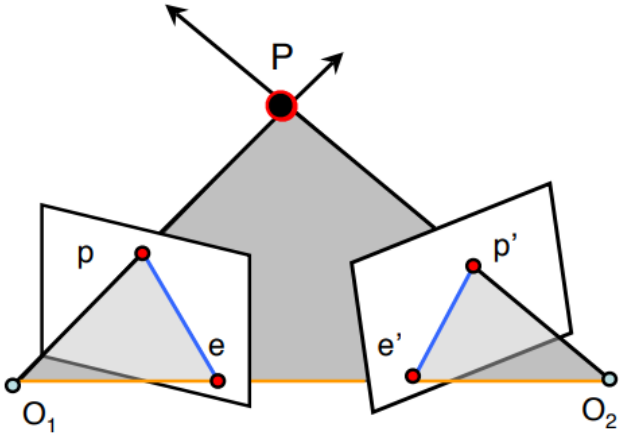


Figure 3.6: Epipolar geometry

The correspondence problem relies on the principles of *epipolar* geometry. In Figure 3.6, the two camera centers O_1 and O_2, along with a 3D point of interest P, define a plane known as the *epipolar plane*. Let p and p' be the corresponding points of P in the left and right images, respectively. The points e and e', where the baseline O_1O_2 intersects the image planes of the left and right cameras, are termed *epipoles*. The intersection of the epipolar plane with the image plane defines the *epipolar lines*. For instance, the line formed by p and e represents an epipolar line for the left image.

Given p as the observation of the point P in 3D world coordinates, the corresponding observation p' in the right camera should lie on the epipolar line. This imposes a constraint on the possible locations of p' in the second image. Assuming calibrated cameras (i.e., known intrinsic and extrinsic parameters), the concept of epipolar lines facilitates establishing a relationship between p and p'.

The epipolar geometry is simplified when the image planes of the two cameras coincide. In such cases, the epipolar lines are parallel to the baseline O_1O_2, allowing the determination of the corresponding points by scanning along horizontal lines (epipolar constraint).

In scenarios involving non-parallel systems, image rectification transforms images to share a common image plane, simplifying the correspondence problem by analyzing only horizontal lines.

A fundamental algorithm for finding the corresponding pixels is the *Block Matching algorithm*. It involves comparing a small window around a point in the left image with multiple small windows along the same horizontal line in the second image (assuming rectified images). For each pair of windows, a loss function is computed. The point $(\bar{x}, y)$ in the second image with the minimum loss is considered the best match for the point (x, y) in the first image. Consequently, the disparity value at the coordinate (x, y) is calculated as $d(x, y) = \bar{x} - x$. The Sum of Absolute Differences (SAD) is a commonly used loss function, defined as:

$$SAD(W^L, W^R) = \sum_{i=1}^{N} \sum_{j=1}^{M} \left| W_{ij}^L - W_{ij}^R \right| \tag{3.9}$$

Another useful function is the Sum of Squared Differences (SSD), defined as:

$$SSD(W^L, W^R) = \sum_{i=1}^{N} \sum_{j=1}^{M} (W_{ij}^L - W_{ij}^R)^2 \tag{3.10}$$

Generally, SAD is preferred over SSD due to its speed and robustness to noise and outliers.

3.2.4 ZED Mini

Figure 3.7: *ZED Mini* Stereo Camera

For the project, the *ZED Mini* Stereo Camera (Figure 3.7) from *Stereolabs* has been used. The principal property of the *ZED Mini* camera lies in its ability to capture high-quality stereoscopic 3D images and videos. This camera utilizes two lenses, allowing for depth perception and creating immersive visual experiences. Its compact size and versatility make it suitable for various applications such as virtual

reality, augmented reality, 3D scanning, and robotics. Furthermore, the *ZED Mini* camera offers real-time depth-sensing and spatial mapping capabilities, making it a valuable tool for developers and researchers in the field of computer vision and spatial understanding. The technical specifications are reported in Table 3.1.

An important aspect of *Stereolabs* products is the *ZED Software Development Kit (SDK)*, whose functional diagram is shown in Figure 3.8. As indicated in Table 3.1, the *ZED SDK* leverages the computational power of a GPU to perform complex tasks in real-time. For this project, an *Nvidia Jetson Nano* GPU has been utilized, and the next section will provide more information about this processing unit. The *SDK* is responsible for depth perception, rectifying the images from the two cameras, and computing the disparity to generate the depth map in real time. In addition, the software includes comprehensive APIs with support for both C++ and Python, which are useful for accessing and manipulating camera data, as well as retrieving images, videos, and depth maps.

As an example and for documentation purposes, the following Python code demonstrates how to retrieve all intrinsic parameters of the left camera using the *SDK* API:

```
import pyzed.sl as sl
    ...
    # Create a ZED camera object
    cam = sl.Camera()
    ...
    # Retrieving left camera parameters
    intrinsic = cam.get_camera_information().camera_configuration.
                calibration_parameters.left_cam
    fx = intrinsic.fx
    fy = intrinsic.fy
    cx = intrinsic.cx
    cy = intrinsic.cy
    k1, k2, p1, p2, k3 = intrinsic.disto
```

The `cam` object represents the camera from which it is possible to manipulate and retrieve all the information of the *ZED Mini*. `fx, fy, cx, cy` are the intrinsic parameters, as explained in the previous section. `k1, k2, p1, p2, k3` are the distortion coefficients, which are usually zero because the SDK provides undistorted images directly to the user. This code is used in the project because these parameters are essential for one of the main steps to retrieve the socket pose.

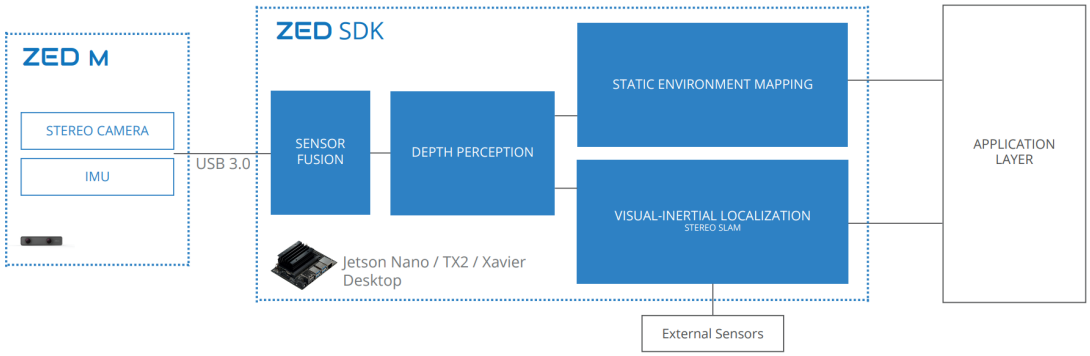


Figure 3.8: Functional *ZED SDK* Diagram [8]

Table 3.1: Features of *ZED Mini* Stereo Camera [8]

Features	**ZED Mini**
Size and weight	Dimensions: 124.5 × 30.5 × 26.5 mm Weight: 62.9 g
Working temperature	0°C to +45°C
Connectivity	USB 3.0 Type-C port (5 V/380 mA)
Lenses	Field of View: Max. 90° (H) × 60° (V) × 100° (D) Focal length: 2.8 mm f/2.0 aperture
Resolutions	Side by Side 2K: 2 × (2208 × 1242) @ 15fps HD1080: 2 × (1920 × 1080) @ 30fps HD720: 2 × (1280 × 720) @ 60fps VGA: 2 × (672 × 376) @ 100fps
Depth Sensing	Baseline: 63 mm Depth Range: 0.10 m – 15 m Depth Map Resolution: Native Video Resolution Depth Accuracy: $< 1.5\%$ up to 3 m, $< 7\%$ up to 15 m
Motion	Motion Sensors: Gyroscope, Accelerometer Technology: Visual-inertial stereo SLAM
SDK Requirements	Dual-core 2.3 GHz or faster Minimum 4GB RAM Memory Nvidia GPU with Compute Capability ≥ 3.0

3.3 The GPU

A Graphics Processing Unit (GPU) is a specialized electronic circuit designed to accelerate the processing of images and complex calculations, particularly those involving parallel processing. It is widely used in applications such as gaming, artificial intelligence, and real-time data processing due to its high computational power and efficiency.

For this project, a *Nvidia Jetson Nano* (Figure 3.9) was utilized. It is a compact and powerful device specifically designed for edge AI applications. Its main characteristics [9] include the following:

- *CUDA Cores:* 128-core Maxwell architecture, providing substantial parallel processing capabilities.
- *Performance:* Capable of delivering up to 472 GFLOPs of computational power, making it suitable for AI and deep learning tasks.
- *Memory:* 4 GB of LPDDR4 memory, ensuring smooth handling of large datasets and complex models.
- *Connectivity:* Includes multiple interfaces such as USB 3.0, HDMI, and Gigabit Ethernet, allowing for versatile connectivity options.
- *Power Efficiency:* Designed to operate with a power consumption of as low as 5 to 10 watts, making it energy-efficient for embedded systems.

Figure 3.9: *Nvidia Jetson Nano*

Chapter 4

Socket Detection and Pose Estimation

4.1 Proposed Approach

The proposed approach for socket detection and pose estimation utilizes several techniques analyzed in the state of the art. Figure 4.1 summarizes all the steps. The inputs to the proposed solution are the darker blocks in the scheme: the left image, the depth map, and the calibration parameters, all retrieved from the *ZED Mini* Stereo Camera.

The approach starts with utilizing the existing neural network for initial detection, specifically the *YOLOv8* model, known for its strong performance in real-time object detection and segmentation tasks. Following initial detection, several image processing techniques are applied to refine the results, including grayscale conversion, median filtering for noise reduction, histogram equalization, and power law (gamma) transformation to enhance image contrast [4]. Subsequent steps involve image binarization and edge detection using the *Canny* algorithm, facilitating contour extraction. Contour extraction is crucial for the next phase, where least-squares fitting identifies elliptical shapes within the contours. This step is essential for accurately classifying and fitting circles. Once identified, socket pose estimation is performed using the *Perspective-n-Point (PnP)* algorithm, calculating the 3D pose of the socket based on the 2D coordinates of the detected circles. The following sections will provide a detailed explanation of each step.

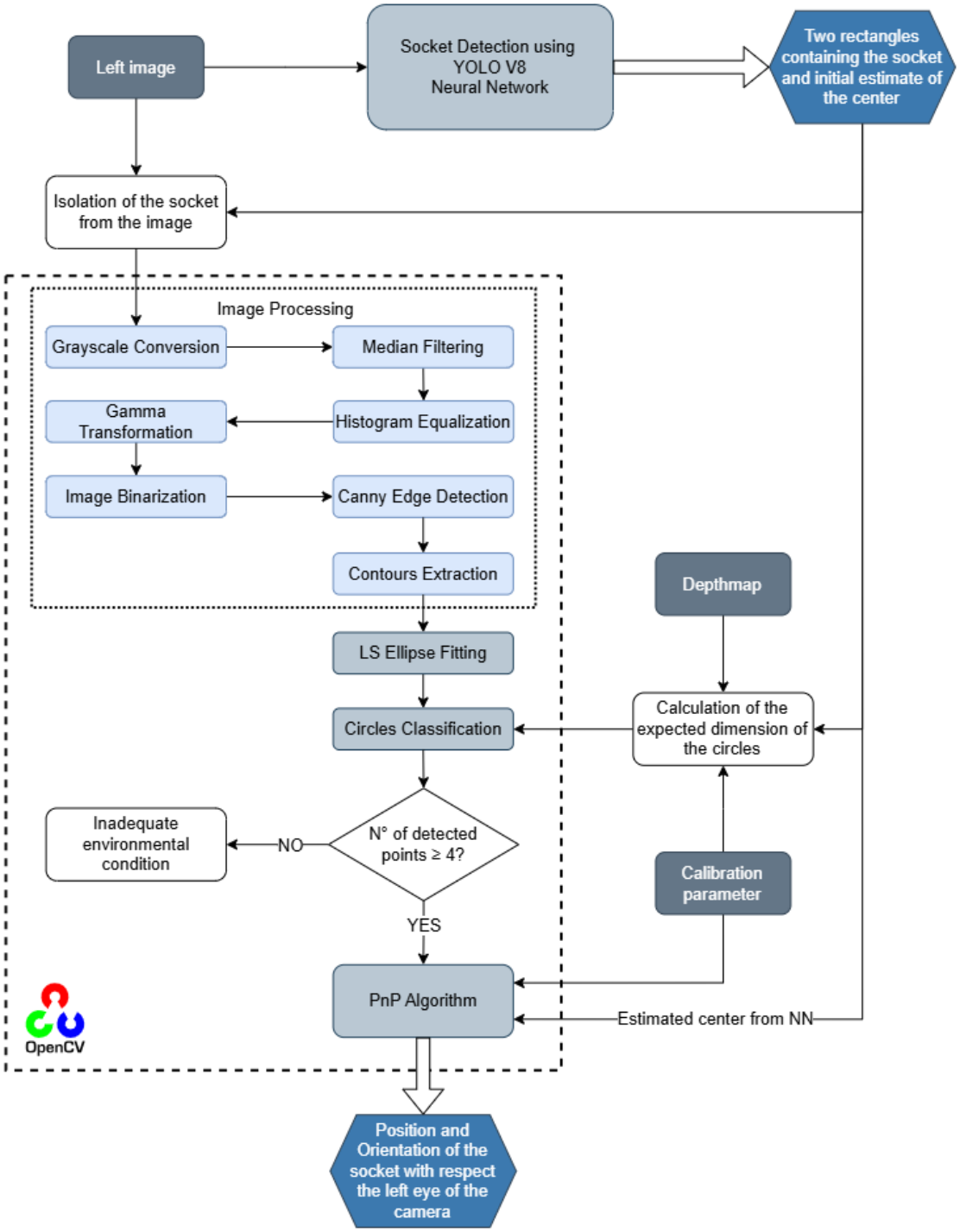


Figure 4.1: Proposed approach

4.2 The Neural Network

The first step of the detection strategy is implemented using a Neural Network (NN). The implementation and training of the NN has already been done, as explained in section 1.2. However, for completeness, a general explanation of the NN used and its generated output will be provided in this section.

