# Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

***Masterthesis***

***Automated Positioning and Quality Assurance in Stacking Crane Operations: Integrating OpenCV Vision System for Efficiency and Precision***

**Brian Beun**

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

**PROMOTOR :**
Prof. dr. ir. Johan BAETEN

**PROMOTOR :**
Ing. Pieter VANNUETEN

Gezamenlijke opleiding UHasselt en KU Leuven

Universiteit Hasselt | Campus Diepenbeek | Faculteit Industriële Ingenieurswetenschappen | Agoralaan Gebouw H - Gebouw B | BE 3590 Diepenbeek

Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE 3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE 3500 Hasselt

2024
2025

# Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

***Masterthesis***

***Automated Positioning and Quality Assurance in Stacking Crane Operations: Integrating OpenCV Vision System for Efficiency and Precision***

**Brian Beun**

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

**PROMOTOR :**

Prof. dr. ir. Johan BAETEN

**PROMOTOR :**

Ing. Pieter VANNUETEN

UHASSELT KU LEUVEN

# Preface

Two and a half years ago, I embarked on the final chapter of my academic journey. At that moment, the path ahead seemed daunting, a steep mountain that I wasn't sure I could conquer. The idea of reaching the point where I could write my master's thesis in industrial engineering felt like a distant dream. Now, standing at the summit of that mountain and reflecting on the past two and a half years, I can confidently say that I would not have achieved this milestone without the invaluable support and encouragement I received along the way.

First and foremost, I am deeply grateful to my friends and family for their unwavering support throughout my academic career, especially when things got challenging. I want to extend special thanks to my girlfriend, Bo, for always being there for me, offering a listening ear, and standing by me every step of the way.

I would also like to express my heartfelt appreciation to the incredible faculty who guided me throughout this journey. Their expertise and mentorship were instrumental in helping me reach this point. My academic journey culminated in this master's thesis, and I am particularly grateful to Professor Johan Baeten. His guidance and feedback which I always got at lighting speed were invaluable. Future students will undoubtedly be fortunate to have him as a supervisor.

Additionally, I owe a debt of gratitude to Ing. Pieter Van Nueten, my internal supervisor at PEC, whose knowledge and dedication were pivotal throughout this process. He consistently set me on the right path to success and provided insights that greatly bettered my work. I wish him all the best in his future endeavors at Catalyx.

I hope this thesis serves as a robust foundation for further advancements in the field of automated warehouses, contributing to innovations and improvements in this exciting area.

# Table of Contents

# List of Tables

# List of Figures

## Glossary

| | |
|---|---|
| IR | Infrared |
| ROI | Region of Interest |
| DOF | Degrees of Freedom |
| TOF | Time of Flight |
| PLC | Programmable Logic Controller |
| CMM | Coordinate Measuring Machine |
| LiDAR | Light Detecting and Ranging |
| SDK | Software Development Kit |
| CAD | Computer Aided Design |
| ABS | Acrylonitrile Butadiene Styrene |
| AI | Artificial Intelligence |

## Abstract

This thesis aims to improve the initial setup phase in warehouse environments by reducing time and cost required utilizing a camera-mounted crane system. Stereo vision with infrared (IR) capabilities was chosen due to its ability to capture detailed structural features cost-effectively and without external lighting. After evaluating several cameras, the OAK-D Pro W was selected for its depth range, accuracy, superior wide field of view, and integrated IR functionality. A test setup was developed to replicate warehouse racking in a controlled environment, enabling reliable validation of the system. This included laser-cut aluminum faceplates and L-shaped support beams mounted on aluminum profiles to simulate actual racking structures. To achieve precise and consistent vertical camera movement, a motorized sled was created using a threaded rod, stepper motor, and an Arduino Uno for automated control and position tracking. Template-matching algorithms were implemented to compare detected positions with theoretical values, generating an offset table to enable accurate and efficient crane movements. Additionally, the system can automatically adjust its coordinates for the different racking positions during the initial setup phase to account for any deformations, reducing the setup time required from days and weeks to hours.

## Abstract in Nederlands

Deze thesis heeft als doel het verbeteren van de initiële installatie in magazijnomgevingen door de benodigde tijd en kosten te verminderen met behulp van een camerasysteem op een kraan voor nauwkeurige identificatie van stellingen. Stereo vision met infrarood (IR)-mogelijkheden werd gekozen vanwege de kosteneffectieve manier om gedetailleerde structurele kenmerken vast te leggen zonder externe verlichting. Na evaluatie van verschillende camera's werd de OAK-D Pro W geselecteerd vanwege zijn dieptebereik, nauwkeurigheid, brede gezichtsveld en geïntegreerde IR-functionaliteit. Een testsopstelling werd ontwikkeld om magazijnstellingen in een gecontroleerde omgeving te simuleren, wat betrouwbare validatie van het systeem mogelijk maakte. Dit omvatte laser-gesneden aluminium frontplaten en L-vormige steunen gemonteerd op aluminium profielen om echte stellingen na te bootsen. Voor precieze en consistente verticale camerabewegingen werd een gemotoriseerde slede gemaakt met een draadstang, een stappenmotor en een Arduino Uno voor geautomatiseerde controle en positietracking. Template-matching-algoritmes werden geïmplementeerd om gedetecteerde posities te vergelijken met theoretische waarden, wat resulteerde in een offsettabel voor nauwkeurige en efficiënte kraanbewegingen. Daarnaast kan het systeem tijdens de opstart fase de coördinaten automatisch aanpassen aan de verschillende stellingposities om rekening te houden met eventuele vervormingen, waardoor de installatietijd van dagen en weken tot enkele uren wordt teruggebracht.

# 1. Introduction

## 1.1. Context

This master thesis is conducted in collaboration with PEC, a company based in Leuven that specializes in the production and quality assurance of battery cells. PEC delivers comprehensive, turn-key solutions for clients seeking automated battery cell filling and quality control systems. Their services cater to both small- and large-scale production needs, ensuring flexibility and efficiency. One critical component in this process is the stacking crane Fig. 1, which works in tandem with a racking system to store and transport pallets filled with battery cells during the various stages of production and testing.


*Fig. 1 Top view of crane and racking [9]*

The racking solutions PEC employs are comprised of large, industrial storage racks, which can vary in size but typically stand around 12 meters high and 50 meters in length. These racks hold pallets that contain the battery cells, and to support the pallets securely, each pick location is equipped with two L-shaped support beams on either side as illustrated in Fig. 2. To guarantee that each support beam is properly secured and can safely hold the weight of the pallet, a safety nut seen circled in red is installed on each beam.

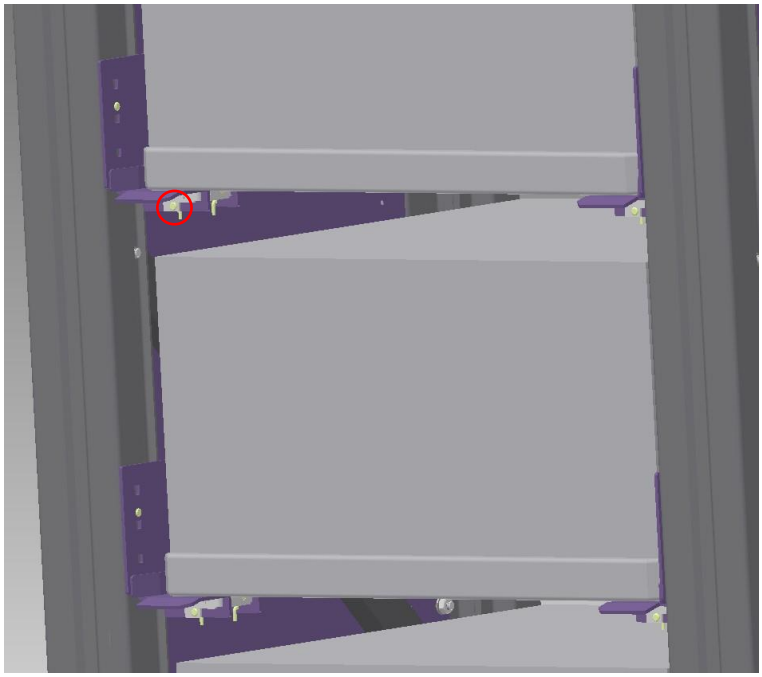

*Fig. 2 L-shaped support beams in purple and safety nut circles in red*

These storage racks are housed in temperature-controlled rooms, an essential feature for the battery testing cycle. Maintaining a stable environment ensures that the battery cells remain in optimal condition throughout the testing process. From these temperature-controlled storage locations, pallets are transported to various machines that conduct thorough quality assessments on the battery cells.

Facilitating the movement of these pallets is the stacking crane, which is strategically positioned between two racks. Mounted on the crane is a pallet handler, designed to pick up and deposit pallets. This is accomplished by extending a pallet holder beneath the stored pallet, lifting it off the support beams, retracting it safely out of the rack and into the pallet holder. The same operation can be performed in reverse to place pallets back onto the support beams in the rack or into the testing machine. This ensures seamless and automated transportation throughout the production and testing cycle.

## 1.2. Problem Statement

It is crucial that the crane knows its exact position at all times. This is done with the help of two lasers paired with corresponding reflectors. A sensor, next to each laser, measures the time delay between light leaving the laser and reaching the sensor, this time value can then be translated to distance. By mounting two lasers on the crane, one to track movement in the X-axis and another for the Y-axis, and positioning stationary reflectors along these axes, the system provides accurate real-time position feedback of the crane. This data is integrated into the programmable logic controller (PLC), allowing the crane to operate automatically. The positional information is used to map out the pick locations in the PLC, so the crane knows exactly where to retrieve and deposit pallets.

Before this automated process can start, the pick locations must be accurately set in the PLC. This is typically done during the system's startup phase where a technician has to manually move the crane to each pick location and record the X and Y coordinates for every position into the PLC. Given the size of these storage racks, this is a very time-consuming task. Depending on the number of positions, it can take several days or even weeks to complete the setup. Additionally, there's a risk of errors during installation. These errors can lead to incorrect tolerances, which may cause stress on the racking system and potentially lead to it bending. In some cases, these inaccuracies might be small enough to not affect the operation, but in other cases, they can cause entire rows of the rack to become unusable. This significantly reduces the system's overall efficiency, as fewer storage positions are available, reducing the capacity of the entire machine. These shortcomings of the current system have a major effect on the efficiency during both set up and run time operations.

## 1.3. Objectives

The current system's inefficiencies result in significant time delays and financial costs, emphasizing the need for improvements. Addressing these challenges efficiently is critical, and the goal of this master's thesis is to develop a robust solution by the end of 2024.

The proposed system will be designed to automatically detect the exact position of the support beams and compare these positions to predefined reference points in order to set each point's offset in the PLC. It will calculate any deviations in the X, Y, and Z (depth) axes, with a tolerance of no more than 5 mm to ensure accurate operation. This level of precision is crucial for ensuring the system operates reliably and safely, particularly when dealing with automated storage and retrieval.

