FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

MASTER'S THESIS

CODESIGN OF EDGE INTELLIGENCE AND AUTOMATED GUIDED VEHICLE CONTROL

Author	Gallage Malith Thiwanka
Supervisor	Asst. Prof. Sumudu Samarakoon
Supervisor	Dr. Rafaela Scaciota
Technical supervisor	Prof. Mehdi Bennis

December 2022

Gallage Malith Thiwanka (2022) Codesign of Edge Intelligence and Automated Guided Vehicle Control Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering, 36 pages.

ABSTRACT

In recent years, edge Artificial Intelligence (AI) coupled with other technologies such as autonomous systems have gained a lot of attention. This work presents a harmonic design of Autonomous Guided Vehicles (AGV) control, edge intelligence, and human input to enable autonomous transportation in industrial environments. The AGV has the capability to navigate between a source and destinations and pick/place objects. The human input implicitly provides the preferences of the destination and exact drop point, which are derived from the AI at the network edge and shared with the AGV over a wireless network. Design and integration of autonomous control of AGV, edge intelligence, and communication therein are carried out in this work and presented as a unified demonstration. The demonstration indicates that the proposed design of hardware, software, and intelligence design achieves the Technology Readiness Level (TRL) of range 4-5.

Keywords: Edge AI, Image Processing, Autonomous Navigation

CONTENTS

PREFACE

This master's thesis titled "Codesign of edge intelligence and automated guided vehicle control" was conducted at the Center of Wireless Communication at the University of Oulu as a part of the master's degree program in Wireless Communication Engineering. I hope that the results and insights presented in this thesis will be useful and informative to the reader. I would like to begin by expressing my heartfelt gratitude to my primary supervisor, Asst. Prof. Sumudu Samarakoon, and my secondary supervisor, Dr. Rafaela Scaciota. Their guidance, support, and expertise have been invaluable throughout my master's program and the development of this thesis. Also, I would like to take this opportunity to extend my appreciation to Prof. Mehdi Bennis for bringing me under the wings of the ICON group which was instrumental in my growth as a researcher. I would also like to thank my family and friends for their unwavering love and support. I am especially thankful to Nimni Tharuni, for being my rock and my source of strength during the long hours of hard work. I must also acknowledge the incredible opportunities that the University of Oulu and my research group have provided me. The resources, facilities, and support that they have given me have been invaluable in allowing me to pursue my passion and achieve my goals. Finally, I would like to take this opportunity to thank everyone who has played a role in my journey. This thesis is as much a tribute to them as it is to me. Thank you for everything.

Malith Gallage

LIST OF SYMBOLS AND ABBREVIATIONS

AGV	Autonomous Guided Vehicles.
AI	Artificial Intelligence.
DoF	Degrees of Freedom.
fps	Frames per Second.
GPIO	General Purpose Input/Output.
HITL	Human-in-the-Loop.
IoT	Internet of Things.
ML	Machine Learning.
PID	Proportional-Integral-Derivative.
px	Pixels.
RGB	Red, Blue and Green.
RoI	Region-of-Interest.
TRL	Technology Readiness Level.

1 INTRODUCTION

The rapid growth of customer demands and the increasing costs of resources, labour, and energy have driven industries to seek new technologies that improve productivity and efficiency [1]. In particular, the modern logistics industry is an ever-evolving sector and one of the key players to adopt and leverage the power of autonomous automation technologies including the Human-in-the-Loop (HITL), in which, human operators can provide the preferences required for automation [2]. With the introduction of Autonomous Guided Vehicles (AGV), it is expected to bring about revolutionary changes to the logistics industry as AGVs are one of the pivotal components that is essential to enable "smart logistics" in manufacturing plants [3]. Realizing full/semi-autonomy with the fusion of control systems, communication networks, and computation servers at the edge over repetitive tasks naturally calls for the concepts of Artificial Intelligence (AI) [4]. While using AI to automate a wide variety of tasks and processes, such as route optimization and warehouse resource management, on the other hand Edge AI facilitates the processing at the edge itself leading to low latency, low bandwidth inference while preserving privacy which becomes an ideal candidate for industrial application in logistics [5].

With the increasing interest in utilizing AGVs in industrial applications, there is a growing need for developing AI-driven efficient and reliable solutions for navigation tasks [4]. Towards this, line-following robots have been at the center of attention due to their ease of use and low complexity and robust operations [6]. The benefits of using AGVs in the logistics industry are numerous. First, they can improve efficiency and accuracy by reducing the need for manual labour. AGVs can be programmed to follow predetermined paths and perform specific tasks without the need for human intervention, which can save time and money. Furthermore, AGVs can provide better tracking and control of goods as they can be programmed to monitor and record the progress of their tasks. But their use is still limited due to the potential safety risks that are associated with autonomous navigation. In order to minimize these risks, many organizations are beginning to incorporate the use of HITL systems whereby providing an additional layer of safety and security with the use of a human operator to monitor and control robotic operations. This allows companies to safely deploy AGVs in their operations while minimizing the risk of accidents or other unforeseen events by having the best of both worlds of AGVs and HITL systems. Overall, the introduction of AGVs, HITL, and edge AI is likely to usher into a new era of manufacturing and logistics industry.

In this work, we present the co-design of an intelligent crawler robot that uses its camera to sense the environment and navigate to deliver objects to their corresponding locations with the aid of an edge server with AI capabilities. The preferences for the destination and exact drop point are set by a human operator, allowing the robot to accurately meet these preferences. This co-designed system demonstrates the potential for intelligent robots to perform tasks in industrial environments with a high degree of accuracy and efficiency while having the human-in-the-loop.

1.1 Background study and literature review

The sudden rise of AI has spurred a significant amount of research on the potential applications of edge AI for automating tasks across a range of industries [7–9]. In addition to its potential for improving task automation, the integration of edge AI with other technologies such as the Internet of Things (IoT) and robotics has the potential to expand its capabilities and enable new applications [9, 10]. In this section, we will delve into several key concepts that were used in this study including edge AI, supervised learning and U-net architecture [11] for image classification and Proportional-Integral-Derivative (PID) controller [12] for guided navigation [13], providing explanations and elaborations.

Edge AI refers to the implementation of AI algorithms and technologies at the edge of a network, closer to the source of data, in order to enable real-time analysis and decision making without the need

for extensive data transfer. This approach has the potential to improve the efficiency and speed of task automation, particularly in industries where large amounts of data must be processed quickly [14, 15]. One key advantage of edge AI is its ability to reduce the amount of data that must be transferred across networks, which can improve the speed and efficiency of task automation. By performing data analysis and decision making at the edge of the network, edge AI can reduce the need for extensive data transfer, which can be time-consuming and potentially introduce latency [16]. This is particularly important in industries where real-time analysis and decision making are critical, such as manufacturing and transportation [17]. Edge AI also has the potential to improve the security of data and systems. By performing data analysis and decision making at the edge of the network, edge AI can reduce the risk of data breaches and other security threats. [18] This is particularly important in industries where data security is critical, such as healthcare and finance [14]. Thus, the use of edge AI has the potential to improve the efficiency and speed of task automation and holds significant promise in various industries.