4.2.1 YOLOv8

YOLO (You Only Look Once) is a popular object detection and image segmentation NN model developed by *Joseph Redmon* and *Ali Farhadi* at the University of Washington [10]. Traditional methods often relied on sliding-window approaches, which were computationally expensive and slow. *YOLO* revolutionized the field by treating object detection as a single regression problem. Instead of sliding windows, *YOLO* predicts bounding boxes and class probabilities for objects directly from the input image in a single forward pass, making it significantly faster.

YOLOv8 is the latest version in the *YOLO* series of real-time object detectors, provided by the GitHub repository *Ultralytics*. The *YOLOv8* architecture can be divided into three main components:

- **Backbone**: It is a CNN responsible for extracting features from the input image. The architecture consists of 53 convolutional layers and uses cross-stage partial connections to improve the information flow between different layers.
- **Neck**: Combines feature maps from various stages of the backbone to capture multi-scale information. This module integrates high-level semantic features with low-level spatial details to improve accuracy.
- **Head**: Handles the final predictions. *YOLOv8* includes multiple detection modules that predict bounding boxes, objectness scores, and class probabilities for each cell on the feature map. These predictions are then aggregated to form the final detection.

YOLOv8 offers several advantages over its predecessors:

- *Real-time Performance*: The model delivers fast inference speeds, making it ideal for applications that require real-time processing, such as robotics and autonomous vehicles.

- *Enhanced Accuracy*: The model achieves cutting-edge accuracy on various object detection benchmarks.

- *Resource Efficiency*: The model is designed to be lightweight and needs fewer computational resources compared to other models.

For these reasons, previous researchers chose this type of model NN for the project.

4.2.2 The NN Inference

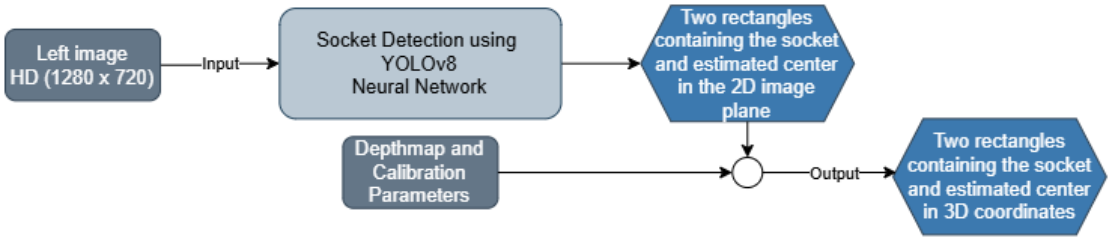


Figure 4.2: Input and output of the detection step with the NN

Once the model is trained, it can be used to predict outcomes from new data. This is known as the inference phase. Figure 4.2 illustrates in detail the input and output of the network. The implemented NN takes as input an HD color image with dimensions 1280×720. This image is the left image from the *ZED Mini* Stereo Camera. As output, it returns the coordinates and centers of two rectangles that enclose the socket, as shown in Figure 4.3. Using the information from the depth map and the calibration parameters of the camera, it is then possible to express the coordinates of the estimated center of the socket in the camera's spatial coordinates, specifically the x, y, and z positions of the center with respect to the left eye of the camera.

Occasionally, the network returns only one of the rectangles due to poor image conditions, particularly in terms of lighting. In other cases, the neural network might not fully recognize the socket. This typically occurs under extreme lighting conditions, such as in complete darkness or when excessive light is directed at the plug. As explained in the Introduction (chapter 1), for the part of the project that is the focus of this thesis, one of the assumptions is that the neural network functions correctly.

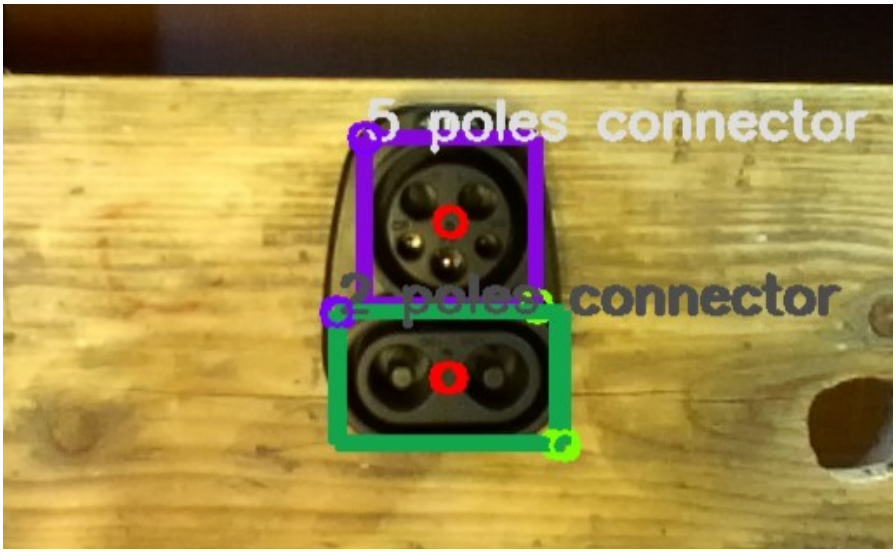


Figure 4.3: Generated output of the Neural Network

4.3 OpenCV

Before delving into the explanation of the implemented technique, it is useful to introduce the library used for certain parts of the algorithm, especially for image processing stages. *OpenCV* is a powerful open source library that offers a wide range of algorithms and tools for image and video manipulation, making it one of the most commonly used libraries in the field of computer vision. Its key features are outlined below:

- *Feature Detection and Description*: *OpenCV* encompasses various methods for detecting and describing features in images, which are essential for tasks such as object recognition, tracking, and matching. This feature is particularly useful for applications that require the precise localization of objects within images.
- *Cross-Platform Support*: *OpenCV* is available in C++, Python, and Java, making it a versatile library. For this project, Python was chosen as the primary programming language due to its ease of use and strong support for scientific computing and image processing tasks.
- *Real-Time Optimization*: *OpenCV* is highly optimized for real-time applications, supporting a wide range of computer vision tasks as well as the

execution of machine learning models. This makes it ideal for projects where performance and speed are critical, such as real-time socket detection and tracking.

- *Image Processing and Analysis*: The library facilitates the reading, writing, and processing of images. Key tasks include edge detection, image filtering, and morphological operations, which are essential to improve image quality and enhance important features.

- *Object Detection and Tracking*: *OpenCV* provides robust tools for detecting specific objects in both images and videos. Additionally, it can be used to track objects over time, allowing for the analysis of their movement and direction in dynamic environments.

The broad functionality of *OpenCV*, combined with its performance optimizations and ease of integration with machine learning frameworks, made it the ideal choice for the image processing needs of this project.

4.4 Image Processing

In the following paragraph, a detailed explanation of the image processing techniques used for the project will be provided. The goal was to identify specific features of the socket with high precision. It was decided to focus on the circular shapes that make up the socket. Before all the image processing steps, the generated output of the neural network is used to isolate the socket from the rest of the image (that is, the image is cropped to include only the socket). To effectively identify circles, the process begins by modifying the image with some preprocessing stages in order to enhance the characteristics that permit the identification of the circular shapes of the socket. The image processing stages are shown in Figure 4.4, and the code implemented for all the preprocessing steps is provided in Listing A.2 in Appendix A. The contours of the socket are then extracted, followed by applying an ellipse fitting algorithm to better approximate the detected circular regions.

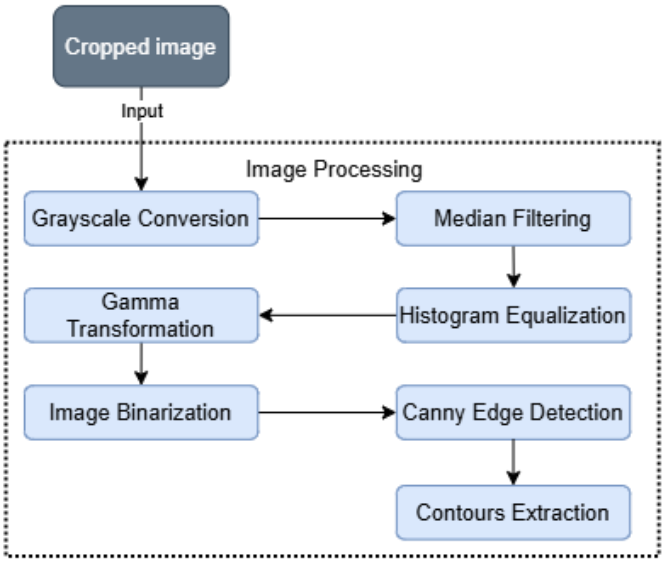


Figure 4.4: Implemented image processing steps

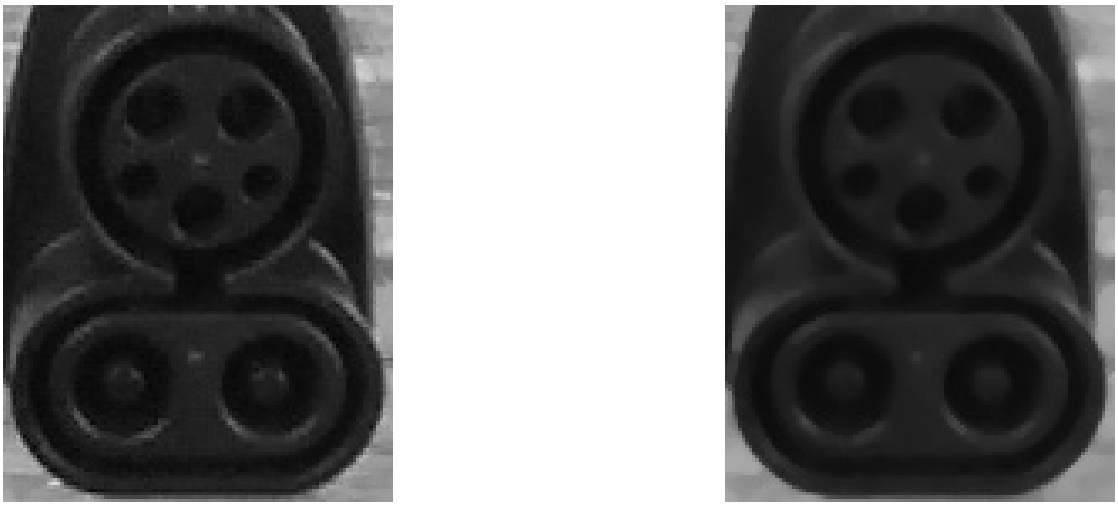
4.4.1 Grayscale Conversion

The grayscale color model is one of the simplest models, representing only the luminance component of the pixel, typically described by a value ranging from 0 (black) to 255 (white). Grayscale images convey less color information than other color models, but require less storage space and computational resources. Consequently, it is a common practice in image processing to initially convert colored images to grayscale images.

The *ZED Mini* stereo camera provides images in the RGB color format. Converting these RGB images to grayscale often involves averaging the color values for each pixel. The Equation 4.1 illustrates this conversion process, where Y represents the resulting grayscale value for the pixel.

$$Y = \frac{R + G + B}{3} \tag{4.1}$$

Figure 4.3 shows the original image retrieved from the left camera of the *ZED*. Meanwhile, Figure 4.5a shows the grayscale image, cropped to focus on the socket, and converted using the respective *OpenCV* function.