In addition to verifying the position of the support beams, the system will measure their alignment, checking whether they are parallel or positioned at an angle. This is a critical pass/fail test to confirm that the pallets can be securely stored on the beams. The tolerance for this angle measurement should be no greater than 5 degrees. Another key feature will be the detection of the safety plug, ensuring it is securely installed. These checks will occur during system startup and must be performed on the fly while the crane is moving at a minimum speed of 0.25 m/s. Furthermore, the system will need to operate effectively in pitch-black conditions.

The solution needs to be fully autonomous, eliminating the need for manual interventions. It will communicate directly with the PLC via Ethernet/IP, allowing it to retrieve precise positional data from the existing system infrastructure. Once the system is operational, it will also conduct periodic checks during low-load periods to detect if pallets are stored correctly and ensure the safety plug is still securely in place. These continuous verifications will help maintain operational safety and efficiency, ensuring that any issues are detected early, and the system runs smoothly over time.

## 1.4. Materials and Methods

To develop the intended automated system, the first and most critical decision is selecting the appropriate camera. Especially on the following key requirements where the camera needs not only to detect the support beams but also to perform accurate depth measurements. There are two main camera types that can meet these requirements: a stereo vision camera or a time-of-flight (ToF) camera.

A time-of-flight camera offers better low-light performance and higher accuracy for depth measurements. These advantages, however, typically come with a higher cost. Stereo vision cameras on the other hand are a more affordable option. While it may not provide the same level of depth accuracy as a ToF camera, it is sufficiently accurate for this specific application. Additionally, many stereo vision cameras come equipped with an infrared (IR) sender and receiver, enabling them to function in low-light or even completely dark environments. If the chosen stereo vision camera lacks IR capabilities, an external light source, such as a light bar, would be necessary to ensure proper functionality in dark settings. Furthermore, considering the current setup's limitations, the camera should have a depth range between 0.4 m and 4 m, along with a horizontal or vertical field of view greater than 110 degrees to cover the entire area effectively.

A key aspect of the system is its ability to correctly recognize the L-shaped support beams, which may vary slightly in shape or color. The most reliable way to achieve this, is through pattern recognition, a vision-based data analysis technique that uses machine learning algorithms to identify patterns in images. The algorithm can be trained on a set of example images to detect specific parts of the support beam. Once the support beam is recognized, the system can calculate the X and Y coordinates and compare them to reference positions to determine any offsets. The same image data can also be used to measure depth, ensuring the beams are correctly positioned and not bent or misaligned.

To ensure smooth automated operation, a system-on-module (SOM) is utilized that manages the image processing and the communication with the PLC. This communication will be handled over Ethernet/IP, as the PLC holds the crane’s positional data and will require support beam position offset information from the SOM and camera to ensure accurate operation in the future.

# 2. Source Study

## 2.1. Context

In recent years, stereo vision technology has gained considerable attention for its ability to provide depth perception and object recognition in complex industrial settings. This source study explores the application of a stereo vision camera mounted on a crane, with the objective of accurately identifying racking structures in a warehouse environment. The study addresses several technical aspects of which are crucial to the effective implementation of stereo vision, including the basic principles of stereo vision technology, comparison with other depth-sensing technologies such as time-of-flight (ToF) cameras, and the specific challenges of recognizing metal racking patterns in variable lighting conditions.

Additionally, the study will investigate the role of pattern recognition algorithms in distinguishing racking from other warehouse features. This includes analyzing existing methods for metal object detection and exploring if any precedents exist for similar implementations in industrial automation. By consolidating these insights, this study aims to provide a comprehensive foundation for implementing stereo vision in a crane-based racking identification system, highlighting both potential benefits and practical challenges.

## 2.2. Stereo Vision

Stereo vision cameras are depth-sensing devices designed to replicate human binocular vision, enabling them to measure distances and generate detailed 3D representations of their surroundings. This technology captures images from two horizontally separated cameras, commonly referred to as the left and right cameras, each providing a slightly different perspective of the same scene. These subtle differences, or disparities, between the two images allow for depth calculation through triangulation. By applying algorithms that correlate these images and map the disparities, stereo vision systems create accurate depth maps. These maps are essential for applications that require precise 3D perception, such as robotics, autonomous vehicles, and industrial automation. In these settings, high accuracy in depth mapping is critical for object recognition, navigation, and interaction with complex environments [1].

Achieving accuracy in stereo vision systems relies heavily on proper camera calibration. Calibration determines both the intrinsic parameters of each camera, such as focal length and lens distortion, and the extrinsic parameters, which specify the relative position and orientation between the cameras. Even small calibration inaccuracies can lead to significant errors in depth estimation. A widely adopted approach to calibration involves observing a known pattern, such as a checkerboard, from multiple perspectives, allowing for precise parameter estimation. These objects can be both coplanar and non-coplanar as shown in Fig. 3, but it is vital to the calibration process that these objects are accurately measured beforehand for example with a CMM (Coordinate Measuring Machine). This calibration step is essential to stereo vision, as accurate parameter estimation is critical for obtaining reliable depth calculations and ensuring the system's effectiveness across a variety of applications [2].

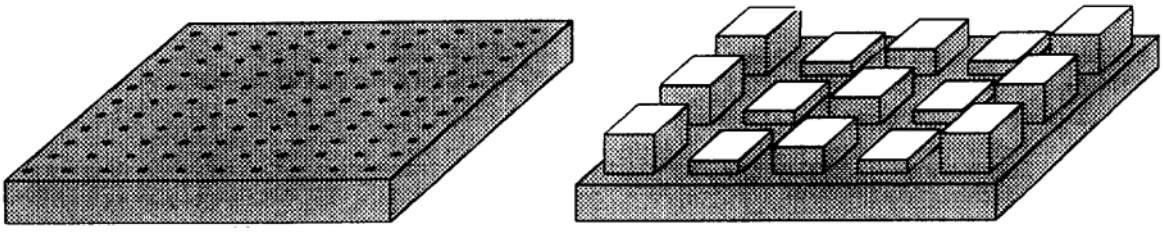

*Fig. 3 Coplanar and non-coplanar objects [2]*

Following calibration, the next critical step in stereo vision is disparity mapping, which estimates depth by analyzing pixel differences between left and right images. Two common methods for this are block matching and semi-global matching. Block matching divides the image into small blocks and searches for corresponding blocks between the two images to determine disparity. It's computationally efficient, making it fast, but it can struggle in low-texture regions where distinct features are sparse, which then lead to inaccuracies. On the other hand, semi-global matching takes a broader approach by comparing pixel disparities across multiple directions in the image, improving accuracy in complex scenes, though it requires more processing power. This method is often more effective in scenes with detailed structures and varying lighting. Recently, neural network-based disparity mapping has advanced stereo vision further, offering improved depth estimation by handling intricate patterns and variable lighting conditions effectively [3].

Stereo vision cameras offer distinct advantages over alternative depth-sensing technologies, such as time-of-flight (ToF) sensors and LiDAR (Light Detecting and Ranging). One significant benefit is that stereo vision relies on ambient light and does not require an external light source, enabling it to capture high-resolution 3D data in a cost-effective way. This characteristic makes stereo vision ideal for applications where capturing fine structural details is essential, such as in warehouse environments for identifying racking structures. Additionally, stereo vision systems typically consume less power than active systems like LiDAR, making them more suitable for mobile or portable applications where energy efficiency is prioritized.

However, stereo vision technology does face limitations. Performance can be affected in low-light or low-texture environments. In low-light conditions, stereo vision systems may struggle to capture high-quality images, compromising depth accuracy. This limitation can be mitigated by integrating infrared (IR) illumination to enhance visibility, although reflective surfaces, such as metal, may cause IR glare, further complicating the process. Similarly, in low-texture areas like plain walls or reflective surfaces, stereo matching algorithms may have difficulty identifying distinct features necessary for disparity calculation, leading to potential depth estimation errors.

In recent years, there have been significant advancements in systems that combine both stereo vision and ToF cameras, leveraging the strengths of each. By integrating stereo vision's ability to capture fine details with ToF's precise depth measurements in low-light and texture-challenged settings, these hybrid systems achieve more robust and versatile depth sensing, enhancing accuracy and reliability across a wider range of environments. These challenges remain active areas of research, with ongoing efforts focused on refining stereo matching algorithms and incorporating additional sensory inputs to improve performance in demanding conditions [4].

In conclusion, stereo vision cameras represent a versatile and efficient solution for 3D depth sensing, with applications spanning numerous industries. Despite some limitations under specific lighting and surface conditions, continuous advancements in algorithmic development and integration with supplementary sensory technologies are steadily expanding the capabilities of stereo vision systems, making them increasingly effective for industrial and mobile applications.

## 2.3. IR Glare

Infrared (IR) glare on metal surfaces, like steel and aluminum, is a common issue that affects depth sensing and object recognition in stereo vision systems. When IR light reflects off these shiny, smooth surfaces, it often creates intense, localized bright spots in the camera's field of view. This glare can obscure important details, causing parts of the metal object to appear washed out or even invisible in the depth map. This is particularly problematic in industrial environments, where metal surfaces are frequently encountered, and accurate depth data is essential [5].

To combat IR glare, several approaches can be taken. One effective method is to use polarization filters on the camera lenses. These filters reduce glare by blocking certain light waves, which helps to minimize the reflection from shiny metal surfaces. Another approach is to adjust the angle or intensity of the IR light source, as reflections can often be reduced by changing the way light interacts with the surface. Additionally, advanced software algorithms, including those that detect and exclude overexposed pixels, can further help mitigate glare. Some systems may even use multiple sensors, such as combining stereo vision with LiDAR or depth sensors that are less sensitive to IR reflections. By carefully managing IR illumination and incorporating filtering or algorithmic solutions, stereo vision systems can achieve more reliable performance when dealing with reflective metals like steel and aluminum.

## 2.4. Pattern Recognition

Pattern recognition is a process used by computer vision systems to identify specific shapes, textures, or structures within images. It works by detecting patterns in visual data, analyzing, and matching them to predefined templates or learned examples depicted in Fig. 4. In stereo vision and depth-sensing applications, pattern recognition enables cameras to locate and classify objects or structures based on their visual characteristics. In the context of detecting racking points in a warehouse, pattern recognition is especially useful because it allows the system to focus on specific features, such as the geometric layout, edges, and key points of the racking.

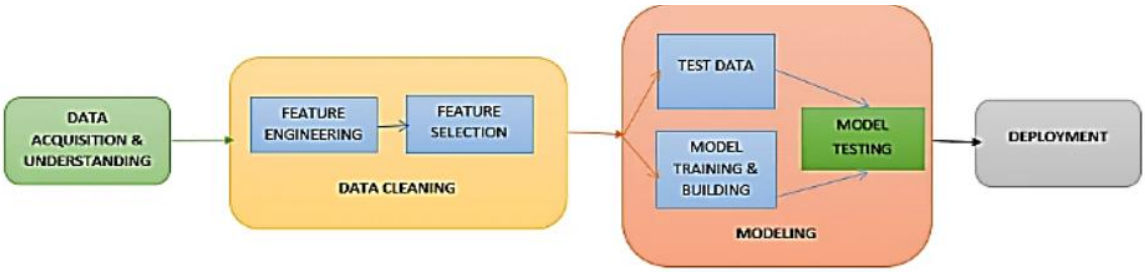


*Fig. 4 Procedure of Pattern Recognition through Machine Learning [6]*

In general, pattern recognition involves several main steps: preprocessing, feature extraction, and classification. Preprocessing cleans and prepares the raw images, typically by enhancing contrast, reducing noise, and sometimes adjusting for lighting conditions. This step is crucial for ensuring that the features in the image are clear and distinguishable. For example, if the camera is used in a dimly lit warehouse, preprocessing might involve brightening the image to make the rack edges and points more visible [6].