Machine Learning (ML) is the enabler of artificial intelligence that involves the use of algorithms to analyze and learn from data, without being explicitly programmed. Supervised learning is a type of machine learning algorithm that uses labeled training data to learn to predict outputs for new data. It is a powerful tool for image classification because it allows a system to learn complex patterns and relationships in the data. By training a supervised learning model on a labeled dataset of images, the model can learn to identify and distinguish different objects, scenes, and other visual concepts that may not be easily identifiable using rule-based image processing techniques. This is achieved through the use of algorithms that interactively adjust the model's parameters to minimize the difference between the predicted labels and the true labels in the training data.

The U-Net architecture is a deep learning model that has been widely used for image classification and segmentation tasks [11]. The U-Net architecture is composed of two parts: a contracting path and an expanding path. The contracting path consists of a series of convolutional layers that extract features from the input image, while the expanding path consists of a series of deconvolutional layers that combine these features to predict the final label or segmentation map. The U-Net architecture is notable for its use of skip connections, which allow the model to combine low-level and high-level features to improve the accuracy of its predictions. The combination of supervised learning and the U-Net architecture has been shown to be effective for image classification tasks. Many researchers have demonstrated the effectiveness of these approaches for tasks such as object recognition and medical image analysis where segmentation is needed [19].

The PID controller is a widely used control system that has been applied to a variety of tasks, from industrial control systems to navigation in line following robots. PID Controllers are a type of feedback control mechanism that uses feedback from the environment to regulate the behaviour of a system. A PID controller consists of three components, a proportional term, an integral term, and a derivative term. The proportional term is used to regulate the magnitude of the output, the integral term is used to regulate the rate of change of the output, and the derivative term is used to regulate the acceleration of the output. These terms work together to provide precise and stable control of a system, by constantly adjusting the control signal based on the error between the desired and measured states of the system.

PID controllers are used in the navigation and control of robots in terms of providing the necessary control decisions to achieve their desired position, velocity, and/or acceleration. In the context of line-following robots, a PID controller can be used to control the motion of the robot along a predefined path, by constantly adjusting the control signal based on the error between the desired and actual position of the robot such as the speed of its motors or the position of its joints. This allows the robot to follow the desired path accurately and reliably, even in the presence of noise and other disturbances. A detailed explanation for the use of PID controllers in this work is provided in later sections.

1.2 Research objectives and thesis organization

In this work, we are evaluating the suitability of the co-design of the edge intelligence with an AGV to automate tasks to aid in logistics by building a system prototype that could be aided by a human operator to provide preference. The system prototype is realized from its conceptual design to the actual implementation in this work. The designed system is incorporated with ML models and control algorithms to help the AGV carry out its tasks accurately and efficiently. The preliminary results of this work are further used to refine the design of the edge intelligence, AGV, and their communication aspects.

The rest of this thesis is organized as follows. The second chapter contains an overview of the proposed system architecture and the communication architecture of the various components of the proposed system design. The third chapter elaborates on the used hardware as well as provides an extensively detailed explanation of the implantation of the two key software solutions that work as the decision maker. Finally, the fourth chapter summarizes the work carried out in this thesis and provides recommendations for potential future extensions of this work.

2 PROPOSED SYSTEM ARCHITECTURE

2.1 Overall System Architecture

In order to demonstrate the capabilities of the AGV and the intelligence provided by the services that are deployed on the edge server, we have built the robot platform as shown in Fig. 2.1. The robot platform and its components are utilized to emulate a logistics management scenario in a warehouse where objects are to be delivered from a source to a destination and to be placed accurately on designated positions.

The robot platform consists of a few key components, which are (1) routes that span from the source point to multiple destinations, (2) an AGV that picks objects from the source and places them on destinations, (3) a camera observing the destinations with a bird's eye view, and (4) an edge server equipped with computing capabilities. The routes are black lines drawn in the robot platform from the source to four of the pre-defined destinations. The AGV uses an onboard camera to sense the routes and the surrounding environment to determine its control decisions and navigate from source to destinations.