(a) Grayscale image **(b)** Median filtered image

Figure 4.5: Grayscale and median filtered images

4.4.2 Median Filter

The median filter is a commonly used non-linear digital filter, especially renowned for its effectiveness in combating salt-and-pepper noise in images. It operates by scanning the image pixel by pixel and replacing each entry with the median value of itself and its neighboring entries. The set of neighboring entries is referred to as a *window* or *kernel*, which moves throughout the image. Therefore, upon selecting a kernel size, the median filter computes the median value of all pixels within the kernel area and substitutes the central element with this computed value. The median filter is a smoothing technique in image processing. This implies its proficiency in reducing noise in smooth areas of a signal. In addition, it exhibits notable effectiveness against "salt-and-pepper" noise. This type of noise, also known as impulsive noise, is characterized by sporadic occurrences of white and black pixels, typically caused by abrupt disturbances in the image signal.

Moreover, unlike traditional filters, such as the mean filter, the median filter is very good at preserving edges and fine details in images. This is because the median operation removes extreme noise values that can blur the boundaries of the image. The median filter's ability to reduce noise selectively helps keep sharp edges, contours, and important features intact, making it great for applications where keeping structural information is important. For these reasons, the median

filter has been chosen as one of the main techniques to apply to the image.

Figure 4.5b shows the socket image after applying a median filter with a kernel size of 5×5, which provides a good balance between noise reduction and detail preservation. With respect to 4.5a, the image is noticeably smoothed, but the details remain intact.

4.4.3 Histogram Equalization

Histogram equalization is a technique used in image processing to enhance the contrast of an image. This method increases the global contrast of images, especially when the usable data of the image are represented by close contrast values. By adjusting the intensities, the values can be more evenly distributed on the histogram, allowing areas of lower local contrast to gain higher contrast.

A significant portion of image processing techniques focuses on gray-level transformations, primarily because enhancing an image often involves directly adjusting pixel intensities. Image enhancement techniques can be summarized by the Equation $s = T[r]$, where T represents a transformation, $r = f(x, y)$ and $s = g(x, y)$ denote the normalized gray levels of the input image f and the output image g at the pixel located at position (x, y).

One of the main challenges in computer vision algorithms is to automatically adapt to changes in the lighting of the scene. Enhancement techniques offer solutions to this problem, with histogram equalization being one of the most important methods.

The **histogram** of a digital image with intensity levels in the range $[0, L-1]$ is a discrete function $h(r_k) = n_k$, where:

- r_k is the kth intensity value.
- n_k is the number of pixels in the image with intensity r_k.

L is the total number of gray levels in the image (256). A normalized histogram is given by:

$$p(r_k) = \frac{n_k}{n} \tag{4.2}$$

where n is the total number of pixels in the image.

A peaked density function corresponds to an image in which luminosity levels are concentrated around a specific value. The goal is to redistribute the levels across the entire available range, making details more distinguishable than in the original image. This operation is executed by remapping the luminosity levels using a function that substitutes one level with another, ensuring that if pixel A is brighter than pixel B in the original image, it will remain brighter in the processed image. This process is known as **histogram equalization**, as shown schematically in Figure 4.6.

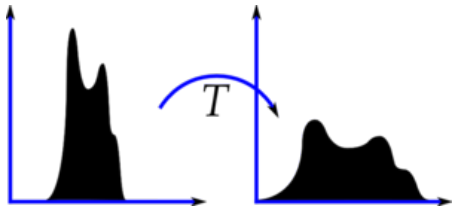


Figure 4.6: Histogram equalization

To derive the transformation function T, the cumulative distribution function (CDF), defined in Equation 4.3, is used.

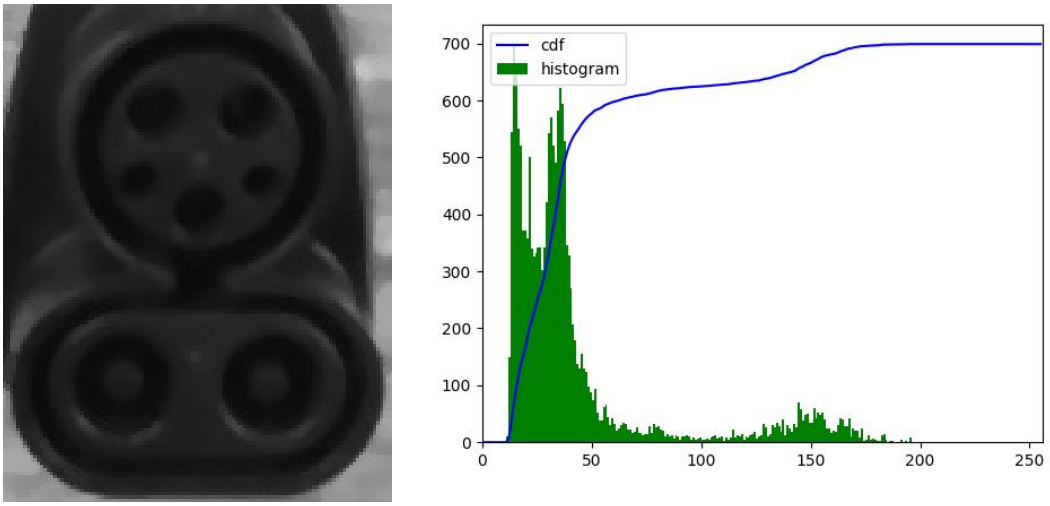
$$cdf(r_k) = \sum_{j=0}^{k} p(r_j) \tag{4.3}$$

The aim is to create a transformation of the form $s = T[r]$ to produce a new image s with a flat histogram, resulting in a linearized CDF throughout the value range. The properties of the CDF allow for such a transformation, defined as follows:

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p(r_j) = (L-1)cdf(r_k) \tag{4.4}$$

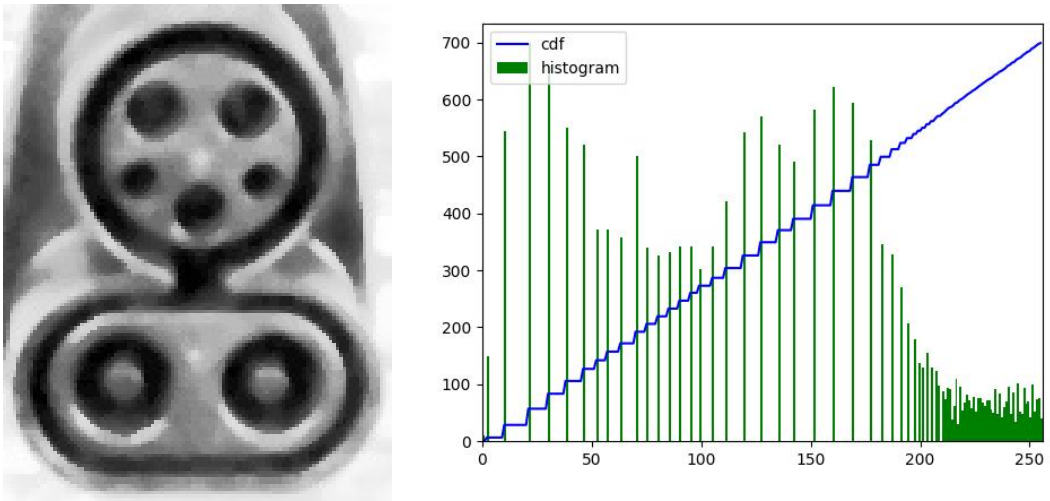
Figures 4.7 and 4.8 show the image before and after histogram equalization, respectively, together with the histogram plots and cumulative distributions. Before applying the transformation for histogram equalization, it is noticeable that the image histogram (Figure 4.7b) is concentrated on the lower pixel values. In other words, the histogram lies in the darker regions. After histogram equalization, the socket appears more visible. The histogram plot (Figure 4.8b) is more evenly

distributed and Cumulative Distribution Function (CDF) is now linearized. This enhanced contrast makes the details of the image more distinguishable, demonstrating the effectiveness of the histogram equalization technique.



(a) Median filtered image **(b)** Corresponding histogram an cumulative histogram

Figure 4.7: Median filtered image and corresponding histogram



(a) Equalized histogram image **(b)** Corresponding histogram and cumulative histogram

Figure 4.8: Equalized histogram image and corresponding histogram

4.4.4 Power-Law (Gamma) Transformation

After the histogram equalization, another enhancement technique is applied. The gamma transform, a common technique, is utilized by *Tadic* [4] for the proposed socket detection algorithm. It is also known as the power-law transformation, mathematically expressed by Equation 4.5, where c and γ are positive constants.

$$s = c \cdot r^{\gamma} \tag{4.5}$$

The parameter γ dictates the shape of the transformation curve that maps the intensity values from the input gray level to the output.

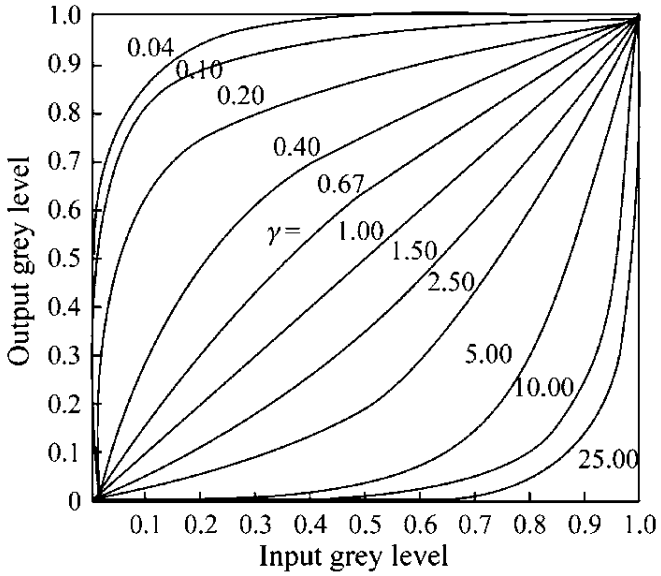


Figure 4.9: Gamma transformation characteristic

Figure 4.9 illustrates the characteristic of the gamma transformation with $c = 1$ and various values of γ. When γ takes on a fractional value, the mapping favors higher output values, effectively widening the range of output values for a narrow range of dark input values. Since the socket is completely black, a fractional γ value helps increase the luminosity of the image, thereby making the contours more visible.

Figure 4.10 shows the socket image after the gamma transformation with a γ value of 0.7. The image does not appear very different from the one before the application

of this transformation, but it was observed that the performance improved slightly with this technique. More information on the selection of the appropriate gamma value will be described in section 4.7, which provides a more detailed explanation of the parameter choice process.

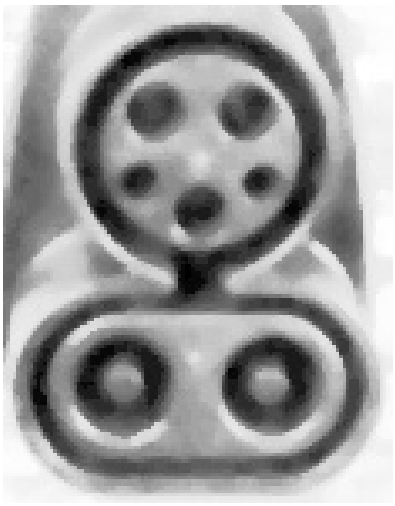


Figure 4.10: Gamma transformed image

4.4.5 Image Binarization

Image binarization, also known as image thresholding, is a technique used to convert a grayscale or RGB image into a binary image. Binarization simplifies an image by representing it in binary form, where each pixel is assigned a value of 0 or 1. The process involves selecting a threshold value. All pixel values below the threshold are set to 0, while those above are set to 1. The choice of threshold is critical and can be determined using various methods. Some techniques are classified as *global thresholding*, where a single threshold value is used for the entire image. In contrast, there is *adaptive thresholding*, where the threshold is determined locally for each pixel based on its neighborhood. Adaptive thresholding performs better in handling varying illumination conditions. For the project, an adaptive binarization algorithm based on the mean has been selected. After specifying a kernel area and a constant c, the threshold value is defined as the mean of the neighborhood area minus the constant c. This approach allows for the use of different thresholds for different regions of the same image.

Figure 4.11 shows the binarized socket image after applying mean thresholding

using the *OpenCV* function `cv2.adaptiveThreshold()` with a kernel size of 13 and a constant c equal to 0. A detailed explanation of the choice of kernel size for the binarization process will be provided in section 4.7.

Figure 4.11: Binarized image

4.4.6 Canny Edge Detection

Canny Edge Detection is a multi-stage edge detection algorithm developed by *John Canny* in 1986 [11]. It is widely used for its ability to detect edges while suppressing noise. In *OpenCV*, the algorithm is implemented through the function `cv2.canny()` [12]. The algorithm's steps are explained below.