Feature extraction is the next step, where the system identifies and highlights the most relevant parts of the image. Features are aspects of the image that the system can analyze, such as edges, corners, textures, or shapes. In this scenario, the stereo vision system would identify key structural elements of the racking, such as the joints, beams, or support points, which are usually visually distinct and repetitive in racking structures. This allows the system to focus on the unique and consistent shapes or lines that define the racking layout. By recognizing these consistent features, the system can locate specific points on the racking more reliably.

Once these features are identified, the system uses classification to determine whether a specific set of features matches the characteristics of the racking. This is done by using pattern recognition algorithms like template matching, which directly compares the features in the live image to predefined models of the racking layout. Alternatively, machine learning approaches such as neural networks can be used, where the system is trained on many images of racking in various positions and lighting conditions to learn what a racking pattern should look like. Once trained, the system can recognize the racking points even if some elements are partially obscured or where lighting conditions change [7].

Pattern recognition is particularly suited to the task of detecting racking points because it can work with the regular, structured patterns found in industrial racks. These racks often follow predictable layouts, with consistent dimensions and repeated shapes, which make them easier to recognize than irregular objects. By identifying specific points or reference structures, the stereo vision system can calculate precise positions for each rack segment. This is essential for applications where the camera needs to navigate around or interact with the racks, such as when mounted on a crane that positions items in a warehouse.

In summary, pattern recognition offers a robust approach to detect racking points, as it enables the system to focus on and accurately identify key structural features within a controlled environment. By using techniques such as feature extraction, classification, and adapting to various lighting and structural conditions, the system can reliably detect and position itself relative to the racking, helping improve accuracy and efficiency in warehouse operations.

## 2.5. Conclusion

This source study examined the use of stereo vision technology for a crane-mounted system designed to identify racking structures in warehouses. Stereo vision was highlighted as an effective and cost-efficient method for generating high-resolution 3D depth maps, relying on ambient light rather than external sources. These qualities make it suitable for detecting fine structural details, which are essential in warehouse operations. However, challenges such as poor performance in low-light and low-texture environments remain as significant obstacles.

Key techniques for improving depth estimation, such as block matching and semi-global matching, were discussed, along with recent advances in machine learning-based disparity mapping, which improve the accuracy in complex scenes. The study also explored hybrid systems that combine stereo vision with time-of-flight (ToF) sensors, which address each technology's weaknesses and enhance overall performance in varied conditions.

Addressing reflective surfaces, particularly metals like steel and aluminum, are identified as another challenge due to IR glare. Solutions such as polarization filters, optimized IR lighting, and advanced software algorithms were reviewed to mitigate this issue. Additionally, pattern recognition techniques were found to be highly effective in identifying and classifying racking structures. Using methods like feature extraction, template matching, and machine learning, makes that these systems can reliably identify key points on racking, even in complex or changing conditions.

In summary, stereo vision offers a promising solution for racking identification, with ongoing advancements in technology and software continuing to address its limitations. By combining stereo vision with other sensors and enhancing algorithms, these systems can achieve reliable and accurate performance, making them increasingly valuable for industrial automation.

# 3. System Design and Development

## 3.1. Camera Selection

The first step in designing the stereo vision-based racking identification system was the careful selection of an appropriate camera. Several factors were critical in evaluating potential options, including the depth range and accuracy of stereo vision measurements, the field of view (FOV), and the availability of infrared (IR) capabilities for operation in pitch-black conditions. The evaluation process focused on comparing the performance and specifications of several leading stereo vision cameras: Oak-D Pro W, Orbbec Gemini, Intel RealSense D435, and ZED 2i.

The Oak-D Pro W emerged as the preferred choice due to its superior overall performance across all key parameters as shown in Table 1. It offers a robust depth range with high accuracy, which is essential for reliably identifying racking structures at various distances in a warehouse environment. The Oak-D Pro W also provides a wide field of view, which is necessary given the close proximity of the camera to the warehouse racking. Furthermore, it includes integrated IR illumination, allowing it to operate effectively in complete darkness, an essential feature for this specific use case.

*Table 1 Camera Comparison*

| Camera's | OAK-D Pro | OAK-D Pro W | OAK-D SR PoE with ToF | Orbbec Gemini 335 | Orbbec Gemini 336 | RealSense D435 | ZED Mini | ZED 2i |
|---|---|---|---|---|---|---|---|---|
| Depth Range (m) | 0.7-12 | 0.4-6 | 0.3-1 & 0.2-5 (Stereo & ToF) | 0.26-3 | 0.26-3 | 0.3-3 | 0.1-9 | 0.3-12 |
| HFOV/VFOV | 80°/55° | 127°/79.5° | 80°/55° & 70°/54.7° | 86°/55° | 86°/55° | 87°/58° | 102°/57° | 120°/70° |
| IR | Yes | Yes | Yes | Yes | Yes | Yes | No | No |
| ToF | No | No | Yes | No | No | No | No | No |
| IP | IP-66 | IP-66 | IP-67 | IP-5X | IP-5X | None | available in X version | IP-66 |
| Interface | USB-C | USB-C | PoE | USB-C | USB-C | USB-C | USB-C | USB-C |
| Onboard computing | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Price | 350 | 500 | 400 | 264 | 284 | 314 | 379 | 507 |

In comparison, the Orbbec Gemini and Intel RealSense D435, while competent in terms of depth measurement, delivers a lower overall image quality. Both models also have narrower fields of view than the Oak-D Pro W, limiting their suitability for scanning wide racking areas efficiently. These limitations would have required more complex system configurations or additional cameras to achieve the same level of coverage. Additionally, their IR capabilities are not as advanced or integrated as those of the Oak-D Pro W.

The ZED 2i, another strong contender, stand out for its high-quality depth mapping and advanced stereo vision features. However, it lacks IR capabilities, making it unsuitable for the warehouse environment, where lighting conditions are often minimal or completely absent. While the ZED 2i might be effective in well-lit scenarios, its inability to function reliably in darkness disqualifies it for this specific application.

Ultimately, the Oak-D Pro W was selected as the camera for this system because it offers the best combination of depth accuracy, field of view, and integrated IR capabilities. This decision ensures reliable and precise performance in the demanding conditions of warehouse operations while maintaining cost-effectiveness and operational simplicity.

## 3.2. Software Tools

Given the selection of the OAK-D Pro W camera, it was determined that OpenCV would be the most appropriate software framework for implementing the stereo vision system. This decision was largely driven by the seamless integration of DepthAI, the OAK-D-specific software development kit (SDK), into OpenCV. DepthAI enhances OpenCV by providing built-in support for the OAK-D Pro W's unique features, such as its stereo depth estimations. By leveraging DepthAI, the development process could directly access and manipulate the camera's depth maps without requiring extensive customization, saving time and reducing the complexity of implementation.

In addition to the integration benefits, Python was chosen as the programming language for this project due to its compatibility with OpenCV and DepthAI, as well as its ease of use. Python offers a wide range of libraries and tools for image processing, machine learning, and computer vision tasks, making it an ideal choice for this application. While other programming languages, such as C++ or Java, might offer faster execution speeds, Python's processing speed was sufficient for the requirements of this system. The use case did not demand extremely high real-time processing rates, as the stereo vision system's role primarily involved capturing and analyzing static racking structures. Python's readability and simplicity also makes it easier to develop and debug the system, which is particularly advantageous in iterative development cycles.

Moreover, Python's widespread adoption in the computer vision and AI communities ensures strong community support and extensive documentation, facilitating the resolution of potential challenges during implementation. The combination of OpenCV, DepthAI, and Python created a robust, flexible, and accessible framework that aligned with the project's goals of accuracy, efficiency, and scalability. This choice also positioned the system for future enhancements, as Python's compatibility with modern machine learning frameworks, such as TensorFlow or PyTorch, could enable more advanced pattern recognition or feature extraction functionalities down the line.

# 4. Test Setup

## 4.1. Experimental Design

To ensure the system's performance could be thoroughly tested, it was important to first build a test setup. This setup provides a controlled environment where the code and system functionality can be evaluated under conditions similar to the actual racking found in the field. By replicating the real-world scenario as closely as possible, the test setup helps identify and resolve any issues early in the development process, ensuring the system is ready for deployment.

Although the detailed design and construction of the test setup will be covered in later chapters, it's worth mentioning that this approach was chosen to create a reliable and repeatable testing environment. Having a dedicated test setup not only aids in the current project but also lays the groundwork for future development and experimentation. It can serve as a platform for testing other systems or algorithms that might be developed later for similar applications, making it a valuable resource beyond this specific use case.

## 4.2. Racking

### 4.2.1. Initial 3D modeling

Due to the lack of sufficient racking material a replica test setup had to be created. The first step in creating this test setup involved designing the face plate, a critical component that accurately mimics the racking structure found in real-world warehouse environments. Fortunately, a small section of the racking was available as a physical template, which provided a precise reference for creating the 3D model Fig. 5. Using CAD software, the dimensions and structural features of the racking were measured and replicated to closely match the original design.

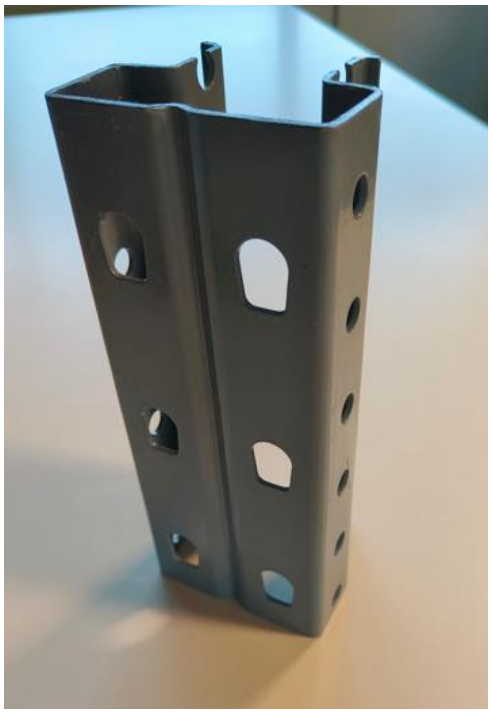

*Fig. 5 Piece of Racking*

Accuracy in the 3D model was paramount because the template matching algorithms that would be employed later depend on the model for effective calibration. Any inaccuracies could compromise the depth estimation and pattern recognition results in the stereo vision system. As part of the design process, mounting holes were incorporated into the face plate to allow secure attachment to the aluminum profile frame used in the test setup. The resulting 3D model shown in Fig. 6 served as the foundation for the subsequent stages of fabrication and testing, enabling a controlled and accurate environment to evaluate the stereo vision system's performance.