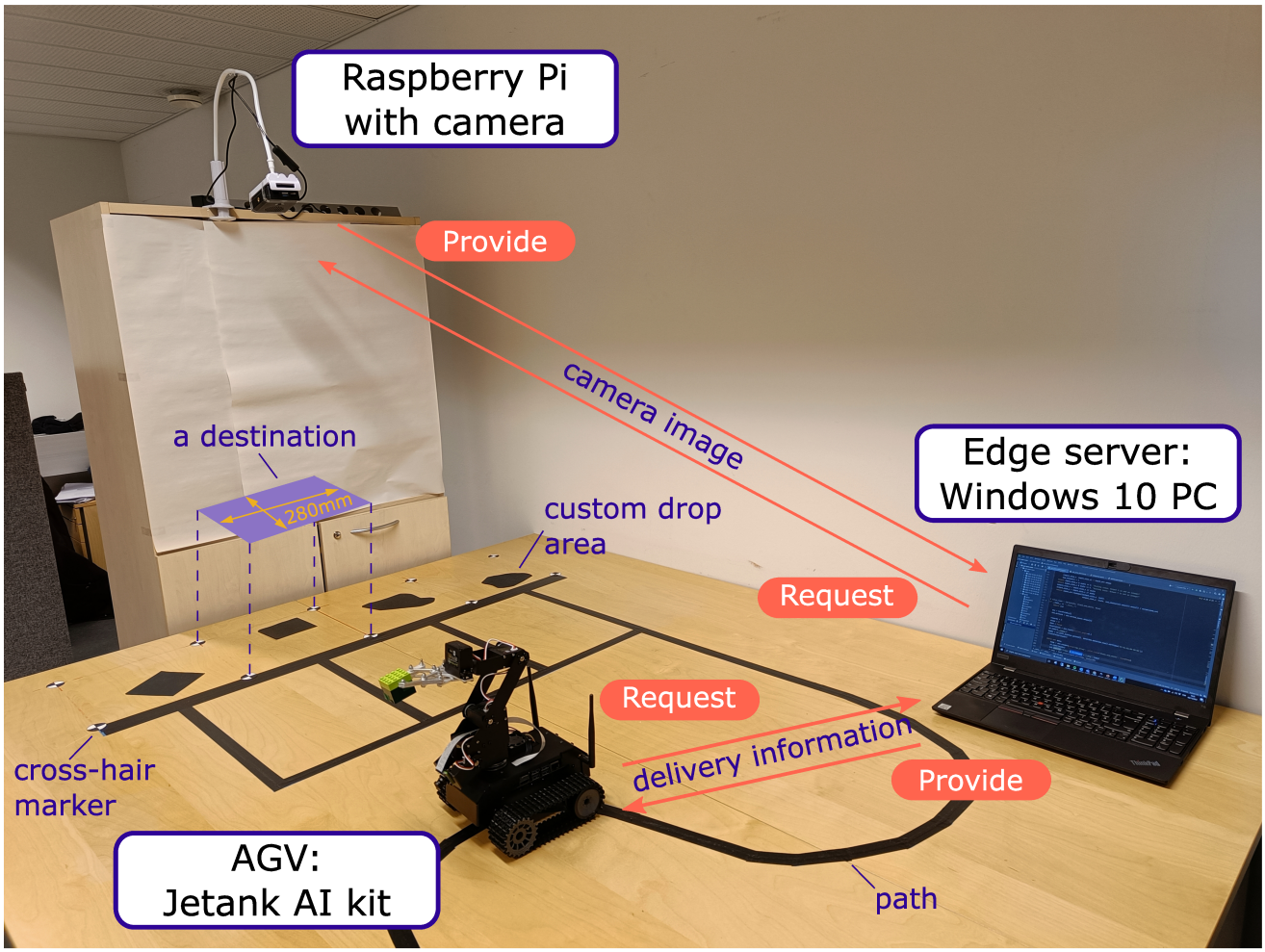


Figure 2.1. The robotic platform highlighting the components and wireless connectivity.

The source is the location where the AGV picks up the object that defines at the beginning of the route and it remains static during the demonstration. This is referred to as *source* hereinafter. When the

AGV picks up the object from the source, the next object is placed by the operator on the source for the next delivery task.

The destination and the precise drop location of the object referred to as *delivery information* hereinafter, are determined by the edge server using the ML inference service running on it with the help of the camera images. Derived delivery information is to be shared with the AGV upon its request before the beginning of each delivery task. The delivery information is derived based on the shape of the custom drop areas defined by an external party (e.g., a human operator) and the location of these drop areas.

The path between the source and the destinations was defined to aid the AGV's navigation and it is drawn on the surface with black color, and it contains curves and junctions. Here, a single path starting from the source branches out to four different paths leading to four adjacent destinations. These four adjacent destinations are defined as *storage areas* and the drop location of the object is inside of these storage areas in each delivery task. Each destination is a square of width 280 mm and the corners of the area are marked by cross-hair markers. Altogether 10 cross-hair markers have been used to mark 4 adjacent storage areas as shown in Fig. 2.1 with the furthest marker at the bottom left being defined as the origin of the 2D coordinate system. Custom drop locations are set as irregular shapes made by an external party which is located inside the storage areas. The precise drop location is defined as the center point of the largest circle placed inside the custom drop area. It should be noted that the largest circle is not necessarily a unique circle. Also, it is worth highlighting that the drop point coordinates provided by the edge server need to be computed with respect to the actual coordinates whereas the camera images used by the edge server, have their measurements in Pixels (px). Thus, the placement of cross-hair markers is predefined and this knowledge is utilized to translate the px distances to the actual measurements during the image processing at the edge server. Then the translated coordinates can be utilized by the AGV to move its robotic arm to the correct position along the plane of the robot platform and place the object in the desired position.

2.2 Communication Architecture

All the components mentioned in the proposed architecture are connected to the same access point via WiFi, thus, enabling them to communicate and coordinate with each other to achieve their final goal. The network architecture of the proposed system is illustrated in Fig. 2.2 where the AGV, camera observing the destinations and the edge server are connected to the same WiFi network.

The communication between the AGV and the edge server takes place through a REST API over the WiFi network. The AGV requests delivery information from the edge server only once, before each transport job is initiated. The edge server provides delivery information in JSON format via its exposed endpoint and an example of the received delivery information is provided in Listing 1.

Listing 1 JSON response from the edge server.

```
{
    "confidence": 95.86259095408846
    "radius-mm" : 47.84779950254917,
    "xcoordinate" : 100.0340522133938,
    "ycoordinate" : 228.59250851305333
}
```

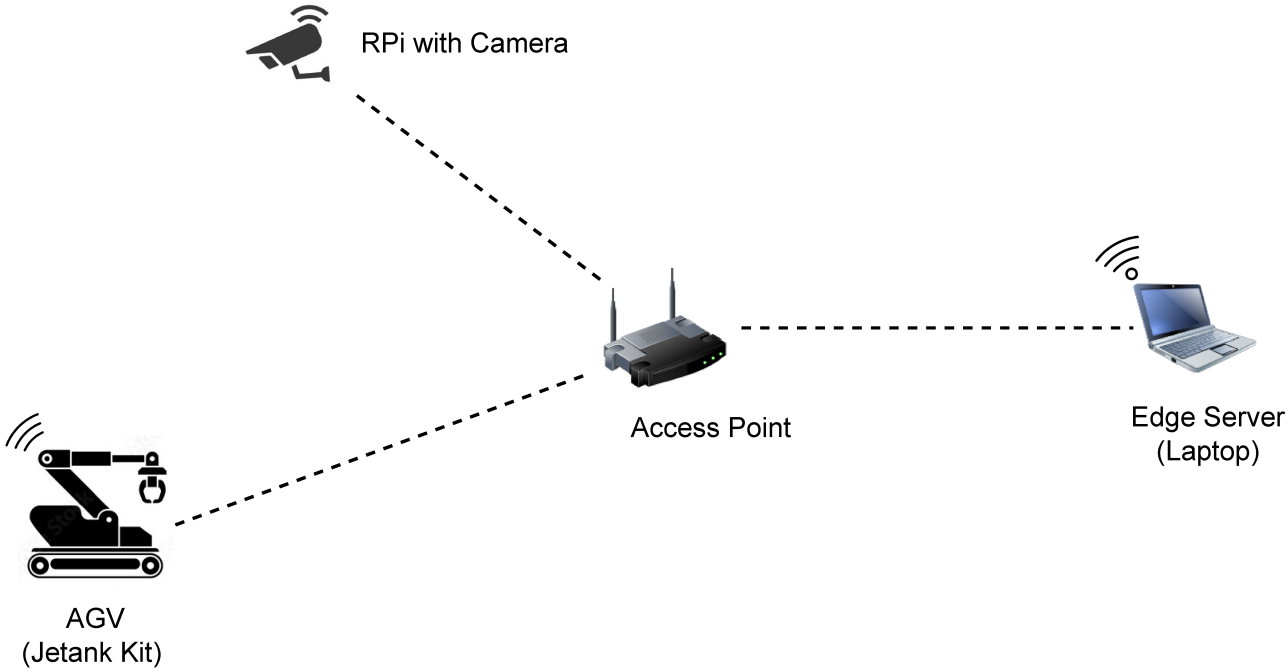


Figure 2.2. Network architecture of the components.

The received delivery information contains the coordinates for the precise drop location and a confidence value to measure the accuracy of the coordinates calculated by the edge server. After receiving the delivery information successfully, and evaluating the confidence value where it should be higher than a predefined threshold to assure high accuracy, the AGV proceeds to pick up the object and continue the rest of the transport job, thus ensuring an accurate placement of the object. After completing the delivery job successfully, the AGV returns to the source and repeats the procedure.