1. **Gaussian Smoothing**. This step involves the convolution of the input image $I(x, y)$ with a Gaussian kernel to reduce noise. The Gaussian kernel is defined as in Equation 4.6.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.6}$$

   The smoothed image is then obtained as follows: $I_{\text{smoothed}}(x, y) = I(x, y) * G(x, y, \sigma)$. Inside the function `cv2.canny()` in *OpenCV*, a 5×5 Gaussian filter is applied for this step.

2. **Gradient Calculation**. The smoothed image is then filtered with a *Sobel* kernel used to compute the gradients along the horizontal (G_x) and vertical

(G_y) directions. The matrices used to calculate the two gradients are shown in Equation 4.7.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.7}$$

The gradient magnitude and angle can be calculated as shown in Equation 4.8.

$$Magnitude(G) = \sqrt{G_x^2 + G_y^2} \quad Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{4.8}$$

The gradient direction is always perpendicular to edges.

3. **Non-maximum Suppression**. In this step, the gradient magnitude is thresholded to remove weak edges. For each pixel (x, y), the gradient magnitude is compared with its neighbors along the gradient direction. If the magnitude of the pixel is greater than the magnitudes of its neighbors in the gradient direction, the pixel value is retained; otherwise, it is suppressed (set to zero).

4. **Hysteresis Thresholding**. This final step involves double thresholding. Two thresholds, $maxVal$ and $minVal$, are used to classify pixels as strong, weak, or non-edges. Any pixel with a gradient magnitude greater than $maxVal$ is classified as a strong edge pixel. Any pixel with a magnitude less than $minVal$ is classified as a non-edge pixel. Pixels that lie between the two thresholds are classified as weak edges. Starting from strong edge pixels, weak edge pixels are traced recursively in the neighborhood. If a weak edge pixel is connected to a strong edge pixel, it is classified as part of the edge. This process continues until no more weak edges can be linked to strong edges.

   As shown in Figure 4.12, pixel A is considered a strong edge pixel because it is above the $maxVal$. Pixel C is a weak edge, but since it is connected to edge A, it is also considered a valid edge. On the contrary, pixel B, which is classified as weak, is not connected to any strong edge and is thus discarded. This step is also useful for removing small pixel noises. In the *OpenCV* function `cv2.canny()`, the second and third arguments are the two thresholding values. They have been chosen using a trial and error method.

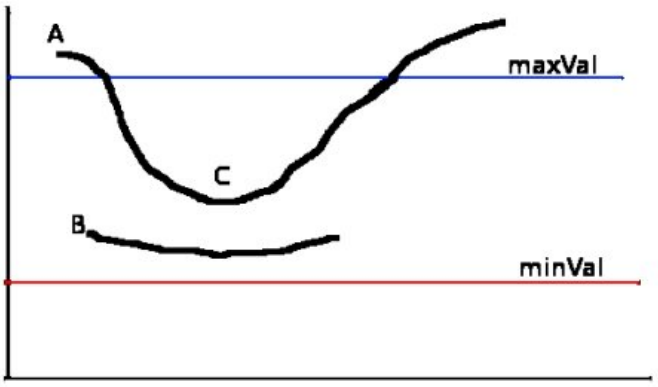


Figure 4.12: Example of hysteresis thresholding in *Canny* algorithm

Figure 4.13 shows the result of applying the *Canny* algorithm to the binarized socket image. At this point, the image contains only the edges. The circles to be fitted are visible, but there are also many distracting elements. The main focus of the following steps is to extract information about useful edges.

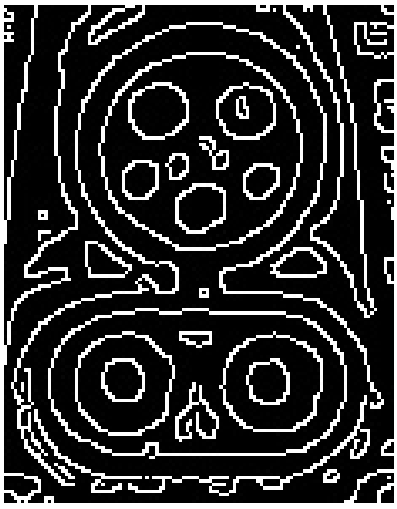


Figure 4.13: Image after *Canny* algorithm

4.4.7 Contours Extraction

After applying the edge detection algorithm, the next step was to extract the contours that delineate the detected edges. These contours are essential for identifying shapes, such as ellipses, in subsequent processing stages. In computer vision, a **contour** is defined as a curve that joins all continuous points along a boundary that share the same intensity or color. The *OpenCV* library provides a widely used function, `cv2.findContours()`, to extract these contours from binary images. The function returns a list of contours, where each contour is represented as a sequence of 2D coordinates corresponding to points on the boundary of an object.

The underlying algorithm implemented in `cv2.findContours()` is based on the work of *Suzuki et al.* [13]. This method focuses on identifying object contours within a binary image using a technique known as **border following**. The algorithm scans the image, pixel by pixel, in search of transitions between foreground (object) and background pixels, which signify the beginning of a contour. Upon detecting a transition, the algorithm starts following the boundary of the object by iterating through the neighboring pixels, capturing the contour structure in a clockwise or counterclockwise manner.

This algorithm is designed to efficiently extract contours by focusing solely on boundary pixels, avoiding redundant operations on pixels located inside the object. This makes the method computationally efficient, an important consideration for real-time computer vision applications. Furthermore, it preserves the **topological structure** of the image, which means that it can handle complex objects with nested or overlapping contours without altering the spatial relationships between different contour components.

The function `cv2.findContours()` requires a binary image as input and offers several configuration parameters:

- The *contour retrieval mode*, which defines how contours are organized and stored (e.g., hierarchical relationships between contours can be retrieved using `cv2.RETR_TREE`, or all contours can be retrieved without hierarchy using `cv2.RETR_LIST`).
- The *contour approximation method*, which specifies the precision with which the contour points are approximated. For example, `cv2.CHAIN_APPROX_SIMPLE` compresses horizontal, vertical, and diagonal segments and leaves only their endpoints, thereby reducing the number of points stored, while

`cv2.CHAIN_APPROX_NONE` retains all contour points.

In the case of the implemented algorithm, the contours are retrieved without hierarchy using `cv2.RETR_LIST`, which ensures that all detected contours are returned as a flat list. Furthermore, the contour approximation method used is `cv2.CHAIN_APPROX_NONE`, meaning that all contour points are retained without any compression. This approach is advantageous because it provides greater precision in subsequent processing steps, particularly in ellipse fitting, where the retention of all boundary points allows a more accurate approximation.

4.5 Circle Fitting and Classification

In this section, a comprehensive analysis of the ellipse fitting method and its classification will be carried out. Having obtained a list of all the contours in the image through the previous steps, the subsequent task was to determine if it was possible to fit each contour with an ellipse. The decision to seek ellipses is based on the fact that images can be captured from various angles. Since a circle viewed from different angles appears as an ellipse and high precision is required, an ellipse detection method was preferable. Subsequently, focusing on the known geometry of the socket, the objective was to accurately identify the ellipses. It was decided to focus on 10 circles on the socket, as depicted in Figure 4.14. The commented code for this part of the algorithm is provided in Listing A.3.

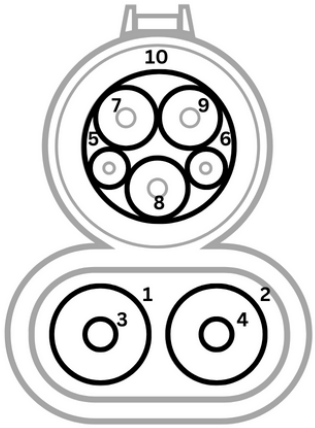


Figure 4.14: CCS Type 1 Socket, labeled circles

4.5.1 Least Squares Fitting

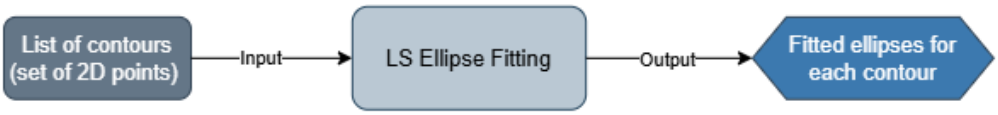


Figure 4.15: Input and output of Least Squares (LS) Ellipse Fitting

The chosen method for fitting ellipses employs a LS approach, based on the algorithm devised by *R. Hal oy and J. Flusser* [14]. An **ellipse** can be described by an implicit second-order polynomial equation along with a specific constraint, as shown in Equation 4.9.

$$ax^2 + bxy + cy^2 + dx + ey + f = 0, \quad b^2 - 4ac < 0 \tag{4.9}$$

By introducing the parameter vector $\theta = [a \quad b \quad c \quad d \quad e \quad f]^T$, the conic equation can be represented in matrix form as:

$$\begin{bmatrix} x^2 & xy & y^2 & x & y & 1 \end{bmatrix} \cdot \theta = 0 \tag{4.10}$$

The conic equation describes the algebraic distance of a point (x, y) to the conic, which is zero for points on the conic. Thus, fitting a general conic to a set of points $(x_i, y_i), i = 1...N$ involves minimizing the sum of squared algebraic distances. This is achieved by building a *design matrix* D, where each row corresponds to the quadratic and linear terms of an input point.

$$\mathbf{D} = \begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^2 & x_ny_n & y_n^2 & x_n & y_n & 1 \end{bmatrix} \tag{4.11}$$

As proposed by *Fitzgibbon* [15], it is essential to introduce a constraint to control the sign of the conic discriminant. The discriminant must be strictly negative to ensure that the conic is an ellipse. To fix the scale and maintain consistency, the inequality constraint given by the discriminant can be transformed into the equality constraint $b^2 - 4ac = -1$. Therefore, after defining the *constraint matrix*

C as shown in Equation 4.12, the ellipse fitting problem can be reformulated as demonstrated in Equation 4.13.

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.12}$$

$$\min_{\theta} \|D \cdot \theta\|^2 \quad \text{s.t.} \quad \theta^T C \theta = 1 \tag{4.13}$$

Introducing the *Lagrange* multiplier λ and differentiating to find the minimum (Equation 4.14), Equation 4.15 is derived, where λ is the eigenvector and θ is the eigenvalue.

$$\nabla_\theta \mathcal{L} = 2D^T D\theta - \lambda \cdot 2C\theta = 0 \tag{4.14}$$

$$\theta = \lambda \cdot (S)^{-1} C\theta \tag{4.15}$$

The matrix $S = D^T D$ is also known as the *scatter matrix*. It is important to note that the objective function of the minimization equals the Lagrange multiplier: $\|D\theta\|^2 = \lambda$. Therefore, the optimal conic parameters θ are obtained by taking the eigenvector corresponding to the smallest strictly positive eigenvalue. The original approach proposed by *Fitzgibbon* has some drawbacks and can produce suboptimal or incorrect results. *Halir* and *Flusser* simplified the structure of the various matrices. The design matrix D is decomposed into its quadratic and linear parts, $D = [D_1, D_2]$, where

$$\mathbf{D_1} = \begin{bmatrix} x_1^2 & x_1 y_1 & y_1^2 \\ x_2^2 & x_2 y_2 & y_2^2 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n y_n & y_n^2 \end{bmatrix} \qquad \mathbf{D_2} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \tag{4.16}$$

The scatter matrix S is divided as shown in Equation 4.17, where $S_1 = D_1^T D_1$,

$S_2 = D_1^T D_2$, and $S_3 = D_2^T D_2$.

$$\mathbf{S} = \begin{bmatrix} \mathbf{S_1} & \mathbf{S_2} \\ \mathbf{S_2^T} & \mathbf{S_3} \end{bmatrix} \tag{4.17}$$

Similarly, the constraint matrix C is decomposed as shown in Equation 4.18.

$$\mathbf{C} = \begin{bmatrix} \mathbf{C_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{4.18}$$

Finally, the vector of coefficients θ is divided into $\theta = [\theta_1 \quad \theta_2]^T$, where $\theta_1 = [a \quad b \quad c]^T$ and $\theta_2 = [d \quad e \quad f]^T$. With this alternative formulation, the algorithm can be expressed as the set of Equations 4.19, where $M = C_1^{-1}(S_1 - S_2 S_3^{-1} S_2^T)$ is the *reduced scatter matrix*.

$$\begin{cases} M\theta_1 = \lambda\theta_1 \\ \theta_1^T C_1 \theta_1 = 1 \\ \theta_2 = -S_3^{-1} S_2^T \theta_1 \\ \theta = [\theta_1 \quad \theta_2]^T \end{cases} \tag{4.19}$$

The algorithm is computationally unambiguous and can be implemented in a numerically stable manner. Therefore, it was chosen for the detection of ellipses in the project. The implemented function in Python is provided in Listing A.4. Figure 4.15 schematically illustrates the main inputs and outputs of this fitting step.