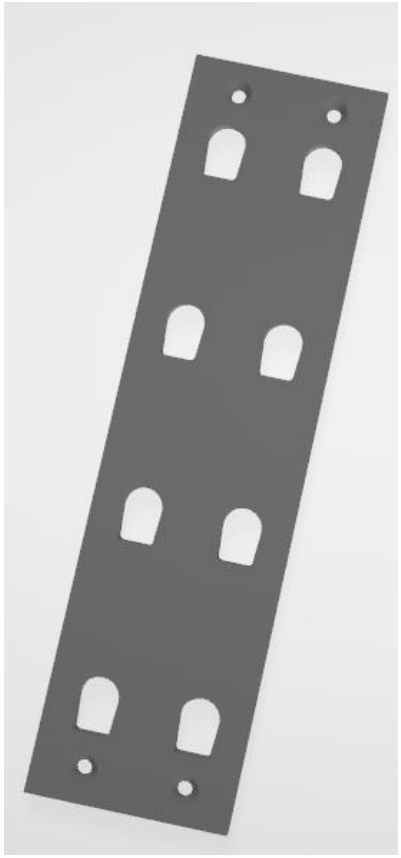

*Fig. 6 3D Model of Racking Face Plate*

### 4.2.2. Prototype Using 3D Printing

After the 3D model of the racking faceplate was completed, it was fabricated using a 3D printer to create a physical prototype. This prototype seen in Fig. 7 served as the basis for initial testing, particularly to evaluate template matching parameters and the performance of the stereo vision system. The printed model was mounted onto an aluminum profile frame, providing a stable setup for testing.

*Fig. 7 3D Printed Face Plate*

While the printed faceplate was valuable for preliminary testing, several limitations became apparent during its use. The most significant issue was the small size of the 3D-printed piece. Due to the limited dimensions, it was not possible to replicate multiple positions along the racking, which restricted the camera's ability to simulate movement up and down the structure, a critical aspect of the final application.

Moreover, the precision of the 3D printing process introduced additional challenges. The model was not entirely straight, and small imperfections, such as layer inconsistencies and surface roughness, were present. These inaccuracies could affect the reliability of template matching algorithms, as the parameters are highly sensitive to the structural and visual features of the target.

Another significant difference was the material of the 3D-printed model compared to the actual racking. The model was made from black ABS plastic, whereas the real racking is metallic. This discrepancy introduced variations in surface texture and reflectivity. The black plastic surface lacked the reflection characteristics of metal, which would impact tests involving IR illumination. Specifically, the plastic material does not produce the same glare or reflective properties that the metal racking would exhibit under IR lighting conditions. As a result, this 3D model could not fully

replicate real-world performance, particularly in assessing how the camera and software handle IR glare, and the pattern recognition accuracy.

Despite these limitations, the 3D-printed prototype provided an essential first step in the development process. It allowed for the initial validation of template matching algorithms and identified areas where improvements were needed for subsequent testing phases. This iterative approach ensured that the setup could be refined for greater accuracy and realism in future stages of the project.

### 4.2.3. Final Test Assembly

To address the limitations of the 3D-printed prototype and create a more accurate test environment, the 3D model of the racking faceplate was refined and fabricated using laser cutting. This process allows for the production of larger, more precise faceplates from 1.5mm thick aluminum sheets. These faceplates Fig. 8, measuring 1 meter in length, provide sufficient coverage to simulate at least 2-3 positions of the racking, significantly enhancing the testing environment's realism.

*Fig. 8 Aluminium Face Plate*

The aluminum faceplates are mounted onto a robust aluminum profile structure, designed to replicate the dimensions and alignment of the actual racking. This structure ensures that the faceplates are securely positioned and stable during testing. Additionally, unused L-shaped support beams from the racking are sourced from stock and integrated into the test setup seen in Fig. 9. This step is crucial for mimicking the physical characteristics and structural layout of the real racking as closely as possible.

*Fig. 9 Final Test Setup*

Careful attention is paid to the alignment and leveling of all components. The faceplates and support beams are measured and adjusted to ensure precise positioning, simulating the angles and spacing found in the actual warehouse environment. As these angles and spacings can be manually adjusted this provides the test setup the ability to replicate any possible deviations found in a warehouse environment.

The use of aluminum for the faceplates also addresses previous limitations regarding material properties. Unlike the 3D-printed plastic model, the metallic surface of the aluminum faceplates exhibits reflectivity similar to the actual racking. This allows for more realistic testing of the camera's infrared (IR) capabilities, including its handling of IR glare. As a result, the test setup provides a more accurate representation of the challenges that the stereo vision system would encounter in the field.

In conclusion, the final test setup is a significant improvement over the initial prototype. By combining precise laser-cut aluminum faceplates, authentic support beams, and a carefully leveled structure, the setup closely mirrors the physical and visual characteristics of the real racking. While some restrictions inherent to workshop conditions remain, this test environment is an excellent platform for thorough testing and refining of the stereo vision system and its associated algorithms. It provides a reliable and repeatable setting for addressing key challenges and ensuring the system's effectiveness in its intended application.

## 4.3. Camera

### 4.3.1. Vertical Sled

To accurately simulate the vertical motion of a robotic crane used in warehouse racking, a specialized test setup is designed and constructed for the stereo vision camera. Manually moving the camera up and down proved to be inconsistent and unstable, making it unsuitable for precise testing. To resolve this, a mechanism was developed to achieve controlled, repeatable vertical movement.

The first step was designing a sled to securely mount the camera Fig. 10. Using 3D modeling software, a sled was created to fit the camera and allow smooth vertical movement. The sled features a hex-shaped cavity designed to hold a hex nut securely in place. Through this nut, a 1-meter threaded rod is inserted. By rotating the threaded rod, the sled can travel up and down its length with precision.

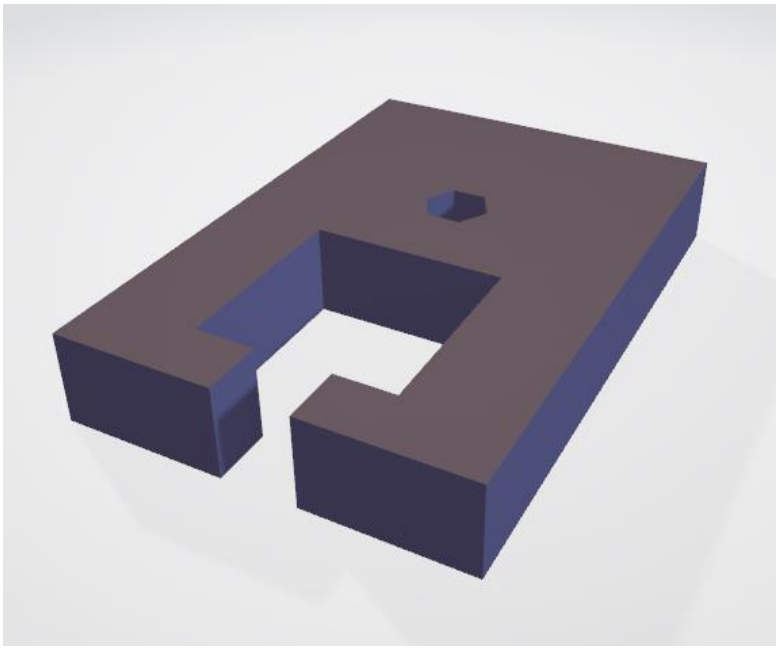

*Fig. 10 Camera Mounting Sled*

The threaded rod is mounted to an aluminum profile to maintain its position and stability. This same aluminum profile serves to guide the sled and prevent lateral movement. To further enhance the smooth operation of the system, a bearing is installed at the bottom of the rod. This reduces friction and allows the rod to rotate freely. A drill was initially used to rotate the threaded rod, driving the sled up and down. While functional, this method introduced vibrations that impacted the stability of the setup. To counteract this, two screws Fig. 11 were added to the sled, threading into the aluminum profile slots. These screws minimize wobbling, significantly improving the sled's stability during motion.

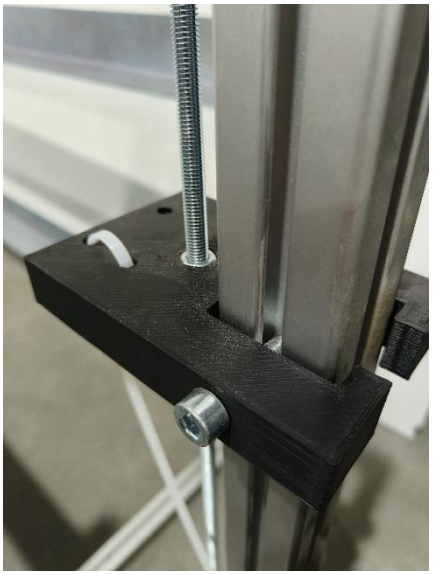

*Fig. 11 Stability Screws*

Although the drill provided an initial proof of concept, it lacked precision and control over speed. To achieve consistent and smooth movement, a stepper motor is integrated into the system. The motor is controlled by an Arduino, allowing for precise rotation of the threaded rod at a constant speed. This not only reduces vibrations but also provides the ability to record and utilize positional data during testing. The integration of the stepper motor marks a significant improvement in the system, enabling accurate and repeatable vertical movement of the camera.

This robust and stable test setup shown in Fig. 12 closely replicates the controlled motion of a robotic crane. It ensures that the stereo vision camera can be reliably positioned at various heights for thorough testing of its capabilities in identifying racking structures. Additionally, the setup is modular and reusable, making it a valuable tool for future projects involving similar testing requirements. This combination of mechanical design and automated control provides a professional and effective solution for camera movement in the testing phase.

*Fig. 12 Final Camera Test Setup*

### 4.3.2. Stepper Motor

To facilitate the vertical movement of the camera sled in the test setup, a stepper motor system was implemented, providing precise control over the sled's position. The motor is tasked with rotating a threaded rod that moves the sled up and down. This approach ensures consistent and repeatable movements, which are essential for accurate testing and calibration of the stereo vision system.

A Nema 17 stepper motor Fig. 13 is chosen for this application, paired with a DRV8825 stepper motor driver. The Nema 17 is a widely used, reliable stepper motor that offers a good balance between performance and cost, making it suitable for mid-range applications like this one. Its common use and availability through a known supplier of PEC, were additional advantages, ensuring easy procurement and compatibility with existing components. The DRV8825 driver is selected due to its compatibility with the Nema 17 and its straightforward integration with microcontrollers such as the Arduino UNO. Together, these components form a robust and versatile system that can also be repurposed for other projects in the future.

*Fig. 13 Stepper Motor*

An Arduino UNO was used to control the stepper motor and driver seen in Fig. 14 was used to simulate the PLC laser combination. The UNO was selected for its ease of use and widespread availability, making it ideal for projects requiring straightforward control mechanisms. The simplicity of the required coding further supported this choice, as the Arduino platform is well-suited for implementing basic motor control functions without the need for advanced programming expertise.