In order to derive delivery information, the edge server obtains the bird's eye view of the destinations by accessing the camera. The camera is associated with a web server running on the Raspberry Pi and it is capable of serving the images of the destinations areas through WiFi upon request, as a byte stream via its exposed REST API endpoint. The served image will be of a resolution such that the edge server is able to identify the cross-hair markers.

3 SYSTEM DESIGN AND IMPLEMENTATION

The demonstration setup for this project consists of various hardware components and two primary software solutions that provide intelligence for navigation and object placement. The hardware components include sensors such as a camera for perception, actuators such as servos for object manipulation, and an AGV that interacts with the environment. The software solutions, on the other hand, provide the intelligence for the demonstration setup in terms of algorithms and models for navigation and object placement. The intelligence for navigation solution enables the system to move through its environment. The intelligence for object placement solution, on the other hand, provides information about the location of objects which enables the system to pick up and manipulate objects, placing them in specific locations.

In the following sections, we provide a comprehensive overview of the hardware components and software solutions that make up the demonstration setup, as well as a detailed explanation of their implementation.

3.1 Hardware

All the hardware components that are used in the demonstration are shown in Fig. 2.1. AGV is an off-the-shelf mobile crawler robot known as "Jetank AI kit" which is powered by Nvidia Jetson nano developer module with 16GB eMMC and 4GB RAM [20]. This is capable of running resource-demanding modern computing algorithms related to machine learning and computer vision and supports many popular libraries and frameworks. The robot has two motors that could be controlled individually and a 4-Degrees of Freedom (DoF) mechanical arm with a gripper. The only sensor it has to sense the surrounding environment is a robust 8MP wide-angle camera with 160° field-of-view and it comes with a Sony IMX219 image sensor. The camera can provide static images up to 3280×2464 px resolution. With the Jetson Nano board's General Purpose Input/Output (GPIO) pins there is a possibility to integrate different types of sensors, but, for this demonstration we have only utilized the onboard camera.

The camera which observes the destinations consists of a Raspberry Pi V2 camera module [21] which comes with a robust 8MP Sony IMX219 image sensor and connects with a Raspberry Pi 4 Model B computer [22], which hosts a web server that serves high resolution images of the storage area upon the requests from the edge server. This camera is able to provide static images up to 3280×2464 px resolution and it has a manually adjustable focal length.

A powerful multi-purpose 64-bit Windows 10 computer acts as the edge server and it hosts the AI service and share the delivery information with the AGV upon request.

3.2 Intelligence For Navigation

This section provides a detailed explanation of the software solution that was developed to provide intelligence for the AGV's navigation. As the only sensor in the AGV is the onboard camera, we have implemented a vision-based line-following system to successfully navigate around the platform. The reason for depending solely on the onboard camera rather than opting into a more robust method by fusing other sensor feedback together was the ease of implementation of the vision-based line following system and the reproducibility of this system in other industrial AGVs by using merely a camera.

The camera and the Jetson nano board are able to process images with a resolution of 300×300 px up to 30 Frames per Second (fps). It can process images with a higher resolution while maintaining 30 fps, but, for the work illustrated in this document we are using the images with the said resolution for faster processing. In the initialization phase, both the robot arm and the camera orientation are set to a position where it provides an obstruction-free view of the path. During the navigation, the following three steps are repeated for each camera frame to derive control commands.

- Acquiring camera images.
- Image pre-processing and line detection.
- Actuating control decisions of the AGV.

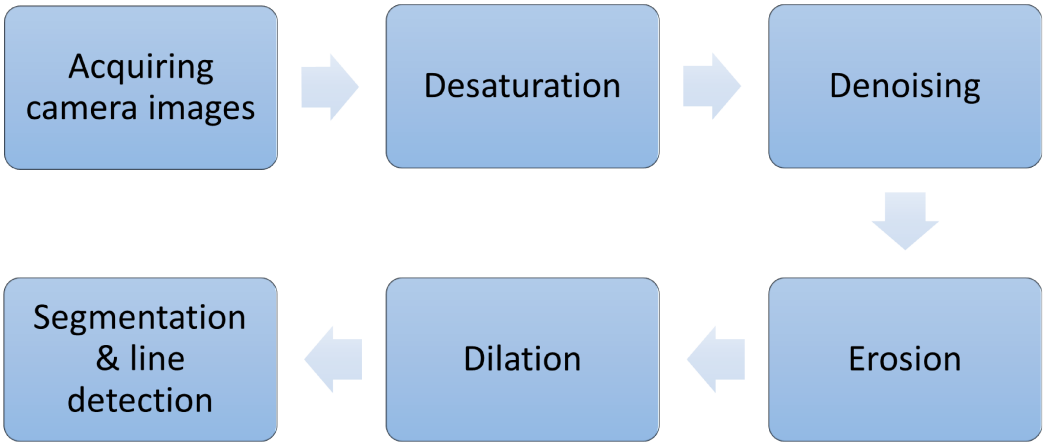


Figure 3.1. The key steps of image pre-processing.

The flow chart presented in Fig. 3.1 illustrates the first two steps of the above list which are acquiring images and image pre-processing to detect lines of the path. The implementation of these steps is explained extensively in next sections.

As illustrated in Fig. 3.1, in the first step, the camera acquires images of the environment in front of the AGV at 30 fps. The raw input from the camera is in the Red, Blue and Green (RGB) color space. As given in the second step, the acquired images are desaturated and converted into gray-scale as we

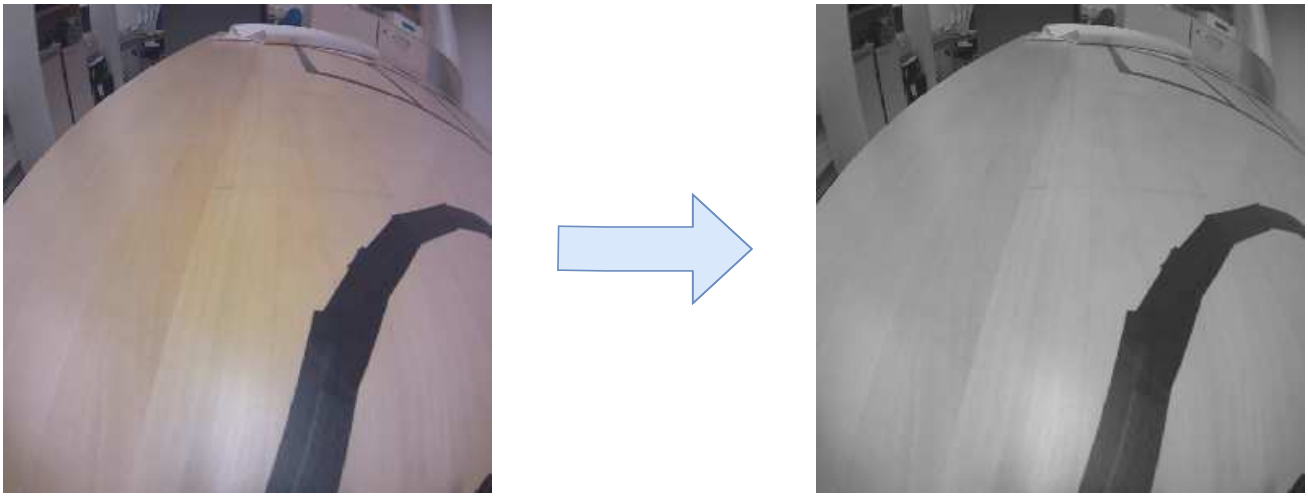


Figure 3.2. Raw image to gray scale conversion.

are only interested in the structure and the intensities of the image rather than using 3 color channels. The gray-scale conversion is done as [23],

$$Y = 0.299 \cdot X_R + 0.587 \cdot X_G + 0.114 \cdot X_B \tag{3.1}$$

where X_R, X_G and X_B denote the red, green, and blue color intensity values respectively. A sample frame of the raw image and its gray-scale converted image is shown in Fig. 3.2.