4.5.2 Circle Classification

After fitting the ellipses, the next step involves classifying it to determine whether it corresponds to one of the circles shown in Figure 4.14 or if it is an outlier. To perform an accurate classification, the known geometry of the socket was utilized, along with the segmentation output generated by the neural network. The schematic representation of the inputs and outputs for this step is illustrated in Figure 4.16.

Since the neural network produces two rectangles that isolate the socket from the

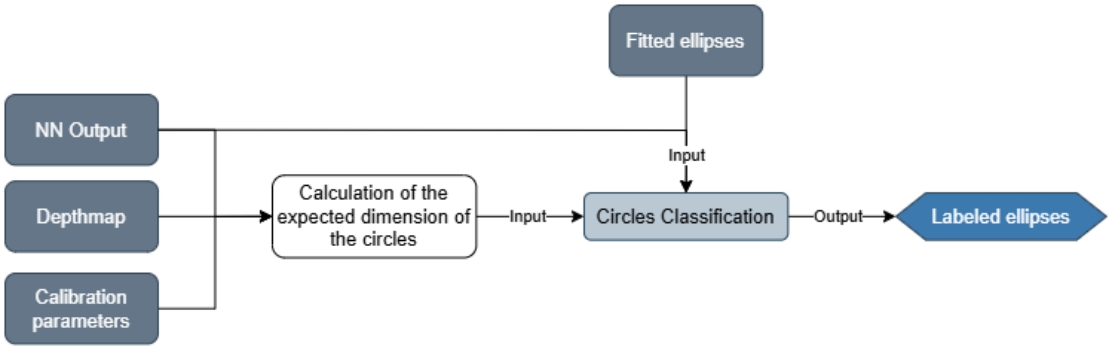


Figure 4.16: Input and output of the classification stage

rest of the image, it was decided to utilize the depth map to compute the mean value of the depth within the region containing the socket. Knowing the precise values of the different circles' radii, it becomes possible to calculate the expected pixel dimensions of the circle radii in the image. This is achieved by evaluating the average depth and applying the relationships presented in the pinhole camera model, which establish a correlation between the position of a point in the real world and its position in the image plane using focal length and depth. In this manner, it becomes feasible to establish certain constraints for the dimensions of the circle radii. Since the previous algorithms identify ellipses, a mean radius for the ellipses has been defined as shown in Equation 4.20, where a and b, respectively, represent the lengths of the major and minor axes of the fitted ellipses.

$$r_{mean} = \frac{a+b}{2} \tag{4.20}$$

This approach allows for the classification of ellipses based on their different mean radii dimensions. By understanding the expected dimensions of the radii and the reciprocal proportion of one ellipse's radius to another, intervals can be defined. Ellipses belonging to a particular interval are classified on the basis of four different dimensions.

Since there are multiple ellipses with the same dimensions, apart from the largest external one, an additional constraint is required to correctly identify the class. A positional constraint has been selected based on the output of the neural network. The two rectangles provided by the neural network have been subdivided into smaller rectangles, as illustrated in Figure 4.17. These rectangles are utilized to unequivocally identify the ellipses.

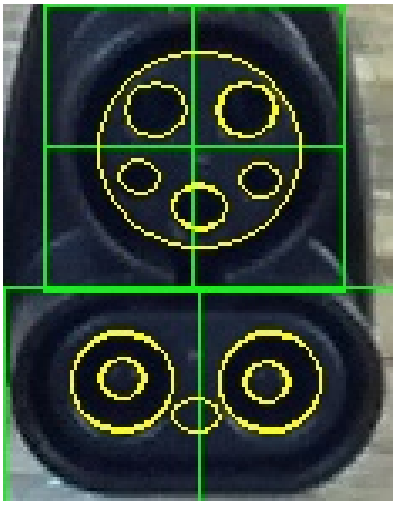


Figure 4.17: Image with fitted circles and sub-rectangles

Figure 4.18 shows the socket image with all detected ellipses, labeled by numbers. This is the final output of the image processing and ellipse detection steps implemented in the project. The result is taken from one of the images in which the algorithm performs the best. The general results of the algorithm implemented will be discussed in chapter 5.

Figure 4.18: Images with extracted ellipses

4.6 Socket Pose Estimation

The final step to achieve the project goal is to estimate the pose of the charging port, specifically its position and orientation. The following section will provide a detailed explanation of the selected algorithm for determining the pose of the socket: the *PnP* algorithm.

4.6.1 PnP Algorithm

The *Perspective-n-Point (PnP)* algorithm is used in computer vision to estimate the position and orientation of an object in the camera's coordinate system, given a set of object points, their corresponding image projections, and the camera's intrinsic matrix and distortion coefficients.

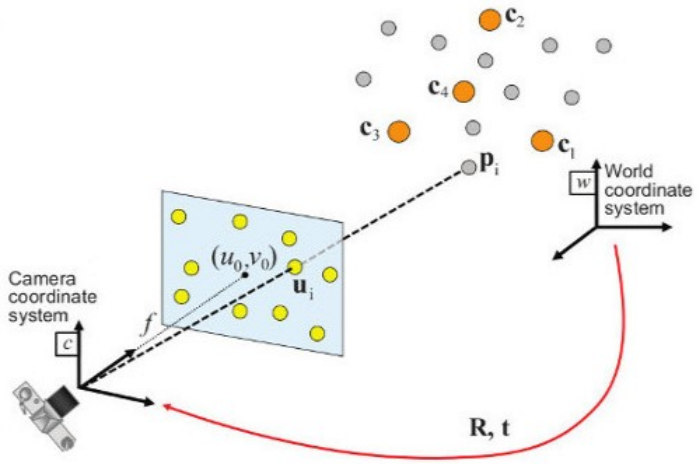


Figure 4.19: *PnP* algorithm visual scheme

Input Data and Objective

As shown in Figure 4.19, the *PnP* algorithm requires the following input data:

- A set of 3D points $\mathbf{c}_i = (X_i, Y_i, Z_i)$ representing specific points on the object with respect to a defined world coordinate system.

- The corresponding 2D points $\mathbf{u}_i = (u_i, v_i)$ in the image plane.

In addition, intrinsic camera parameters are needed, typically represented by the camera matrix K. The Python code used to extract these parameters is presented and discussed in section 3.2. The relationship between a 3D point $\mathbf{c}_i$ on the object and its 2D projection $\mathbf{u}_i$ on the image is represented by Equation 4.21, which has already been discussed in chapter 3.

$$\mathbf{u}_i = K[R|t]\mathbf{c}_i \tag{4.21}$$

R represents the rotation matrix and t is the translation vector. The goal of the algorithm is to find R and t that best satisfy Equation 4.21 for all given point correspondences, thus determining the pose of the object.

The minimum number of correspondent points required to solve the *PnP* algorithm depends on the number of unknowns. The pose of the object involves six degrees of freedom, comprising three for translation and three for rotation. Each point i in the image provides two constraints represented by u_i and v_i. To solve for these six unknowns, theoretically, at least three points are necessary. However, with only three points, the system provides exactly six equations, resulting in a determined system with no redundancy, which is impractical for real-world scenarios affected by noise.

For general *PnP* problems involving non-coplanar points, a minimum of four points is required. This provides eight equations, offering some redundancy that helps in managing noise and errors. In the case of planar objects, such as the socket, the points lie on a single plane. This can lead to underconstrained scenarios in some configurations. Nevertheless, the mathematical requirement of at least four points still applies to ensure solvability of the equations. In practice, using at least six points is commonly recommended as a practical minimum. This number of points provides sufficient redundancy to handle noise and mitigates issues related to planar degeneracies.

In the implemented project, the chosen world coordinate system is defined as the object's local coordinate system, as shown in Figure 4.20. The 3D points used for the algorithm, as well as the corresponding 2D points in the image plane, are the centers of the circles, as shown in Figure 4.20. Therefore, it is possible to retrieve correspondences for a maximum of **eight points**.

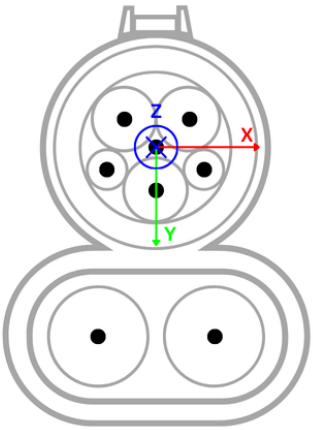


Figure 4.20: *CCS* Local Coordinate System and Centers

Initialization Step

The *PnP* algorithm begins with the initialization step, where an initial guess is made for the rotation matrix R an the translation vector t. One common method to obtain this initial guess is Direct Linear Transform (DLT). This method linearizes the problem by ignoring the non-linear nature of the camera projection model. The DLT method works by solving a system of linear equations, providing an initial estimate for the transformation parameters.

Sometimes, instead of using the DLT method, it is possible to directly use the estimated values for R and t obtained from previous steps. In the case of the project presented, it is a good idea to use the translation vector t estimated directly by NN. In this way, a more accurate starting point for the optimization process is provided. Using these estimates can, moreover, reduce the number of iterations needed for the optimization to converge and improve the overall accuracy of the final pose estimation.

Optimization Step

Once an initial estimate for R and t is obtained, the next step is to refine the values using iterative optimization techniques. The goal of this optimization is to minimize the reprojection error, which measures how well the projected 3D points align with the observed 2D points in the image. The reprojection error is defined in Equation 4.22, where $\mathbf{u}_i$ are the observed 2D points and $\hat{\mathbf{u}}_i$ are the projected

3D points using the current estimates of R and t. The projected points $\hat{\mathbf{u}}_i$ are calculated using Equation 4.21.

$$\text{Reprojection Error} = \sum_i \|\mathbf{u}_i - \hat{\mathbf{u}}_i\|^2 \tag{4.22}$$

To achieve this, non-linear optimization techniques are commonly used. The *Levenberg-Marquardt* algorithm combines the advantages of the *Gauss-Newton* algorithm and the gradient descent method, which makes it well suited to solve non-linear LS problems [16]. During each iteration, the algorithm adjusts R and t to reduce the reprojection error. Adjustments are made based on the Jacobian matrix, which contains partial derivatives of the reprojection error with respect to R and t. The iterative optimization process continues until the change in the reprojection error is below a predefined threshold or the maximum number of iterations is reached.

Figure 4.21 schematically illustrates the main steps, inputs, and outputs of the algorithm implemented. The corresponding Python code is provided in Listing A.5.

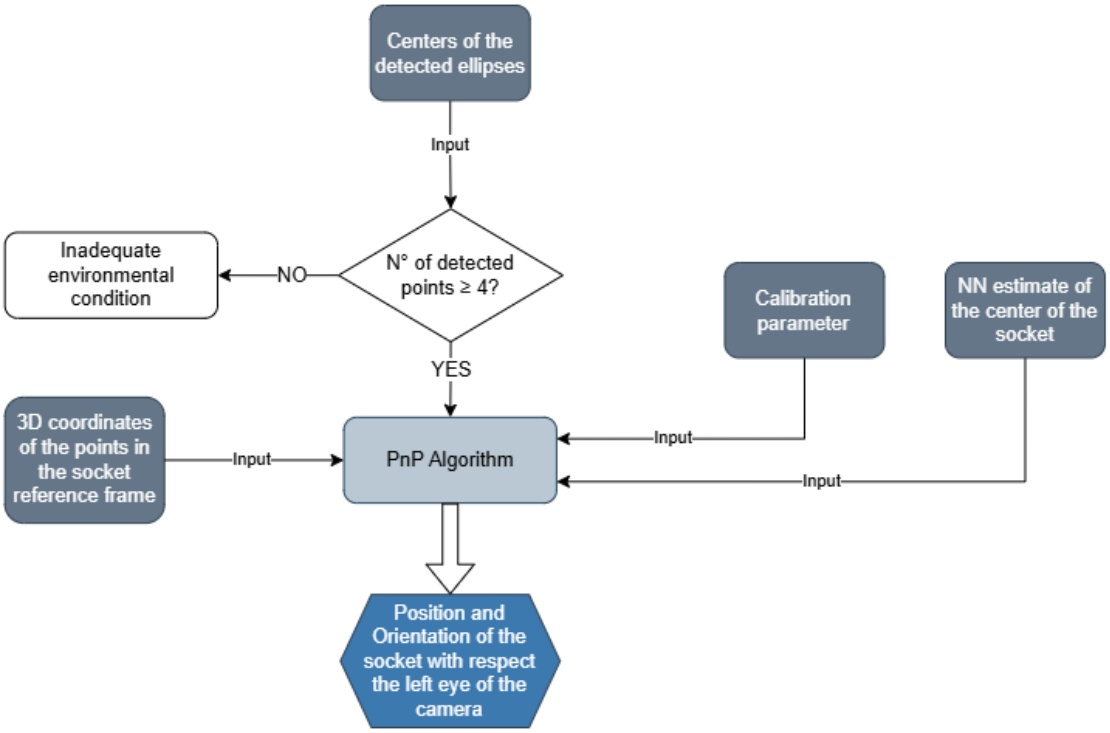


Figure 4.21: Schematic representation of the *PnP* implementation

4.7 Parameter Tuning

In this section, an overall discussion of the techniques used for choosing the parameters at various stages of image processing will be presented. The specific values to be decided are:

- The kernel size for the median filter
- The gamma value for the power-law transformation
- The kernel size for adaptive threshold binarization
- The two thresholds, *maxVal* and *minVal*, for the *Canny* algorithm

The chosen kernel size for the median filter is 5, and a detailed explanation of this selection has already been provided in the respective section. For the remaining three parameters, further discussion is required.