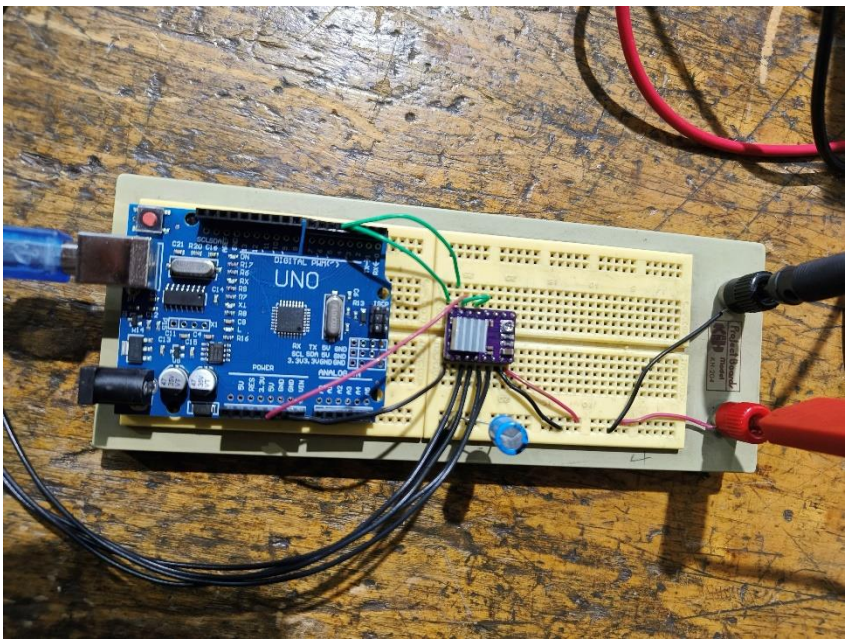


*Fig. 14 Arduino UNO and DRV8825*

The Arduino code is designed to allow control over the motor's speed, direction, and operation through serial commands. Using a connected laptop, commands can be sent via the Arduino serial interface to start or stop the motor, adjust its rotation speed, or change its direction. This setup provides a user-friendly method for operating the motor and allows fine adjustments to the sled's position during testing. By enabling these controls, the system ensures that the camera can be precisely positioned at various heights for template matching and depth measurement validation.

Additionally, the inclusion of a stepper motor in the setup reduces the inconsistencies and vibrations associated with manual movement of the sled. The motorized system's reliability and adaptability ensures that the camera can be accurately positioned for testing while maintaining a scalable design for potential future uses.

# 5. Offset and Alignment Analysis

## 5.1. Jupyter Notebook

To implement the code for the project, Jupyter Notebook was chosen as the development environment. Python was the programming language of choice due to its versatility, user-friendly syntax, and excellent integration with Depth AI, and the software library tailored for the OAK-D Pro camera. Jupyter Notebook provided a well-organized interface, ideal for testing individual code sections independently. This modularity makes it easier to debug and validate components before integrating them into the larger system, ensuring a smoother development process and better manageability throughout the project.

## 5.2. Evaluating Best Template Matching Method

The first code that was developed aimed to test and compare multiple template-matching methods available in OpenCV to determine the most effective approach for the system. Template matching is a crucial step in aligning the camera's observations with predefined templates to ensure accurate measurements and recognition of racking structures.

The program works by loading a reference image (the "template") and a test image, both in grayscale. Several OpenCV methods, such as *TM_CCOEFF, TM_CCOEFF_NORMED, TM_CCORR, TM_CCORR_NORMED, TM_SQDIFF, and TM_SQDIFF_NORMED*, are applied to measure how well the template matches different regions of the test image. Each method calculates a score indicating the level of similarity, with some methods emphasizing correlations and others focusing on differences.

For each method, the program determines the best match location by identifying the maximum or minimum value from the resulting similarity map, depending on the method used Fig. 15. A bounding box is drawn around the detected region to visually inspect the match. By iterating through all the available methods, the program provides insights into which algorithm performs best for the specific use case, considering factors like accuracy, robustness, and computational efficiency.

```
methods = [cv2.TM_CCOEFF, cv2.TM_CCOEFF_NORMED, cv2.TM_CCORR,
           cv2.TM_CCORR_NORMED, cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]

for method in methods:
    img2 = img.copy()

    result = cv2.matchTemplate(img2, template, method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        location = min_loc
        print (min_loc)
    else:
        location = max_loc
        print (max_loc)
```

*Fig. 15 Best Template Matching Code*

This testing process was essential in selecting the most suitable template-matching approach for the system, ensuring reliable and consistent performance in identifying features in the warehouse racking environment. The flexibility of OpenCV allowed for easy experimentation and visualization of the results, facilitating a well-informed decision, in the end *TM_SQDIFF_NORMED* gave the most consistent results for this application given out of the 20 tests run it only incorrectly identified the bracket ones while the others made 5 or more incorrect detections.

## 5.3. Camera Pipeline

The camera pipeline is established using the DepthAI framework to enable effective integration of the cameras and facilitate the necessary image processing tasks for stereo vision. At the core of the system is the *dai.Pipeline*, which coordinates the camera streams and processing stages, ensuring efficient capture and management of video and depth data. This pipeline is specifically designed to work with the OAK-D camera, which features both color and mono cameras essential for stereo vision. The pipeline is constructed to accommodate both the color camera for visual feedback and the left and right mono cameras, which are key for depth perception in stereo vision.

The setup of the pipeline begins with configuring the color camera (CAM_A) to capture video at 720p resolution. This camera serves as the main source of visual input, providing color frames that are essential for object recognition and template matching. Additionally, two mono cameras are set up for stereo depth processing. These mono cameras, one placed on the left (CAM_B) and the other on the right (CAM_C), are configured at the same resolution to capture the necessary data for depth computation. This stereo setup allows the system to generate disparity maps by comparing the slight differences between the images captured by the left and right cameras, a fundamental process for 3D scene reconstruction and depth measurements.

In terms of camera configuration, the *getMonoCamera* function is used to define each mono camera's properties, such as its resolution and socket connections to the pipeline. This approach ensures that the cameras are correctly initialized for stereo vision. The left and right cameras are then linked to a stereo depth processor, which combines the outputs from the two mono cameras. The depth processor is configured to enhance depth precision by enabling features like left-right consistency checks, extended disparity, and subpixel processing Fig. 17. These settings improve the quality of the generated stereo depth data, which can later be used for object detection and spatial analysis.

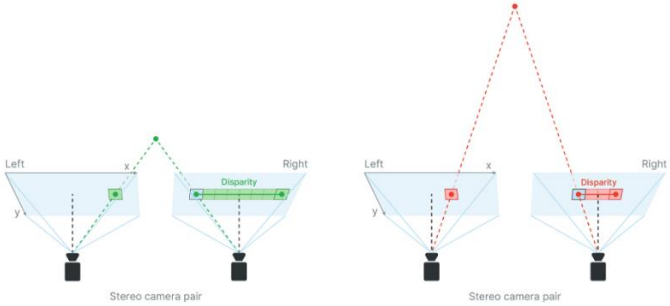


*Fig. 16 Visualization of Stereo Vision Depth Measurements [8]*

```python
# Function to set up mono cameras
def getMonoCamera(pipeline, isLeft):
    mono = pipeline.createMonoCamera()
    mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
    if isLeft:
        mono.setBoardSocket(dai.CameraBoardSocket.CAM_B)  # Use CAM_B for left camera
    else:
        mono.setBoardSocket(dai.CameraBoardSocket.CAM_C)  # Use CAM_C for right camera
    return mono

# Function to configure stereo depth
def getStereoPair(pipeline, monoLeft, monoRight):
    stereo = pipeline.createStereoDepth()
    stereo.initialConfig.setLeftRightCheck(True)
    stereo.initialConfig.setExtendedDisparity(True)  # Enable extended disparity
    stereo.initialConfig.setSubpixel(True)  # Enable subpixel for higher precision
    stereo.initialConfig.setConfidenceThreshold(200)  # Adjust confidence threshold
    stereo.initialConfig.setMedianFilter(dai.StereoDepthProperties.MedianFilter.KERNEL_7x7)  # Apply median filtering

    monoLeft.out.link(stereo.left)
    monoRight.out.link(stereo.right)

    return stereo
```

*Fig. 17 Camera Setup Pipeline*

A critical component of this pipeline is its flexibility in adjusting the various camera parameters, such as resolution and depth settings, to optimize performance based on the requirements of the specific task. The system is capable to perform real-time adjustments, which is beneficial in a warehouse environment, where lighting and object positions can vary. The pipeline also includes functions for setting up output queues, such as *xout_video* for video frames and *xout_disp* for depth information. These output queues are used to stream live video and disparity data from the cameras, which can be processed and visualized in real-time.

The integration of the video and stereo cameras into a single pipeline simplifies the process of gathering and managing camera data, enabling a smooth flow of information from the sensors to the computer for processing. This real-time video feed and its associated disparity data are crucial for tasks such as racking identification, where both visual and depth information are necessary for precise measurements. The modular nature of the pipeline also allows for future expansion, for example the addition of more cameras and/or advanced processing algorithms, making it a versatile platform for a range of applications.

By using the DepthAI framework, the setup ensures reliable performance while maintaining flexibility for future development. The ability to process the data and/or to configure the pipeline in real-time provides a solid foundation for integrating further image processing steps, such as template matching, that are essential for the system's overall functionality.

## 5.4. Calibration

The code for taking screenshots of the left and right bracket regions is a crucial part of the template creation process, used for aligning the crane with the warehouse racking and generating reference templates. This setup is designed to allow an operator to manually position and align the crane with the first position. Once aligned, the operator can use the system to select regions of interest (ROIs) that will be saved as template images for later use in object recognition tasks.

This approach to capturing the left and right bracket regions is crucial for creating the template reference points that will be used in later sections for comparison and alignment. The saved templates will serve as a benchmark for positioning the crane and camera during the startup and operational phases, allowing for automated comparisons and the creation of an offset table.

The process starts by initializing the necessary variables for handling the Region of Interest (ROI) selection. The variables *roi_start, roi_end*, and drawing track the state of the rectangle being drawn by the operator. The *screenshot_count* variable keeps track of the number of ROIs saved, ensuring that the left and right bracket regions are saved sequentially. Currently the initial instruction text guides the operator through the process, informing them to select the left bracket first. This is a process that could be automated in future iterations.

The core functionality is provided by the *draw_rectangle* callback function, which is linked to mouse events. The operator can click and drag the mouse to define the region of interest on the live camera feed. When the mouse button is pressed (*cv2.EVENT_LBUTTONDOWN*), the drawing state is set to True, and the starting point of the rectangle (*roi_start*) is recorded. As the mouse moves (*cv2.EVENT_MOUSEMOVE*), the rectangle's end point (*roi_end*) is updated. Once the mouse button is released (*cv2.EVENT_LBUTTONUP*), the rectangle is finalized, and the drawing state is set to False Fig. 18.