In the next step, the unwanted Gaussian noise is removed from the image by convoluting the gray-scale image using a 3×3 Gaussian kernel. After this de-noising process, the image will appear smoothened and less noisy as illustrated in Fig. 3.3. The reason for resorting to a 3×3 Gaussian kernel was, the details of the image were declining when the kernel size was higher. Therefore, a 3×3 Gaussian kernel was used to remove the Gaussian noise of the image while preserving other details of the frame. The 3×3 Gaussian filter is constructed using the 2-dimensional Gaussian distribution [24],

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2}$$

and using a discrete approximation to it.

Even after the de-noising process, small-scale inconsistencies can be observed in the detected path mainly due to the glare of the surrounding light sources. In order to remove those undesired white spots on the path, we apply one of the gray-scale morphological operators known as gray-scale erosion (see step 4 in Fig. 3.1). The mathematical expression for the gray-scale erosion is expressed as follows [24]:

$$(f \ominus b)(x) = \inf_{y \in B} [f(x + y) - b(y)] \tag{3.3}$$

where $f(x)$ is the given gray-scale image and B is the space where structuring element $b(x)$ is defined. This is an extension of the binary morphology operator erosion. The structuring element we have used here is a 3×3 kernel with value 1 and the erosion process is applied 8 times interactively. The resulting image after applying the erosion process is illustrated in Fig. 3.4.

Erosion helps to remove bright objects smaller than the structuring element. This can be confirmed from the resulting image in Fig. 3.4 that the small bright areas which were on the path are removed. But, as a result of erosion, darker regions of the image have expanded and the image appears darker. To avoid these undesired dark regions and to preserve the shape and the structure of the path, the next gray-scale

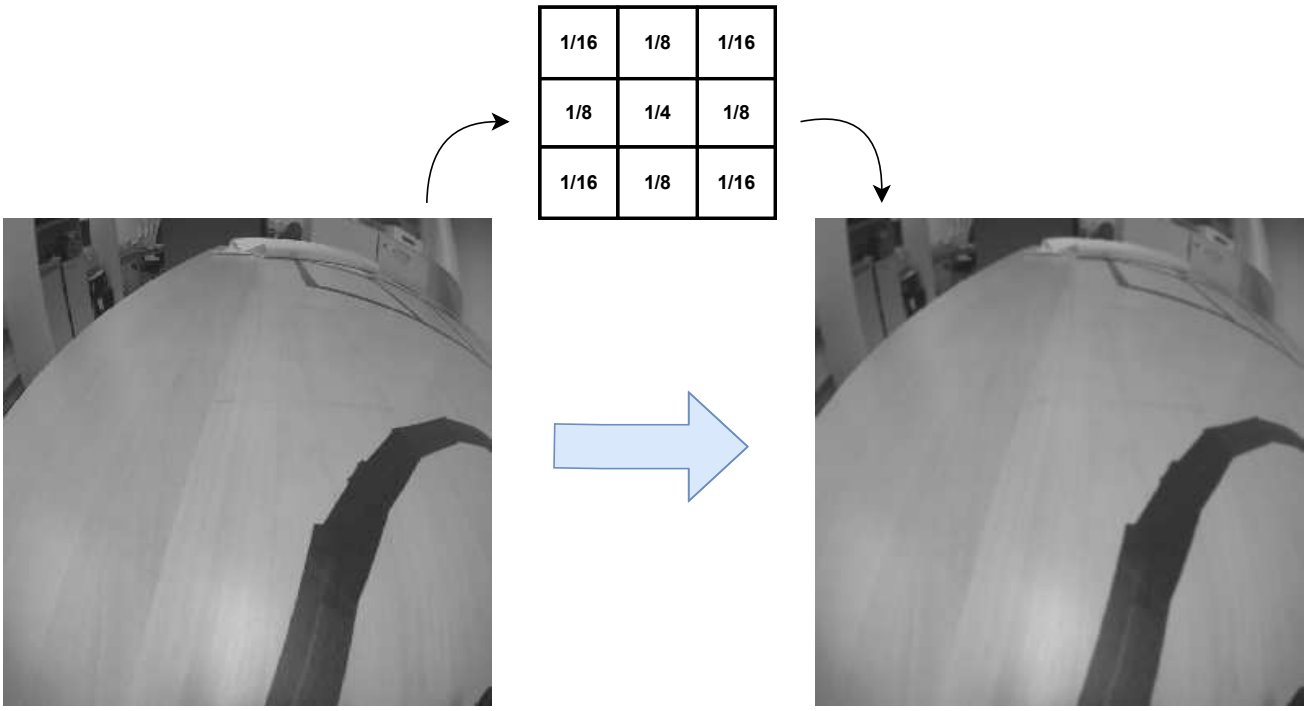


Figure 3.3. De-noising gray-scale image.