The most critical parameters for the successful outcome of the image processing steps were observed to be the gamma value and the block size for binarization. Therefore, a precise parameter adjustment was performed for these two values. For the two *Canny* thresholds, common values were used.

To apply a parameter-tuning approach, an evaluation metric was needed. As explained in the previous section, the PnP algorithm theoretically requires at least four detected points in an image to function properly. Therefore, the number of images with at least four detected points was chosen as the evaluation metric. However, since the algorithm works more reliably with six points in practice, the number of images with at least six detected points was also used as an additional evaluation metric.

A set of 240 images was created, for which the neural network generates valid output, under different environmental conditions. This set was divided into a *training set* for parameter tuning, which contains half of the images, and a *validation set* for subsequent testing, composed of the other half.

First, the number of training images with at least four points was evaluated. A grid search approach was used to exhaustively search for gamma and block size values that yield the best evaluation metric. For gamma, values ranging from 0.1 to 1 in steps of 0.1 were evaluated. For the block size used in binarization, odd values from 5 to 21 were considered.

Figure 4.22 shows the table of the percentage of images with at least four detected points across different gamma values and block sizes. The goal of the parameter-tuning process is to maximize the percentage of valid images. Therefore, by analyzing the table in Figure 4.22, values greater than 93% can be considered a good result. This means that the *PnP* algorithm can be launched 93% of the time. Using this threshold, it is possible to discard gamma values of 0.1 and 0.2 and block size values of 5, 19, and 21.

Block Size \ Gamma	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
5	61.8%	71.5%	81.3%	82.1%	84.6%	82.9%	88.6%	90.2%	88.6%	90.2%
7	75.6%	87.0%	86.2%	91.1%	93.5%	92.7%	92.7%	95.9%	94.3%	92.7%
9	77.2%	87.0%	89.4%	94.3%	93.5%	92.7%	94.3%	95.1%	95.1%	92.7%
11	78.0%	84.6%	90.2%	91.9%	91.9%	91.9%	91.9%	91.9%	92.7%	92.7%
13	78.0%	87.0%	91.9%	93.5%	92.7%	95.1%	94.3%	95.1%	92.7%	92.7%
15	82.9%	89.4%	93.5%	93.5%	93.5%	94.3%	95.1%	93.5%	91.9%	92.7%
17	82.9%	87.8%	92.7%	92.7%	92.7%	93.5%	93.5%	92.7%	91.9%	93.5%
19	82.1%	89.4%	91.9%	92.7%	91.1%	91.9%	91.1%	92.7%	91.1%	89.4%
21	83.7%	88.6%	91.9%	91.9%	91.9%	91.9%	91.1%	88.6%	87.8%	85.4%

Figure 4.22: Parameter tuning considering at least 4 detected points

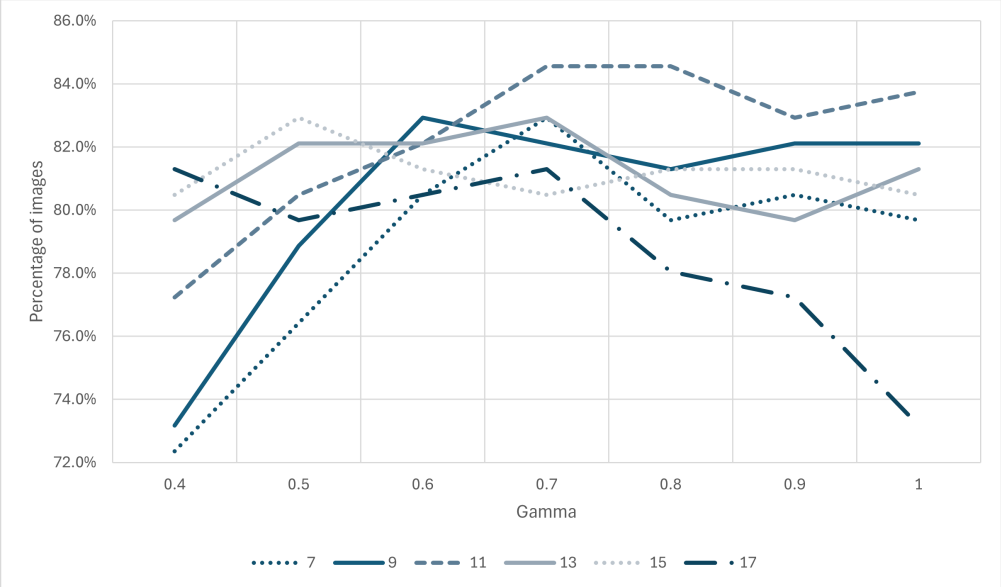


Figure 4.23: Parameter tuning considering at least 6 detected points

Subsequently, a detailed analysis of the algorithm's performance considered the number of images with at least six detected points. This analysis focused only on

block sizes and gamma values that showed high results in the previous evaluation. Figure 4.23 illustrates curves of the percentage of images with at least six detected points across gamma values, with each line representing different block sizes.

Using this evaluation metric, the maximum of 84.6% is achieved with combinations (11, 0.7) and (11, 0.8). However, in these cases, the value of the metric considered before is below 93%. Consolidating the findings of both evaluations, the chosen values are gamma = 0.7 and block size = 13. This combination closely approaches (82.9%) the maximum percentage of images in the latter consideration and maintains a valid image percentage for the *PnP* algorithm above 93%. Further evaluation of the performance will be done in the next chapter using the validation set of images.

Chapter 5

Test and Results

In this chapter, the experiments and their results are explained in detail. As mentioned in the Introduction (chapter 1), the goal of the project is to accurately determine the pose of the socket. One of the prerequisites is to use the existing neural network to preliminarily detect the socket. Therefore, for the experimental evaluation of the proposed approach, only the conditions where the neural network provides a valid output are examined.

5.1 Circle detection

In this section, the circle detection algorithm is evaluated, which includes all the image processing techniques implemented. The goal of the algorithm is to detect 10 circles within the socket, as explained in the previous chapter.

A key factor in developing the image processing algorithm is obtaining high-quality input images. In addition to using a high-quality capturing device (the *ZED Mini* Camera), another main challenge is to ensure the validity of the algorithm across various scenarios. Therefore, the images were taken in different environmental settings. Some images were captured with artificial illumination Light Emitting Diode (LED) at different times of the day, with varying levels of natural light, but with the same artificial lights positioned around the setup.

To further diversify the image set, additional images were taken in a different range of artificial lighting scenarios, including some without any lights, with varying

natural illumination, and even in darker environments. Since the number of images taken for each scenario is insufficient for generalization, all these images were grouped together, forming a set of 240 images.

This set was then divided into two subsets: 120 images for parameter *fitting*, and the remaining 120 for *validation*. To evaluate the performance of the algorithm, the validation set was used and the number of centers detected was evaluated, with a maximum of eight.

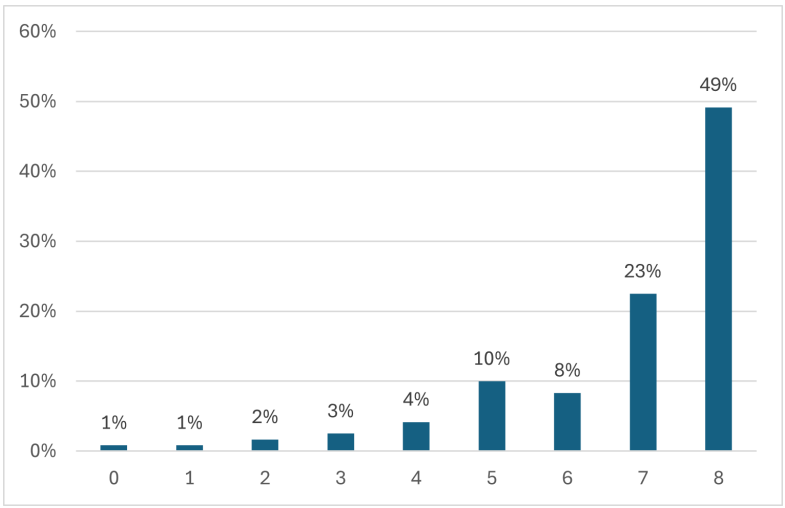


Figure 5.1: Number of detected centers for the validation set of images

Figure 5.1 shows the percentage of images, out of the set of 120, for which a fixed number of centers were detected. In 49% of the cases, all possible centers were detected. Considering that the practical number of centers for the *PnP* algorithm to work well is 6, as explained in the previous section, the percentage of images with 6 or more detected centers is 80%. Furthermore, if a threshold of at least 4 points is considered for the *PnP* algorithm to give a result, the percentage reaches 94%. These values are slightly lower, especially considering the evaluation metrics of at least six points, compared to the performance obtained during the parameter fitting phase, but the values are still appreciable.

If only the subset of images taken under adequate LED illumination is considered, consisting of 93 images, the performance is much better, as shown in Figure 5.2. In this case, the percentage of images with at least 6 detected points is 91%, and the percentage of images for which the *PnP* algorithm can be implemented is 98%. This means that in 98% of cases, in adequately illuminated environments, the

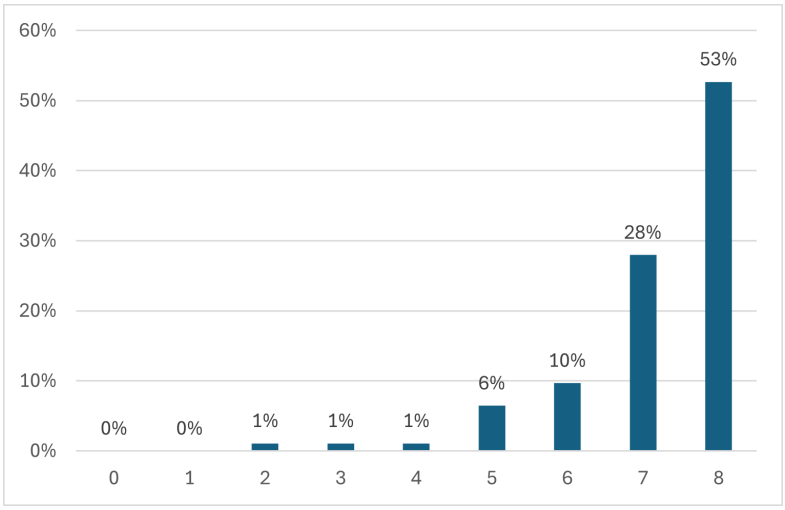


Figure 5.2: Number of detected centers in an adequately illuminated environment

algorithm to retrieve the pose can work, even if some degeneracies are encountered. However, to be more confident in the outcome of the algorithm, the percentage decreases to 91%, which is still a good result.