*Fig. 18 Taking Reference Images*

Within the main loop, the camera feed is continuously captured, and if an ROI is being drawn, it is overlaid on the frame as a green rectangle. The instruction text is displayed at the top-left corner of the screen, guiding the operator through the process. If the user presses the 's' key, the selected ROI is saved as an image file. The first ROI saved is stored as *TemplateROI_L.jpg* for the left bracket, and the second as *TemplateROI_R.jpg* for the right bracket Fig. 19. These template images are essential for the template-matching process, which will later be used to compare crane positions and alignments with the reference coordinates.

```
if screenshot_count == 0:
    # Save the first ROI as TemplateROI_L
    cv2.imwrite('C:/Data/TemplateROI_L.jpg', roi)
    print("ROI saved as 'TemplateROI_L.jpg'.")
    instruction_text = "Select Right Bracket And Press 's' to Save"  # Change text to "Select Right Bracket"
    screenshot_count += 1  # Increment the counter to save the next ROI
elif screenshot_count == 1:
    # Save the second ROI as TemplateROI_R
    cv2.imwrite('C:/Data/TemplateROI_R.jpg', roi)
    print("ROI saved as 'TemplateROI_R.jpg'.")
    break  # Exit the loop after saving the second ROI
```

*Fig. 19 Code for Saving Reference Images*

The cv2.imwrite function is used to save the selected ROIs as image files on the local disk, and the coordinates of the top-left and bottom-right corners of the selected rectangle are printed for reference. Once the second ROI is saved, the process exits, and the program ends. If the user decides not to save the ROI, they can quit the process by pressing the 'q' key.

## 5.5. Template Matching and Offset Calculation

The extended implementation of the template matching system introduces several enhancements to improve its integration within the larger project scope, focusing on real-time processing, coordinate the comparison method, and offset calculation. This system continually analyzes the frame from the OAK-D Pro W camera, performing template matching on grayscale representations of the live video stream using two predefined template images (*TemplateROI_L* and *TemplateROI_R*). The OpenCV function *cv2.matchTemplate*, combined with the *cv2.TM_SQDIFF_NORMED* method, is employed to detect the regions within each frame that most closely resemble the templates. The resulting coordinates for the top-left corners of the matched areas are extracted using *cv2.minMaxLoc*, enabling precise localization Fig. 20.

```
# Perform template matching for TemplateROI_L
result_l = cv2.matchTemplate(gray_frame, template_l, cv2.TM_SQDIFF_NORMED)
min_val_l, max_val_l, min_loc_l, max_loc_l = cv2.minMaxLoc(result_l)
top_left_l = min_loc_l
bottom_right_l = (top_left_l[0] + template_l_width, top_left_l[1] + template_l_height)

# Draw rectangle around the detected area for TemplateROI_L
cv2.rectangle(color_frame, top_left_l, bottom_right_l, (0, 255, 0), 2)
text_l = f"Match: {1-min_val_l:.2f}, X: {top_left_l[0]}, Y: {top_left_l[1]}"
cv2.putText(color_frame, text_l, (top_left_l[0], top_left_l[1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

*Fig. 20 Implemented Template Matching Code*

These detected coordinates are subsequently compared against predefined reference coordinates *(REFERENCE_COORDS_L and REFERENCE_COORDS_R).* By calculating the differences in both the X and Y axes, the system determines the offset between the current alignment and the ideal reference positions. During execution, the program displays this information visually by overlaying rectangles on the video feed, with green and red annotations, respectively, for the left and right templates, respectively seen in Fig. 21. Corresponding disparity data from the depth stream is processed in parallel to calculate depth values within the detected regions, enhancing the functionality of the alignment system.


*Fig. 21 Dual Template Matching with Coordinates*

When a user presses the 's' key, the system outputs the detected coordinates, reference coordinates, and the calculated offsets for both templates to the console, aiding in debugging and verification shown in Fig. 22. However, the final implementation will eliminate manual triggers in favor of signals from the crane control system, reflecting a transition toward full automation. Instead of printing offsets to the console, the system will save them in an offset table, forming a persistent record for alignment correction and system analytics. This shift from manual operation to automated data handling enhances the system's precise, real-time alignment in the larger operational workflow. By continuously monitoring and comparing alignment data, this implementation significantly contributes to the overall reliability and accuracy of the project.

```
Left Template - Detected: (139, 529), Reference: (139, 529), Difference: (X: 0, Y: 0)
Right Template - Detected: (1572, 533), Reference: (1572, 533), Difference: (X: 0, Y: 0)
Left Template - Detected: (187, 584), Reference: (139, 529), Difference: (X: 48, Y: 55)
Right Template - Detected: (1530, 591), Reference: (1572, 533), Difference: (X: -42, Y: 58)
```

*Fig. 22 Template Matching Offset Output*

## 5.6. Depth Measurement

The depth measurement component in the system is a critical feature for accurately determining the distance between the camera and objects in the scene. This process starts with the disparity map generated by the stereo pair of cameras. The stereo depth configuration employs settings such as subpixel accuracy *(stereo.initialConfig.setSubpixel(True))* and extended disparity *(stereo.initialConfig.setExtendedDisparity(True))* to enhance precision. The disparity map, representing pixel-wise differences between the left and right camera images, is converted into depth values using a known baseline distance between the cameras and the focal length. Depth values are further refined by applying a median filter, in this case (KERNEL_7x7) to reduce noise and improve clarity in the resulting disparity frame.

Given that the main camera, used for template matching, has a different field of view compared to the stereo cameras, corrections are implemented to align the regions of interest. This involves scaling and translating the coordinates of the detected template areas to match the stereo cameras perspective. Adjustments, such as dividing the X and Y coordinates by specific scaling factors, ensure that the depth measurements correspond to the exact areas detected in the primary video feed.

To mitigate inaccuracies caused by potential graininess or noise in the disparity map, the depth measurement is calculated as an average over a defined rectangular region rather than relying on a single point as seen in Fig. 23. The *calculateAverageDepth* function extracts the region of interest from the disparity map and computes the mean of all valid disparity values within the rectangle. This approach minimizes the effect of anomalies in individual pixels, resulting in more reliable depth estimates. By leveraging an averaged area, the system produces robust measurements that are less susceptible to minor variations or noise in the feed.

```
# Get and process the disparity map
disparityFrame = getFrame(disparity_queue)
if disparityFrame is not None:
    disparity = (disparityFrame * disparityMultiplier).astype(np.uint8)
    disparity = cv2.applyColorMap(disparity, cv2.COLORMAP_JET)

    # Draw the rectangles on the disparity map as well
    adjusted_top_left_l = (int(top_left_l[0] / 1.25), int(top_left_l[1] / 1.6))
    adjusted_bottom_right_l = (
        int(adjusted_top_left_l[0] + template_l_width / 1.6),
        int(adjusted_top_left_l[1] + template_l_height / 1.6)
    )
    adjusted_top_left_r = (int(top_left_r[0] / 1.25), int(top_left_r[1] / 1.6))
    adjusted_bottom_right_r = (
        int(adjusted_top_left_r[0] + template_r_width / 1.6),
        int(adjusted_top_left_r[1] + template_r_height / 1.6)
    )