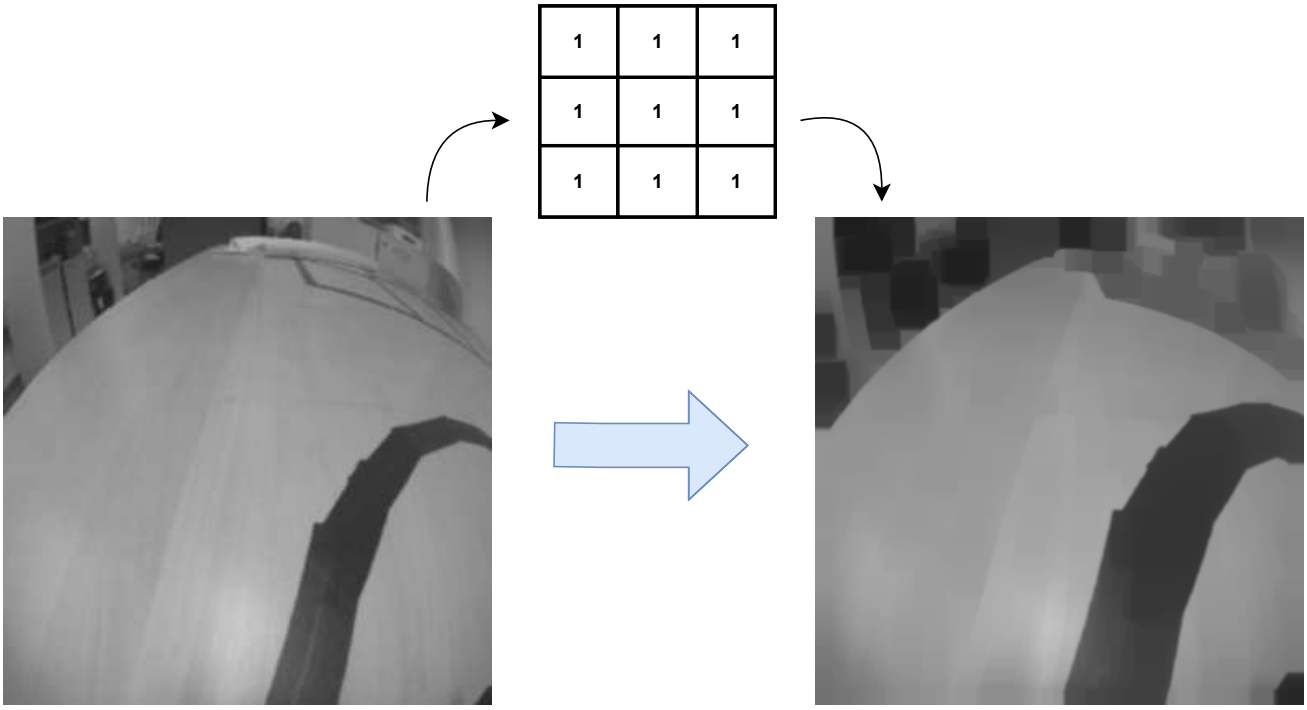


Figure 3.4. Result after erosion.

morphological operator, dilation is applied (see step 5 in Fig. 3.1). The mathematical expression for gray-scale dilation is given by [24]

$$(f \oplus b)(x) = \sup_{y \in E}[f(y) - b(x - y)], \tag{3.4}$$

where $f(x)$ is the given gray-scale image which is in the Euclidean space E and $b(x)$ is the structuring element. This is an extension of the binary morphology operator dilation. A 3 kernel with value 1 has been used as the structuring element and it has been applied 5 times interactively to remove the undesired outcome of erosion. Dilation causes the image to appear brighter and to extend bright regions. The resulting image after dilation can be seen in Fig. 3.5 and it can be observed that the size and the structure of the path have been restored closer to its original size. The combination of the dilation and erosion such that dilation is performed on the result of the eroded image by the same structuring element is known as morphological opening.

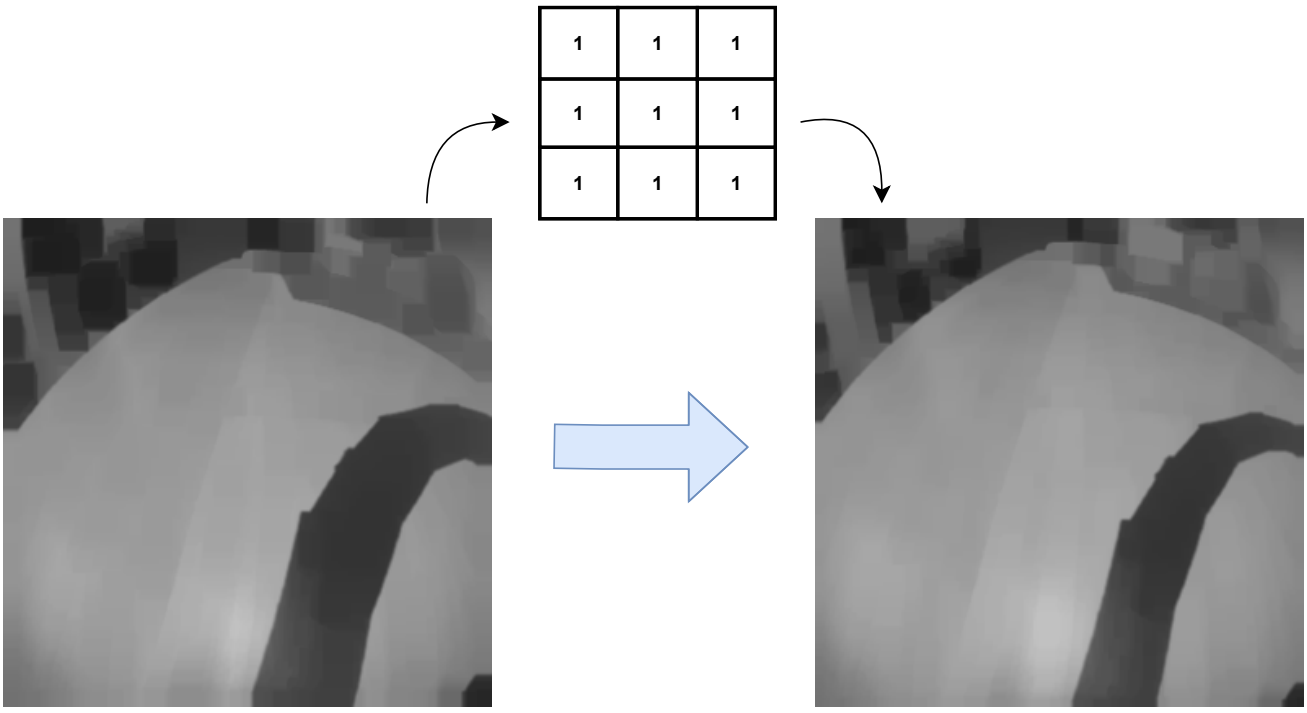


Figure 3.5. Result after dilation.

The next step is image segmentation to identify the path from the rest of the background. For this, the following two methods were evaluated:

1. Binary thresholding.
2. OTSU method [25].

Due to the simplicity of the binary thresholding method, it was first used to segment the image [26]. But as the AGV moves, the intensities of the frame rapidly change due to the surrounding environment, different light sources, shadows, and uneven illumination. Hence, having a constant threshold value was not successful as seen in Fig. 3.6.

Hence, to make the segmentation more robust, the OTSU method was evaluated, as it dynamically calculates the thresholding value depending on the intensity values. In the OTSU method, it assumes that the intensity distribution of the image follows a bimodal distribution [27]. The optimal threshold value corresponds to the value which minimizes the intra-class variance of the two pixel groups that are separated by the threshold value. The histogram of the dilated image approximately follows a bimodal distribution as shown in Fig. 3.7 if there are no drastic changes in the illumination. Hence, the OTSU method was chosen to segment the image.