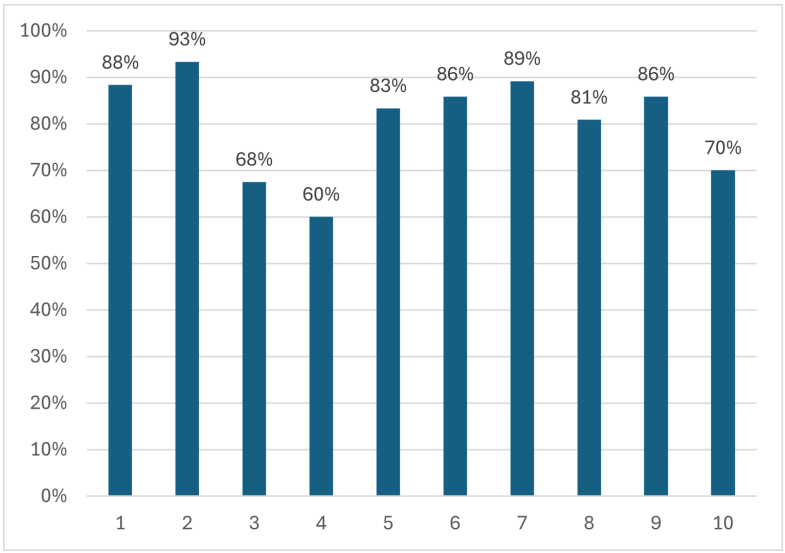


Figure 5.3: Percentage of detection for each circle

Another useful parameter to analyze is which circles are detected most frequently. Figure 5.3 examines this aspect, showing the detection percentage for each circle. The numbers on the horizontal axis correspond to those in Figure 4.14. The image

set used for this analysis is the entire validation set, consisting of 120 images. The average detection percentage for circles is 80%. Circles 1, 2, 7, and 9 have the highest scores. The largest circle (number 10) has one of the lowest scores, along with circles 3 and 4, which are the smallest. The dimension ranges used to classify each circle might not always be accurate for the smallest and largest circles, as these ranges depend on the average distance calculated from the depth map of the stereo camera. Another reason could be that the image processing stages cannot effectively extract the contours of the smallest and largest circles. However, these values do not deviate significantly from the average, with a standard deviation of 11%.

5.2 Pose Estimation

This section evaluates the results of the final step of the project, which involves the *PnP* algorithm. The primary objective of this thesis is to accurately determine the position and orientation of the socket. To achieve this, the results produced by the algorithm are compared to the robot's recorded pose at the moment each photo was captured. By comparing these values, errors are calculated and analyzed.

5.2.1 Position

To evaluate the error in position, 18 different starting positions of the camera were verified. Knowing the position of the base of the robot with respect to the socket and the position of the camera with respect to the base of the robot, it is possible to calculate the actual position of the socket with respect to the camera using a simple multiplication of transformation matrices. Hence, the algorithm was tested for each starting position and the results were evaluated. Table 5.1 reports the evaluated position error for each axis.

Axis	**Error (mm)**
x	2.95
y	2.47
z	2.07

Table 5.1: Position errors

The reference system used is the TCP reference system, as reported in Figure 3.2,

where the x axis points towards the plug, the y axis points to the right, and the z axis points downward. The error is evaluated as the standard deviation of the difference between the measured position and the one returned by the algorithm. The mean represents a bias given by a fixed error in the measurement of the starting position. The greatest error is observed in the x direction, which represents the depth. The values in the other directions are slightly better.

5.2.2 Orientation

To evaluate the orientation error, images were taken from different starting angles of the robot's end effector. The base of the robot was positioned such that it was rotationally aligned with the socket. This setup ensures that, given a defined angle to the camera, the expected solution of the algorithm should be the opposite value of the one given for the respective rotational axis. During the evaluation, it was observed that the algorithm sometimes produces very inaccurate values due to degeneracies that arise from detecting only 4 or 5 points. Such cases were excluded from the algorithm performance evaluation.

A set of 69 valid images, taken from various angles and under different environmental conditions, was used for the evaluation. The errors were calculated as the standard deviation of the difference between the solution provided by the *PnP* algorithm and the opposite angles of the camera's initial position. The results are reported in Table 5.2. The roll axis showed the best precision, while the other axes had slightly higher errors but still remained below 2 degrees. This level of precision can be considered a good result, consistent with those obtained in previous works analyzed in the State of the Art (chapter 2).

Direction	**Error (°)**
Roll (R_x)	0.68
Pitch (R_y)	1.98
Yaw (R_z)	1.68

Table 5.2: Orientation errors

Overall, the results obtained are satisfactory for the requirement of achieving precise alignment. The final step of plug insertion will be performed using a force control algorithm that accounts for the compliance of the robotic arm, allowing it to compensate for small alignment errors.

Chapter 6

Conclusion

This thesis presents a comprehensive study on the development and implementation of a robotic system for automatic EV charging. The system integrates a stereo camera with deep learning techniques to accurately detect and estimate the pose of the *CCS Type 1* charging socket. The adoption of *YOLOv8* for object detection, together with a *PnP* algorithm for pose estimation, has proven to be effective in providing reliable and accurate results.

The primary goal of this research was to develop a system capable of detecting the position and orientation of an EV charging socket, which allows autonomous connection of a charging plug. The key findings of this work demonstrate the feasibility and reliability of such a system. Through the integration of a stereo camera and a pre-trained neural network, the system was able to accurately identify the position of the charging socket. The use of image processing techniques, including grayscale conversion, histogram equalization, and edge detection, further enhanced the accuracy of detection. The implementation of a pose estimation algorithm using the *PnP* method successfully achieved minimal errors in the determination of position and orientation. Testing under various conditions confirmed the robustness and reliability of the system, making it a viable solution for autonomous EV charging applications.

Future work can focus on addressing the challenges associated with the final stages of the charging process, specifically the insertion and disinsertion of the charging plug. The next steps of this research include the integration of a force sensor to

achieve precise control during these critical stages. The force sensor provides real-time feedback on the forces exerted during the insertion and disinsertion processes, ensuring safe and accurate handling of the plug. This is essential to prevent any potential damage to the robot or the vehicle's charging socket.

In addition to the force sensor, an impedance control strategy is implemented. This control approach manages the interaction between the robotic arm and the charging socket, allowing the robot to adjust its movements based on the sensed forces. This strategy is crucial to providing compliant and safe interactions, particularly when dealing with delicate components of the charging infrastructure. Furthermore, this research can explore the use of machine learning techniques to optimize control algorithms, enhancing the system's adaptability to different vehicle models and socket placements. This is expected to improve the robustness and efficiency of the system, making it more suitable for deployment in real world scenarios.

In conclusion, this thesis has established a solid foundation for the development of an automatic EV charging system using robotic technology. The results achieved thus far indicate the potential of the system, but more work is needed to address the remaining technical challenges. The next phase of research will focus on improving the precision and reliability of the system, with an emphasis on practical application. Continued advances in sensor technology and control systems will be crucial to making autonomous charging EV a practical reality.

Appendix A

Implemented Codes

Listing A.1: Definition of the main function and parameter loading

```
def image_processing(image_path, coord_path, depth_path, image_flag, params):
    """
    Processes an image to identify and classify ellipses based on contours and
        depth information.

    Args:
        image_path (str): Path to the input image.
        coord_path (str): Path to the file containing segmented coordinates.
        depth_path (str): Path to the file containing the depth map.
        image_flag (int): Flag to indicate whether to display intermediate images
            (1 for display, 0 otherwise).
        params (list): List containing calibration parameters [fx, fy, cx, cy,
            k1, k2, p1, p2, k3].

    Returns:
        tuple: A tuple containing:
            - int: Status code (1 for success, -1 for failure).
            - list: List of classified ellipses, or None if not found.
            - numpy.ndarray: The final processed image with annotated ellipses,
                or None if not applicable.
    """

    # Values from calibration file of ZED mini
    fx, fy = params[0], params[1]

    # Load the image
    image = cv2.imread(image_path)
    shape = image.shape
    if image is None:
        print("Failed to load image.")
        return -1, None, None

    # Load the segmented coordinates
```

```
    if coord_path:
        with open(coord_path, 'r') as file:
            content = file.read().replace('\n', '')
        array = np.array(content.replace(',', ' ').replace('(', '').replace(')',
            '').split(), dtype=int)

        if len(array) == 8: # The NN returns both rectangles
            x_min, x_max, y_min, y_max = array[4], array[6], array[1], array[7]
            rect_1 = np.array([[array[4], array[5]], [array[6], array[7]]])
            rect_2 = np.array([[array[0], array[1]], [array[2], array[3]]])
            both = 1
        elif len(array) == 4: # The NN returns only the lower rectangle
            x_min, x_max, y_min, y_max = array[0], array[2], array[1], array[3]
            rect_2 = np.array([[array[0], array[1]], [array[2], array[3]]])
            both = 0
            if x_min > x_max:
                print('Incorrect segmentation')
                return -1, None, None
        else:
            print('Incorrect segmentation')
            return -1, None, None

    # Load the depth map
    with open(depth_path, 'r') as file:
        depth_map = np.array([[float(val) if val not in ['nan', 'inf', '-inf']
            else 0 for val in line.split()] for line in file])
```

Listing A.2: Pre-processing stage

```
def image_processing(image_path, coord_path, depth_path, image_flag, params):

    # ... Loading images and parameters ...

    # Values for pre-processing
    gamma = 0.7
    block_size_th = 13
    canny_minVal = 200
    canny_maxVal = 300

    # Isolate the plug using NN output
    image_cut = image[y_min:y_max, x_min:x_max]

    # Grayscale conversion
    gray_image_cut = cv2.cvtColor(image_cut, cv2.COLOR_BGR2GRAY)

    # Noise removal (median filter)
    median = cv2.medianBlur(gray_image_cut, 5)

    # Histogram equalization
    equalized = cv2.equalizeHist(median)

    # Increase luminosity using Gamma transformation
    gamma_corrected_image = np.uint8(((equalized / 255.0) ** gamma) * 255)

    # Image binarization (adaptive thresholding)
```

```
    im_th_mean = cv2.adaptiveThreshold(gamma_corrected_image, 255,
        cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV,
                                       block_size_th, 0)

    # Canny edge detection
    edges = cv2.Canny(im_th_mean, canny_minVal, canny_maxVal, apertureSize=7)

    # Restoring image to original dimensions for contour matching
    post_processed = np.zeros((shape[0], shape[1]), dtype=np.uint8)
    post_processed[y_min:y_max, x_min:x_max] = edges
    contours, _ = cv2.findContours(post_processed, cv2.RETR_LIST,
        cv2.CHAIN_APPROX_NONE)
```

Listing A.3: Ellipses fitting and classification

```
def image_processing(image_path, coord_path, depth_path, image_flag, params):

    # ... Loading images and parameters ...
    # ... Pre-processing ...

    p = 0.2  # Percentage for size tolerance
    r = 0.3  # Percentage for the ratio a/b tolerance
    ss = 4  # Minimum radius in mm
    ll = 9.5  # Maximum radius in mm

    # Isolate the depth map using the NN output to extract the mean value of z
    depth_map_cut = depth_map[y_min:y_max, x_min:x_max]
    vec = depth_map_cut[depth_map_cut != 0]
    z_mean = np.mean(vec)

    # Calculate the value of the radius in pixels
    f_mean = (fx + fy) / 2
    maximum = ll * f_mean / z_mean  # Maximum radius in pixels
    minimum = ss * f_mean / z_mean  # Minimum radius in pixels

    ellipses = []
    classified = [None] * 10

    # Split the lower rectangle into two
    if both:
        x_mean = (rect_1[0, 0] + rect_1[1, 0]) / 2
        r1 = np.array([rect_1[0, :], [x_mean, rect_1[1, 1]]])
        r2 = np.array([[x_mean, rect_1[0, 1]], rect_1[1, :]])

    # Split the upper rectangle into four
    x_mean = (rect_2[0, 0] + rect_2[1, 0]) / 2
    y_mean = (rect_2[0, 1] + rect_2[1, 1]) / 2
    r3 = np.array([rect_2[0, :], [x_mean, y_mean]])
    r4 = np.array([[x_mean, rect_2[0, 1]], [rect_2[1, 0], y_mean]])
    r5 = np.array([[rect_2[0, 0], y_mean], [x_mean, rect_2[1, 1]]])
    r6 = np.array([[x_mean, y_mean], rect_2[1, :]])
    r7 = np.array([[rect_2[0, 0], y_mean], rect_2[1, :]])

    # Ellipse fitting
    for cnt in contours:
```

```
41          x, y = cnt[:, 0, 0].astype(float), cnt[:, 0, 1].astype(float)
42          try:
43              ell = ls_ellipse(x, y)
44              ap, bp, error = ell.major_len, ell.minor_len, ell.error
45
46              # Only the ellipses inside a certain range of the axis ratio and
                    below a certain value of the fitting error are considered
47              if (1 - r) < ap / bp < (1 + r) and error < 700:
48                  ellipses.append(ell)
49          except Exception:
50              continue
51
52      # Ellipse classification
53      try:
54          ellipses.sort(key=lambda e: e.error) # Order the ellipses per ascending
                fitting error
55          if not both:
56              ellipses = [e for e in ellipses if e.mean >= 0.3 * maximum]
57              minimum = ellipses[0].mean
58
59          for ell in ellipses:
60              # Classification in case the NN returns both rectangles
61              if both:
62                  # Small holes
63                  if 0.35 * maximum * (1 - p) < ell.mean < 0.35 * maximum * (1 + p):
64                      if ell.is_inside(r1) and not classified[2]:
65                          classified[2] = ell
66                      elif ell.is_inside(r2) and not classified[3]:
67                          classified[3] = ell
68                      elif ell.is_inside(r5) and not classified[4]:
69                          classified[4] = ell
70                      elif ell.is_inside(r6) and not classified[5]:
71                          classified[5] = ell
72                  # Medium holes
73                  elif 0.55 * maximum * (1 - p) < ell.mean < 0.55 * maximum * (1 +
                        p):
74                      if ell.is_inside(r3) and not classified[6]:
75                          classified[6] = ell
76                      elif ell.is_inside(r4) and not classified[8]:
77                          classified[8] = ell
78                      elif ell.is_inside(r7) and not classified[7]:
79                          classified[7] = ell
80                  # Large holes
81                  elif maximum * (1 - p) < ell.mean < maximum * (1 + p):
82                      if ell.is_inside(r1) and not classified[0]:
83                          classified[0] = ell
84                      elif ell.is_inside(r2) and not classified[1]:
85                          classified[1] = ell
86                  # XLarge hole
87                  elif 1.65 * maximum * (1 - p) < ell.mean < 1.65 * maximum * (1 +
                        p) and ell.is_inside(rect_2) and not classified[9]:
88                      classified[9] = ell
89
90              # Classification in case the NN returns only the upper rectangle
91              else:
92                  # Small holes
```

```
                if minimum * (1 - p) < ell.mean < minimum * (1 + p):
                    if ell.is_inside(r5) and not classified[4]:
                        classified[4] = ell
                    elif ell.is_inside(r6) and not classified[5]:
                        classified[5] = ell
                # Medium holes
                elif 1.3 * minimum * (1 - p) < ell.mean < 1.3 * minimum * (1 + p):
                    if ell.is_inside(r3) and not classified[6]:
                        classified[6] = ell
                    elif ell.is_inside(r4) and not classified[8]:
                        classified[8] = ell
                    elif ell.is_inside(r7) and not classified[7]:
                        classified[7] = ell
                # XLarge holes
                elif 4 * minimum * (1 - p) < ell.mean < 4 * minimum * (1 + p) and
                    not classified[9]:
                    classified[9] = ell

        for i, ell in enumerate(classified):
            if ell is not None:
                # Extract ellipse parameters ...

                # Draw ellipse on the image ...

        # Display image if flag is set
        if image_flag == 1:
            cv2.imshow('image', image[y_min:y_max, x_min:x_max])
            cv2.waitKey(0)

        return 1, classified, image

    except Exception:
        print('No ellipses found.')
        return -1, None, None
```