    # Calculate the average depth within the regions defined by the rectangles
    avg_disp_l, depth_l = calculateAverageDepth(disparityFrame, adjusted_top_left_l, adjusted_bottom_right_l)
    avg_disp_r, depth_r = calculateAverageDepth(disparityFrame, adjusted_top_left_r, adjusted_bottom_right_r)
```

*Fig. 23 Disparity Map and Average Area Depth Measurement*

The resulting depth values are displayed in real time on the annotated video stream and disparity map shown in Fig. 24, providing a clear visual representation of the object distances. This depth information is then stored in the offset table along with the x and y coordinates. This implementation not only ensures accurate depth calculations, but also integrates seamlessly into the overall system by delivering reliable spatial information to support precise alignment and positioning within the larger project.

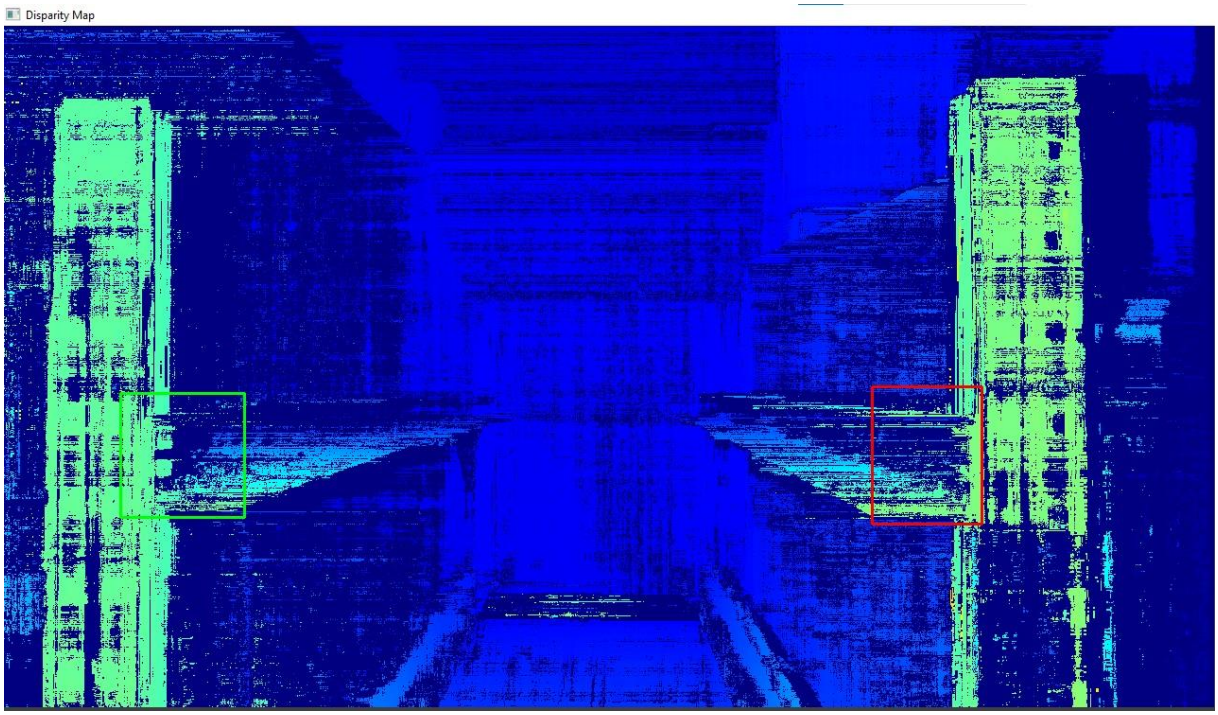


*Fig. 24 Disparty Map*

# 6. Test Setup Implementation and Testing

The final testing and implementation of the test setup demonstrated the effectiveness and accuracy of the system under realistic conditions. With the stepper motor operating at maximum speed, the camera successfully tracked the bracket even in the presence of significant vibrations, showcasing its robustness. To evaluate the system's accuracy, the camera was moved up and down by known amounts, and the pixel offsets (X and Y) returned by the code were recorded displayed in Table 2.

*Table 2 Actual vs Calculated Offset*

| Actual mm offset | Pixel offset | mm/pixel | \|mm/Pixel\| | Pixel Offset * Average | \|Actual - Calculated Offset\| |
|---|---|---|---|---|---|
| 20 | 42 | 0,48 | 0,48 | 19,32 | 0,68 |
| 33 | 72 | 0,45 | 0,45 | 33,12 | 0,62 |
| -14 | -31 | 0,45 | 0,45 | -14,26 | 0,26 |
| 15 | 34 | 0,44 | 0,44 | 15,64 | 0,64 |
| 54 | 113 | 0,48 | 0,48 | 51,98 | 2,02 |
| 36 | 81 | 0,44 | 0,44 | 37,26 | 1,26 |
| -12 | -26 | 0,46 | 0,46 | -11,96 | 0,04 |
| -63 | -133 | 0,47 | 0,47 | -61,18 | 1,82 |
| | | **Average:** | **0,46** | | **0,9** |
| | | | | **Standard Deviation:** | **0,7** |

Using these offsets, the average conversion rate of 0.46 mm per pixel was established. By multiplying this value by the measured pixel offsets, the calculated offsets in millimeters were obtained. Comparing these calculated offsets with the actual offsets revealed an average difference of 0.9 mm and a standard deviation of 0.7 mm. These results confirm that the system is highly accurate and suitable for this application given the required accuracy should be less than 5mm. For the depth measurement an average of both sides were taken and compared to the actual distance. These results showed an overall lower precision and repeatability of ±2mm due to the graininess of the depth map, nevertheless these results are accurate enough for this implementation. However, it is essential to note that these tests were conducted in a controlled environment, and further testing in an actual warehouse is necessary to validate the system's performance under real-world conditions. This step is crucial to account for variables like lighting, surface inconsistencies, and operational vibrations that may differ from the test setup.

# 7. Conclusion

In conclusion, this thesis presents the development and implementation of a comprehensive system leveraging template matching, depth measurement, and coordinate comparison for precise spatial analysis. The integration of these technologies demonstrates a robust approach in identifying and tracking objects while providing real-time depth data in a warehouse environment. The design emphasizes practical considerations such as reliability, efficiency, and speed, ensuring that the system is suitable for real-world applications.

The template matching process, utilizing pre-defined reference images, allows the system to detect and track objects accurately, while the mechanism for comparing detected coordinates to reference points ensures precise alignment. The incorporation of a depth measurement system, which employs stereo vision and disparity mapping techniques, enhances the system's capabilities by providing spatial depth data. The implementation addresses challenges such as noise and graininess in the feed by using averaged depth measurements over defined areas, ensuring greater accuracy and robustness. These procedures ensured the systems accuracy fell well within the 5mm requirement.

A significant benefit of this solution is its potential to dramatically reduce the setup phase from days or weeks to just a few hours, making the configuration process much faster and more cost-effective. This efficiency gains not only result in reduced operational downtime but also streamline the deployment of similar systems. Furthermore, the presence of the stereo camera opens up opportunities for future improvements, such as leveraging stereo vision to analyze structural elements for alignment or angle deviations, further enhancing operational reliability.

This project also emphasizes adaptability by accommodating field-of-view differences between cameras and implementing scaling corrections to ensure alignment between the main and stereo cameras. These adjustments not only ensure accurate depth calculations but also align seamlessly with the object detection process, demonstrating the importance of cohesive system design. The use of real-time feedback mechanisms, such as visual annotations and offset comparisons, further highlights the practical utility of the system.

Overall, this work provides a strong foundation for future advancements in template-based detection and depth analysis. The modularity of the system enables potential enhancements, such as replacing manual triggers with automated signals or integrating additional sensors for improved performance.

# 8. Future Additions and Improvements

While the current implementation marks a significant first step in leveraging stereo vision technology for racking identification, there are several important areas for future development and improvement. The most critical next step is to test the system in an actual warehouse environment. This requires integrating the developed code into the compute unit of the robotic crane already in operation. Additionally, communication must be established with the PLC, which manages the crane's motors, to provide accurate positional data.

Due to time constraints the detection of the safety isn't completed in the current project but can be added using similar template matching techniques and/or utilizing the OpenCV cv2.findContours function. Another potential enhancement is to utilize the stereo vision capabilities to measure the angle of the support beams. By determining whether these beams are parallel, the system could detect structural misalignments and provide error messages to the operator if the deviations exceed a predefined threshold. This functionality could be implemented using Python's edge detection functions.

Incorporating such features would not only increase the system's utility but also improve efficiency in warehouse operations. However, further research and iterative testing will be necessary to refine these functionalities. This continued development will ensure that the solution becomes a robust, reliable tool for industrial applications.

## Acknowledgments

"chatgpt.com," OpenAI, [Online]. Available: https://chatgpt.com/.

Was used to improve the overall language quality of the text, and aided in the python code

## Reference list

[1] M. A. Mahammed, A. I. Melhum, and F. A. Kochery, "Object Distance Measurement by Stereo VISION," *2013 International Journal of Science and Applied Information Technology (IJSAIT)*, vol. 2, no. 2, 2013.

[2] J. J. Aguilar, F. Torres, and M. A. Lope, "Stereo vision for 3D measurement: accuracy analysis, calibration and industrial applications," *Measurement*, vol. 18, no. 4, pp. 193–200, Aug. 1996, doi: 10.1016/S0263-2241(96)00065-6.

[3] A. Jafari Malekabadi, M. Khojastehpour, and B. Emadi, "Comparison of block-based stereo and semi-global algorithm and effects of pre-processing and imaging parameters on tree disparity map," *Sci Hortic*, vol. 247, pp. 264–274, Mar. 2019, doi: 10.1016/J.SCIENTA.2018.12.033.

[4] S. Á. Guðmundsson, H. Aanæs, and R. Larsen, "Fusion of stereo vision and Time-Of-Flight imaging for improved 3D estimation," *International Journal of Intelligent Systems Technologies and Applications*, vol. 5, no. 3–4, 2008, doi: 10.1504/IJISTA.2008.021305.

[5] G. W. Poling, "Infrared reflection studies of metal surfaces," *J Colloid Interface Sci*, vol. 34, no. 3, pp. 365–374, Nov. 1970, doi: 10.1016/0021-9797(70)90195-5.

[6] T. D. Adugna, A. Ramu, and A. Haldorai, "A Review of Pattern Recognition and Machine Learning," 2024. doi: 10.53759/7669/jmc202404020.

[7] "Pattern Matching Techniques," National Instruments. Accessed: Nov. 04, 2024. [Online]. Available: https://www.ni.com/docs/en-US/bundle/ni-vision-concepts-help/page/pattern_matching_techniques.html#:~:text=The%20pattern%20matching%20process%20consists,searching%20in%20the%20inspection%20image.

[8] "Configuring Stereo Depth," Luxonis. Accessed: Jan. 16, 2025. [Online]. Available: https://docs.luxonis.com/hardware/platform/depth/configuring-stereo-depth/

[9] "Een virtueel kijkje in de kluis van DNB," DeNederlandscheBank. Accessed: Oct. 01, 2024. [Online]. Available: https://www.dnb.nl/algemeen-nieuws/nieuws-2024/een-virtueel-kijkje-in-de-kluis-van-dnb/

## Attachments

```python
# TAKING SCREENSHOT OF LEFT & RIGHT BRACKET

import depthai as dai
import cv2

# Initialize variables for the ROI (Region of Interest)
roi_start = None
roi_end = None
drawing = False
screenshot_count = 0  # Counter to track which ROI to save
instruction_text = "Select Left Bracket And Press 's' to Save"  # Initial instruction text

# Callback function for mouse events to draw the rectangle
def draw_rectangle(event, x, y, flags, param):
    global roi_start, roi_end, drawing

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        roi_start = (x, y)
        roi_end = (x, y)
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            roi_end = (x, y)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        roi_end = (x, y)

# Create a pipeline
pipeline = dai.Pipeline()

# Add the color camera node
cam_rgb = pipeline.createColorCamera()
cam_rgb.setBoardSocket(dai.CameraBoardSocket.CAM_A)  # Main camera
cam_rgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)  # resolution
cam_rgb.setInterleaved(False)

# Create an XLinkOut node to get the RGB frames
xout_video = pipeline.createXLinkOut()
xout_video.setStreamName("video")
cam_rgb.video.link(xout_video.input)

# Start the pipeline
with dai.Device(pipeline) as device:
    # Get the output queue
    video_queue = device.getOutputQueue(name="video", maxSize=1, blocking=False)

    # Create a window and set a mouse callback
    cv2.namedWindow("OAK-D Lite - Main Camera")
    cv2.setMouseCallback("OAK-D Lite - Main Camera", draw_rectangle)

    while True:
        # Get the frame from the video queue
        frame = video_queue.get().getCvFrame()
        display_frame = frame.copy()

        # Draw the rectangle on the frame if the user is selecting a region
        if roi_start and roi_end:
            cv2.rectangle(display_frame, roi_start, roi_end, (0, 255, 0), 2)

        # Display dynamic instruction text at the top-left corner
        cv2.putText(display_frame, instruction_text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        # Display the frame
        cv2.imshow("OAK-D Lite - Main Camera", display_frame)
```

```
        # Check for key presses
        key = cv2.waitKey(1) & 0xFF
        if key == ord('s'):  # Press 's' to save the selected ROI
            if roi_start and roi_end:
                # Ensure the rectangle coordinates are valid
                x1, y1 = min(roi_start[0], roi_end[0]), min(roi_start[1], roi_end[1])
                x2, y2 = max(roi_start[0], roi_end[0]), max(roi_start[1], roi_end[1])

                # Print the coordinates
                print(f"Coordinates: Top-left ({x1}, {y1}), Bottom-right ({x2}, {y2})")

                # Crop and save the ROI
                roi = frame[y1:y2, x1:x2]

                if screenshot_count == 0:
                    # Save the first ROI as TemplateROI_L
                    cv2.imwrite('C:/Data/TemplateROI_L.jpg', roi)
                    print("ROI saved as 'TemplateROI_L.jpg'.")
                    instruction_text = "Select Right Bracket And Press 's' to Save"  # Change text to
                    screenshot_count += 1  # Increment the counter to save the next ROI
                elif screenshot_count == 1:
                    # Save the second ROI as TemplateROI_R
                    cv2.imwrite('C:/Data/TemplateROI_R.jpg', roi)
                    print("ROI saved as 'TemplateROI_R.jpg'.")
                    break  # Exit the loop after saving the second ROI
            else:
                print("No ROI selected. Please draw a rectangle before saving.")

        elif key == ord('q'):  # Press 'q' to quit without saving
            print("Quit without saving.")
            break

    cv2.destroyAllWindows()
```

```python
#TEMPLATE MATCHING WITH DEPTH MEASUREMENT & OFFSET CALC

import depthai as dai
import cv2
import numpy as np

# Constants for depth calculation
BASELINE_METERS = 0.075  # 7.5 cm
FOCAL_LENGTH_PIXELS = 885  # Focal length in pixels
SUBPIXEL_SCALING = 32  # Subpixel scaling factor

# Reference coordinates
REFERENCE_COORDS_L = (170, 619)  # Reference coordinates for Left template
REFERENCE_COORDS_R = (1557, 608)  # Reference coordinates for Right template