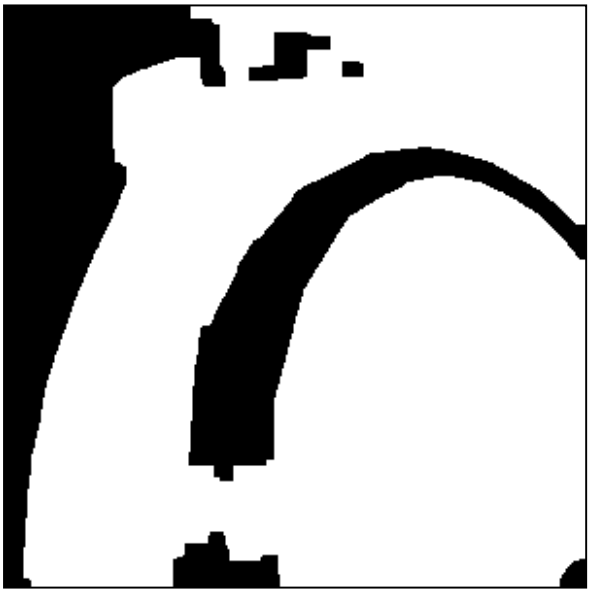


Figure 3.6. An example of a failure in binary thresholding.

The resulting binary image after applying the OTSU method is illustrated in Fig. 3.8. In the binary image, the path appears in black while the background appears in white. However, due to the low illumination in the horizon of the image, it appears in black as well. To neglect the undesired black areas corresponding to the horizon, a rectangular window that is likely to contain the path has been defined as the *Region-of-Interest (RoI)* (see Fig. 3.10). If the detected lines are within the RoI, it proceeds to issue the control commands.

After detecting the line, the next step of the navigation would be to issue control commands. To ensure the AGV stays on track and to ensure a smooth navigation, a PID controller [12] is used to produce control commands.

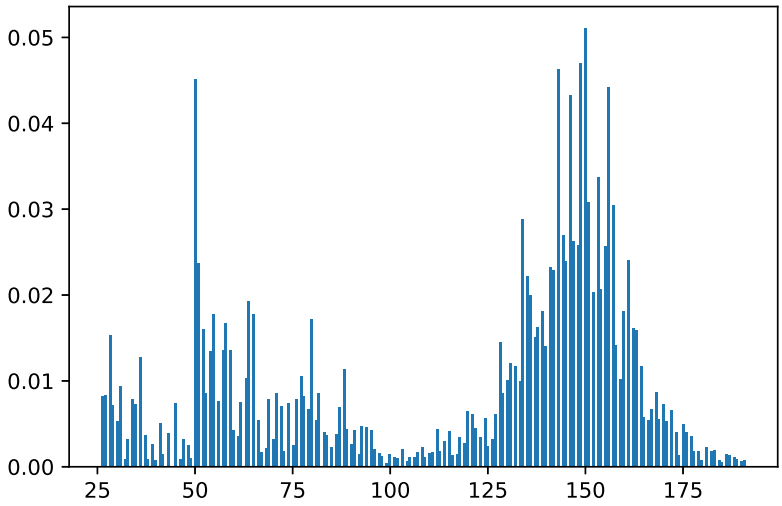


Figure 3.7. Histogram of the eroded image.

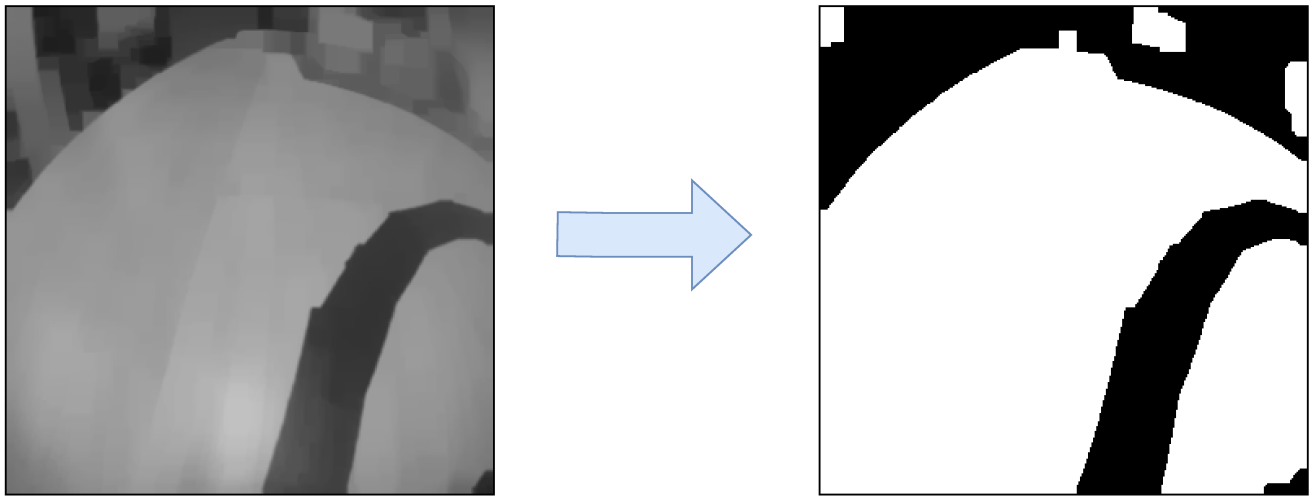


Figure 3.8. Result after thresholding.

PID controller is widely used in the industry and it has been tested rigorously in process control systems in which it has been proven to be a reliable method for automatic control [28]. It uses proportional, derivative and integral control terms to regulate the control function to achieve accurate and optimal control. In the PID controller, it continuously monitors the measured process variable $y(t)$ against its desired set point $r(t)$ and calculates the error term $e(t)$ as [29],

$$e(t) = r(t) - y(t) \tag{3.5}$$

The control function is defined with the proportional, derivative and integral terms [29],

$$u(t) = K_p e(t) + K_i e(t) \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{3.6}$$

where the controller applies a correction to the process depending on the control function output such that the error is minimized overtime. The control process is graphically illustrated in Fig. 3.9

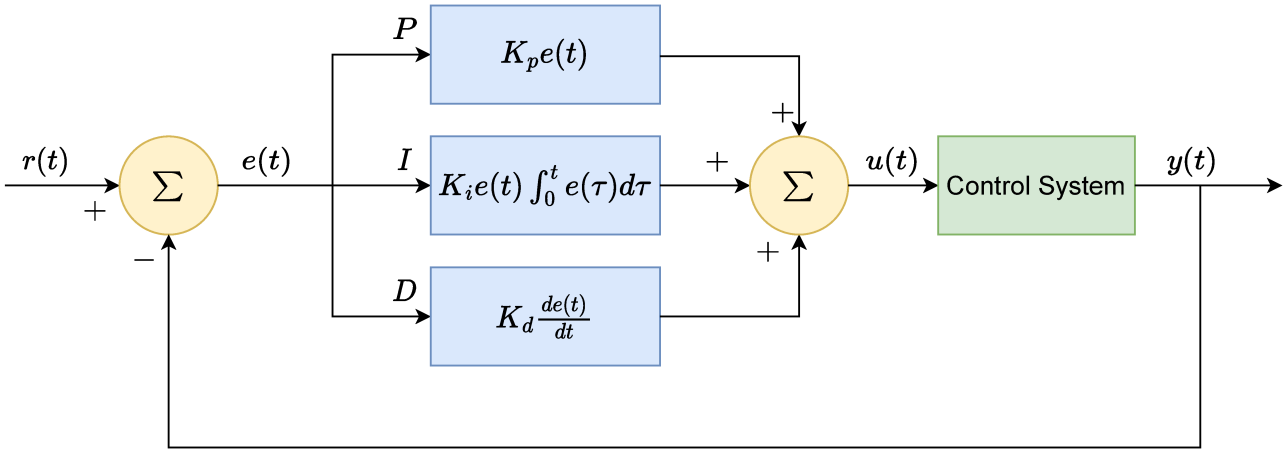


Figure 3.9. Block diagram of the PID controller.