Listing A.4: Least square ellipse fitting function

```
def fit_ellipse(x, y):
    """
    Fits an ellipse to a set of data points (x, y) using the Direct Least Squares
        Fitting method.

    Parameters:
    x (array-like): 1D array of x-coordinates of the data points.
    y (array-like): 1D array of y-coordinates of the data points.

    Returns:
    tuple:
        coeffs (numpy array): Array of coefficients [a, b, c, d, e, f] of the
            fitted ellipse.
        norm_2 (float): The L2 norm of the residuals, which indicates the quality
            of the fit.
    """

    # Form the design matrices D1 and D2
```

```
    D1 = np.vstack([x**2, x*y, y**2]).T  # Terms quadratic in x and y
    D2 = np.vstack([x, y, np.ones(len(x))]).T  # Linear and constant terms

    # Scatter matrices
    S1 = D1.T @ D1  # S1 is a 3x3 matrix for the quadratic terms
    S2 = D1.T @ D2  # S2 is a 3x3 matrix representing cross-terms
    S3 = D2.T @ D2  # S3 is a 3x3 matrix for the linear and constant terms

    # Constraint matrix, ensuring the ellipse is not a degenerate conic section
    C = np.array([[0, 0, 2], [0, -1, 0], [2, 0, 0]], dtype=float)

    # Compute the inverse of S3 and form matrix T
    T = -np.linalg.inv(S3) @ S2.T  # T is used to transform the problem into a
        smaller subspace

    M = S1 + S2 @ T
    M = np.linalg.inv(C) @ M  # Apply the constraint matrix
    eigval, eigvec = np.linalg.eig(M) # Eigenvalue decomposition of M

    # Ensure that the conic section is an ellipse (con > 0)
    con = 4 * eigvec[0]* eigvec[2] - eigvec[1]**2
    ak = eigvec[:, np.nonzero(con > 0)[0]].flatten()

    # Combine the coefficients for the quadratic and linear terms
    D = np.vstack([D1.T, D2.T]).T
    coeffs = np.concatenate((ak, T @ ak)).ravel()

    # Compute the L2 norm of the residuals (quality of fit)
    norm_2 = np.linalg.norm(D @ coeffs)

    return coeffs, norm_2
```

Listing A.5: *PnP* Algorithm implementation

```
def pnp_algorithm(classified, params, initial_estimate_tvec=None):
    """
    Estimates the pose of an object using the PnP algorithm.

    Args:
        classified (list): A list of detected ellipses.
        params (tuple): Camera intrinsic parameters (fx, fy, cx, cy, k1, k2, p1,
            p2, k3).
        initial_estimate_tvec (numpy.ndarray, optional): Initial translation
            vector estimate.

    Returns:
        tuple: (1, [x_rob, y_rob, z_rob, roll, pitch, yaw]) on success, or -1 on
            failure.
    """
    if classified != [None] * 10:
        # Camera intrinsic parameters
        fx, fy, cx, cy, k1, k2, p1, p2, k3 = params
        dist_coeff = np.array([k1, k2, p1, p2, k3], dtype=np.float64)
        # Camera matrix
```

```
18          camera_mat = np.array([[fx, 0, cx], [0, fy, cy], [0, 0, 1]],
                dtype=np.float64)
19          # Known 3D points in the object coordinate system
20          known_center = [(-12.67, 39.79, 0), (12.67, 39.79, 0), (-10.5, 4.63, 0),
                (10.5, 4.63, 0),
21                          (-7.5, -8.14, 0), (0, 9.32, 0), (7.5, -8.14, 0), (0,
                                -1.76, 0)]
22          # Arrays for detected 2D points and corresponding 3D points
23          point2d = np.empty((0, 2), dtype=np.float64)
24          point3d = np.empty((0, 3), dtype=np.float64)
25          is_detected = np.zeros(10)
26
27          # Collect detected points
28          for i in range(10):
29              if classified[i] is not None:
30                  is_detected[i] = 1
31                  if i > 3:
32                      point2d = np.vstack([point2d, classified[i].center])
33                      point3d = np.vstack([point3d, known_center[i - 2]])
34
35          # Additional checks for certain key points
36          if is_detected[0]:
37              point2d = np.vstack([point2d, classified[0].center])
38              point3d = np.vstack([point3d, known_center[0]])
39          elif is_detected[2]:
40              point2d = np.vstack([point2d, classified[2].center])
41              point3d = np.vstack([point3d, known_center[0]])
42
43          if is_detected[1]:
44              point2d = np.vstack([point2d, classified[1].center])
45              point3d = np.vstack([point3d, known_center[1]])
46          elif is_detected[3]:
47              point2d = np.vstack([point2d, classified[3].center])
48              point3d = np.vstack([point3d, known_center[1]])
49
50          # Ensure at least 4 points are available for solvePnP
51          if len(point2d) >= 4:
52              if initial_estimate_tvec is None:
53                  ret_val, rvec, tvec = cv2.solvePnP(
54                      point3d, point2d, camera_mat, dist_coeff,
                            flags=cv2.SOLVEPNP_ITERATIVE)
55              else:
56                  initial_estimate_rvec = np.zeros((3, 1), dtype=np.float64)
57                  initial_estimate_tvec = initial_estimate_tvec.reshape(3, 1)
58                  ret_val, rvec, tvec = cv2.solvePnP(
59                      point3d, point2d, camera_mat, dist_coeff,
                            initial_estimate_rvec, initial_estimate_tvec, True,
60                      flags=cv2.SOLVEPNP_ITERATIVE)
61
62              if not ret_val:
63                  return -1
64
65              # Convert rotation vector to rotation matrix
66              rot_mat, _ = cv2.Rodrigues(rvec)
67              # Convert rotation matrix to Euler angles (in radians)
68              r_x_mat, r_y_mat, r_z_mat = rotationMatrixToEulerAngles(rot_mat)
```

```
69              # Convert from radians to degrees
70              r_x = np.degrees(r_x_mat)
71              r_y = np.degrees(r_y_mat)
72              r_z = np.degrees(r_z_mat)
73              # Convert into the robot coordinate system
74              x_rob = float(tvec[2])
75              y_rob = float(tvec[0])
76              z_rob = float(tvec[1])
77              roll = float(r_z)
78              pitch = float(r_x)
79              yaw = float(r_y)
80
81              return 1, [x_rob, y_rob, z_rob, roll, pitch, yaw]
82          else:
83              print('Insufficient number of detected ellipses')
84              return -1
85
86      else:
87          return -1
```

Bibliography

[1] *Initial contact: The mobile charging robot – Presenting a vision.* Accessed: 2024-07-03. URL: https://www.volkswagen-group.com/en/press-releases/initial-contact-the-mobile-charging-robot-presenting-a-vision-16731 (cit. on p. 8).

[2] *Siemens autonomous charging station for electric vehicles.* Accessed: 2024-07-17. URL: https://www.siemens.com/global/en/company/stories/research-technologies/energytransition/autonomous-charging-system.html (cit. on p. 8).

[3] *Autonomous Charging Technology, RocSys.* Accessed: 2024-07-03. URL: https://www.rocsys.com/technology (cit. on p. 9).

[4] Vladimir Tadic. «Study on Automatic Electric Vehicle Charging Socket Detection Using ZED 2i Depth Sensor». In: *Electronics* 12.4 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12040912. URL: https://www.mdpi.com/2079-9292/12/4/912 (cit. on pp. 9, 11, 25, 36).

[5] Mingqiang Pan, Cheng Sun, Jizhu Liu, and Yangjun Wang. «Automatic recognition and location system for electric vehicle charging port in complex environment». In: *IET Image Processing* 14.10 (2020), pp. 2263–2272. DOI: https://doi.org/10.1049/iet-ipr.2019.1138. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-ipr.2019.1138. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ipr.2019.1138 (cit. on pp. 10, 11).

[6] Quan Pengkun, Ya'nan Lou, Haoyu Lin, Zhuo Liang, and Shichun Di. «Research on Fast Identification and Location of Contour Features of Electric Vehicle Charging Port in Complex Scenes». In: *IEEE Access* PP (June 2021), pp. 1–1. DOI: 10.1109/ACCESS.2021.3092210 (cit. on pp. 10, 11).

[7] Pengkun Quan, Ya'nan Lou, Haoyu Lin, Zhuo Liang, Dongbo Wei, and Shichun Di. «Research on Fast Recognition and Localization of an Electric Vehicle Charging Port Based on a Cluster Template Matching Algorithm». In: *Sensors* 22.9 (2022). ISSN: 1424-8220. DOI: 10.3390/s22093599. URL: https://www.mdpi.com/1424-8220/22/9/3599 (cit. on pp. 10, 12).

[8] *ZED Mini Datasheet*. Accessed: 2024-06-28. URL: https://store.stereolabs.com/cdn/shop/files/ZED_Mini_Datasheet.pdf?v=10329892655671744459 (cit. on p. 23).

[9] *NVIDIA Jetson Nano*. Accessed: 2024-07-17. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development (cit. on p. 24).

[10] *Ultralytics YOLO Docs*. Accessed: 2024-06-24. URL: https://docs.ultralytics.com/ (cit. on p. 27).

[11] John Canny. «A Computational Approach to Edge Detection». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851 (cit. on p. 38).

[12] *OpenCV documentation, Canny Edge Detection*. Accessed: 2024-05-13. URL: https://docs.opencv.org/4.9.0/da/d22/tutorial_py_canny.html (cit. on p. 38).

[13] Satoshi Suzuki and KeiichiA be. «Topological structural analysis of digitized binary images by border following». In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985), pp. 32–46. ISSN: 0734-189X. DOI: https://doi.org/10.1016/0734-189X(85)90016-7. URL: https://www.sciencedirect.com/science/article/pii/0734189X85900167 (cit. on p. 41).

[14] Radim Hal oy and Jan Flusser. «Numerically Stable Direct Least Squares Fitting of Ellipses». In: (1998). URL: https://api.semanticscholar.org/CorpusID:15772208 (cit. on p. 43).

[15] A. Fitzgibbon, M. Pilu, and R.B. Fisher. «Direct least square fitting of ellipses». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.5 (1999), pp. 476–480. DOI: 10.1109/34.765658 (cit. on p. 43).

[16] Kaj Madsen, Hans Bruun Nielsen, and Ole Tingleff. *Methods for Non-Linear Least Squares Problems (2nd ed.)* English. 2004 (cit. on p. 51).