# Function to get a frame from the queue
def getFrame(queue):
    frame = queue.get()
    return frame.getCvFrame()

# Function to set up mono cameras
def getMonoCamera(pipeline, isLeft):
    mono = pipeline.createMonoCamera()
    mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
    if isLeft:
        mono.setBoardSocket(dai.CameraBoardSocket.CAM_B)  # Use CAM_B for left camera
    else:
        mono.setBoardSocket(dai.CameraBoardSocket.CAM_C)  # Use CAM_C for right camera
    return mono

# Function to configure stereo depth
def getStereoPair(pipeline, monoLeft, monoRight):
    stereo = pipeline.createStereoDepth()
    stereo.initialConfig.setLeftRightCheck(True)
    stereo.initialConfig.setExtendedDisparity(True)  # Enable extended disparity
    stereo.initialConfig.setSubpixel(True)  # Enable subpixel for higher precision
    stereo.initialConfig.setConfidenceThreshold(200)  # Adjust confidence threshold
    stereo.initialConfig.setMedianFilter(dai.StereoDepthProperties.MedianFilter.KERNEL_7x7)  # Apply median filtering

    monoLeft.out.link(stereo.left)
    monoRight.out.link(stereo.right)

    return stereo

# Function to calculate average depth in a given rectangle
def calculateAverageDepth(disparityFrame, top_left, bottom_right):
    x1, y1 = top_left
    x2, y2 = bottom_right

    # Ensure valid rectangle coordinates
    x1, x2 = min(x1, x2), max(x1, x2)
    y1, y2 = min(y1, y2), max(y1, y2)

    # Extract the region of interest
    roi = disparityFrame[y1:y2, x1:x2]
    if roi.size > 0:
        # Calculate the average depth using valid disparity values
        valid_disparity = roi[roi > 0]
        if valid_disparity.size > 0:
            avg_disparity = np.mean(valid_disparity)
            # Convert disparity to depth (meters) with subpixel scaling
            depth_in_meters = ((BASELINE_METERS * FOCAL_LENGTH_PIXELS) / (avg_disparity / SUBPIXEL_SCALING)) / 6
            return avg_disparity, depth_in_meters
    return None, None

# Load the template images
template_l = cv2.imread("TemplateROI_L.jpg", cv2.IMREAD_GRAYSCALE)
template_r = cv2.imread("TemplateROI_R.jpg", cv2.IMREAD_GRAYSCALE)

if template_l is None or template_r is None:
    raise FileNotFoundError("One or both template images not found. Make sure the files exist in the working directory.")

template_l_height, template_l_width = template_l.shape
template_r_height, template_r_width = template_r.shape
```

```
pipeline = dai.Pipeline()

cam_rgb = pipeline.createColorCamera()
cam_rgb.setBoardSocket(dai.CameraBoardSocket.CAM_A)  # Main camera
cam_rgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_720_P)
cam_rgb.setInterleaved(False)

xout_video = pipeline.createXLinkOut()
xout_video.setStreamName("video")
cam_rgb.video.link(xout_video.input)

monoLeft = getMonoCamera(pipeline, isLeft=True)
monoRight = getMonoCamera(pipeline, isLeft=False)
stereo = getStereoPair(pipeline, monoLeft, monoRight)

xout_disp = pipeline.createXLinkOut()
xout_disp.setStreamName("disparity")
stereo.disparity.link(xout_disp.input)

with dai.Device(pipeline) as device:
    device.setIrLaserDotProjectorIntensity(1200)
    device.setIrFloodLightIntensity(1200)

    video_queue = device.getOutputQueue(name="video", maxSize=1, blocking=False)
    disparity_queue = device.getOutputQueue(name="disparity", maxSize=1, blocking=False)

    maxDisparity = stereo.initialConfig.getMaxDisparity()
    disparityMultiplier = 255.0 / maxDisparity

    print("Starting video feed. Press 'q' to quit.")
    print("Press 's' to compare detected coordinates with reference and calculate differences.")

    while True:
        frame = video_queue.get().getCvFrame()
        color_frame = frame.copy()
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Perform template matching for TemplateROI_L
        result_l = cv2.matchTemplate(gray_frame, template_l, cv2.TM_SQDIFF_NORMED)
        min_val_l, max_val_l, min_loc_l, max_loc_l = cv2.minMaxLoc(result_l)
        top_left_l = min_loc_l
        bottom_right_l = (top_left_l[0] + template_l_width, top_left_l[1] + template_l_height)

        # Draw rectangle around the detected area for TemplateROI_L
        cv2.rectangle(color_frame, top_left_l, bottom_right_l, (0, 255, 0), 2)
        text_l = f"Match: {1-min_val_l:.2f}, X: {top_left_l[0]}, Y: {top_left_l[1]}"
        cv2.putText(color_frame, text_l, (top_left_l[0], top_left_l[1] - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        # Perform template matching for TemplateROI_R
        result_r = cv2.matchTemplate(gray_frame, template_r, cv2.TM_SQDIFF_NORMED)
        min_val_r, max_val_r, min_loc_r, max_loc_r = cv2.minMaxLoc(result_r)
        top_left_r = min_loc_r
        bottom_right_r = (top_left_r[0] + template_r_width, top_left_r[1] + template_r_height)

        # Draw rectangle around the detected area for TemplateROI_R
        cv2.rectangle(color_frame, top_left_r, bottom_right_r, (0, 0, 255), 2)
        text_r = f"Match: {1-min_val_r:.2f}, X: {top_left_r[0]}, Y: {top_left_r[1]}"
        cv2.putText(color_frame, text_r, (top_left_r[0], top_left_r[1] - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
```

```
        # Get and process the disparity map
        disparityFrame = getFrame(disparity_queue)
        if disparityFrame is not None:
            disparity = (disparityFrame * disparityMultiplier).astype(np.uint8)
            disparity = cv2.applyColorMap(disparity, cv2.COLORMAP_JET)

            # Draw the rectangles on the disparity map as well
            adjusted_top_left_l = (int(top_left_l[0] / 1.25), int(top_left_l[1] / 1.6))
            adjusted_bottom_right_l = (
                int(adjusted_top_left_l[0] + template_l_width / 1.6),
                int(adjusted_top_left_l[1] + template_l_height / 1.6)
            )
            adjusted_top_left_r = (int(top_left_r[0] / 1.25), int(top_left_r[1] / 1.6))
            adjusted_bottom_right_r = (
                int(adjusted_top_left_r[0] + template_r_width / 1.6),
                int(adjusted_top_left_r[1] + template_r_height / 1.6)
            )

            # Calculate the average depth within the regions defined by the rectangles
            avg_disp_l, depth_l = calculateAverageDepth(disparityFrame, adjusted_top_left_l, adjusted_bottom_right_l)
            avg_disp_r, depth_r = calculateAverageDepth(disparityFrame, adjusted_top_left_r, adjusted_bottom_right_r)

            # Display depth on the color frame
            if avg_disp_l is not None:
                cv2.putText(color_frame, f"Depth: {depth_l:.2f}m", (top_left_l[0], bottom_right_l[1] + 20),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

            if avg_disp_r is not None:
                cv2.putText(color_frame, f"Depth: {depth_r:.2f}m", (top_left_r[0], bottom_right_r[1] + 20),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

            # Draw adjusted rectangles on the disparity map
            cv2.rectangle(disparity, adjusted_top_left_l, adjusted_bottom_right_l, (0, 255, 0), 2)
            cv2.rectangle(disparity, adjusted_top_left_r, adjusted_bottom_right_r, (0, 0, 255), 2)

            cv2.imshow("Disparity Map", disparity)

        # Show the color frame with the rectangles and depth info
        cv2.imshow("OAK-D Lite - Template Matching with Depth", color_frame)

        key = cv2.waitKey(1) & 0xFF
        if key == ord('s'):  # Compare coordinates and calculate differences
            # Calculate differences for Left template
            diff_x_l = top_left_l[0] - REFERENCE_COORDS_L[0]
            diff_y_l = top_left_l[1] - REFERENCE_COORDS_L[1]
            print(f"Left Template - Detected: ({top_left_l[0]}, {top_left_l[1]}), "
                  f"Reference: {REFERENCE_COORDS_L}, Difference: (X: {diff_x_l}, Y: {diff_y_l})")

            # Calculate differences for Right template
            diff_x_r = top_left_r[0] - REFERENCE_COORDS_R[0]
            diff_y_r = top_left_r[1] - REFERENCE_COORDS_R[1]
            print(f"Right Template - Detected: ({top_left_r[0]}, {top_left_r[1]}), "
                  f"Reference: {REFERENCE_COORDS_R}, Difference: (X: {diff_x_r}, Y: {diff_y_r})")

        if key == ord('q'):  # Quit the application
            break

cv2.destroyAllWindows()
```

```cpp
#include <AccelStepper.h>

// Define stepper connections and interface type
#define STEP_PIN 4
#define DIR_PIN 3
AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);

bool motorRunning = false;
bool directionForward = true;  // Motor starts in forward direction
int motorSpeed = 1000;  // Initial speed

void setup() {
  Serial.begin(9600);  // Start serial communication
  stepper.setMaxSpeed(3500); // Maximum speed
  stepper.setAcceleration(5000); // Acceleration rate
  stepper.setSpeed(motorSpeed); // Initial speed
  Serial.println("Stepper motor control:");
  Serial.println("Commands:");
  Serial.println("s - Start/Stop the motor");
  Serial.println("f - Change direction to Forward");
  Serial.println("b - Change direction to Backward");
  Serial.println("+ - Increase speed");
  Serial.println("- - Decrease speed");
}

void loop() {
  if (Serial.available()) {
    char command = Serial.read(); // Read the incoming command

    // Start/Stop the motor
    if (command == 's') {
      motorRunning = !motorRunning;  // Toggle motor state
      if (motorRunning) {
        Serial.println("Motor started");
      } else {
        Serial.println("Motor stopped");
        stepper.stop();  // Stop the motor immediately
      }
    }

    // Change motor direction to forward
    if (command == 'f') {
      directionForward = true;
      stepper.setSpeed(motorSpeed); // Ensure speed is set correctly
      Serial.println("Motor direction set to Forward");
    }

    // Change motor direction to backward
    if (command == 'b') {
      directionForward = false;
      stepper.setSpeed(-motorSpeed); // Negative speed for reverse direction
      Serial.println("Motor direction set to Backward");
    }

    // Increase motor speed
    if (command == '+') {
      motorSpeed += 200;  // Increase speed by 200
      stepper.setSpeed(directionForward ? motorSpeed : -motorSpeed);  // Adjust speed and direction
      Serial.print("Motor speed increased to ");
      Serial.println(motorSpeed);
    }

    // Decrease motor speed
    if (command == '-') {
      motorSpeed = max(500, motorSpeed - 200);  // Decrease speed by 200, but not below 500
      stepper.setSpeed(directionForward ? motorSpeed : -motorSpeed);  // Adjust speed and direction
      Serial.print("Motor speed decreased to ");
      Serial.println(motorSpeed);
    }
  }

  // If motor is running, continue moving it
  if (motorRunning) {
    stepper.runSpeed();  // Run the motor at the set speed
  }
}
```