The PID controller is implemented in the AGV by obtaining the output of the control function which

is referred to as *pid value* hereinafter. This is calculated in each image frame and applied to control the angular velocities of the left and right DC motors separately. The measured process variable in the AGV is the coordinates of the center of mass of the RoI in the segmented binary image. The center of mass is calculated by assigning weights ones and zeros to black and white px, respectively, and referred to as the *center of the path* hereinafter. The desired set-point is defined as the vertical symmetrical axis of the image. Then the *error* term is produced as the displacement of the center of the path from the desired set point and it is fed to the control function after normalizing. The constructed *RoI*, the *center of path* and the *error* is shown in Fig. 3.10 for reference. Then the *pid value* is added to the base speed of the left DC motor whereas it is reduced by the base speed of the right DC motor to regulate their angular velocities such that it attempts to minimize the *error* in the next iteration and over time, controller converges into the steady state where the error term is nullified and the center of path coincides with the vertical symmetrical axis of the image frame. Thus, the AGV navigates smoothly within the path. This control procedure is repeated on each image frame in such a manner it ensures that the AGV stays within its course of navigation.

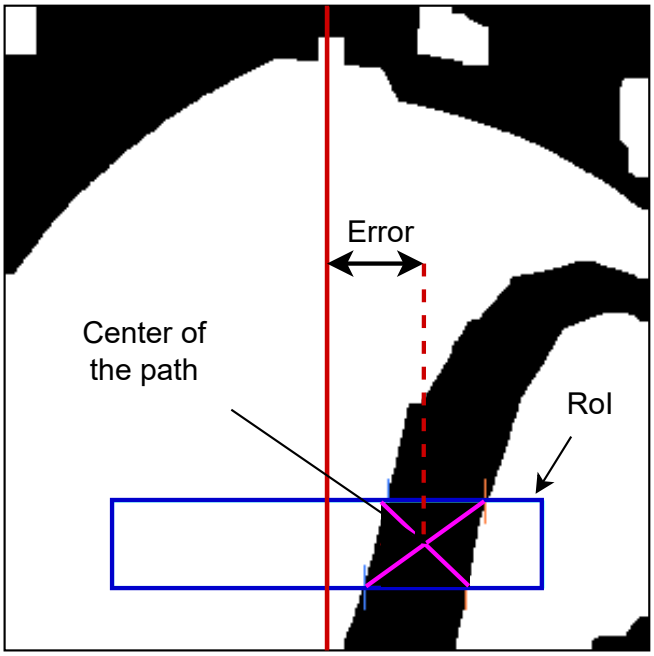


Figure 3.10. Region of interest, center of mass and error calculation.

In order to achieve the optimal control function, the PID controller should be tuned to have a balance of the proportional, derivative and integral terms. Initially, the coefficient values of proportional, derivative and integral terms which are K_p, K_d and K_i, respectively, set to an arbitrary value and tested the AGV on the laboratory environment to observe its control function response in (3.6). Then by trial and error sub optimal control function coefficients were obtained which provided a smoother navigation of the AGV. The obtained K_p, K_d and K_i values are 1,1 and 0 respectively for the AGV which was tested on the platform shown in the Fig. 2.1 and for the hardware that was specified under Sec. 3.1. If this system is to be replicated in another environment or another type of AGV, the PID controller coefficients should be tuned accordingly.

The robot platform consists of several types of T-intersections and those intersections are labelled according to the way they are perceived by the AGV, which are namely, *T junction*, *Left junction* and *Right junction*. The images of these junctions perceived by the AGV is shown in the Fig. 3.11. From

Figure 3.11. Different types of junctions.

the last step in image pre-processing pipeline, a junction can be identified and its type can be decided by analyzing the boundaries of the RoI. For instance, if the lines are within the RoI and the majority of the pixels in top left boundary of the RoI are black while the majority of the pixels in the top right boundary of the RoI are white, then the type of the junction is derived as a *left junction*. Fig. 3.13 shows the types of junctions and the corresponding RoI which is used to derive the type of the junction.

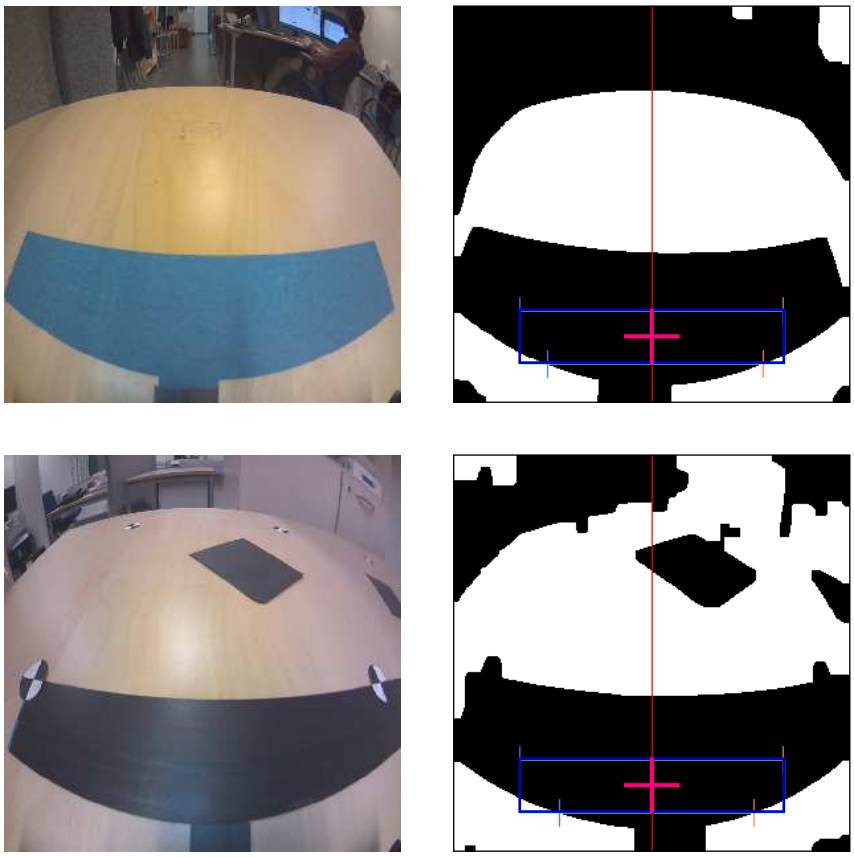


Figure 3.12. Terminal points with the corresponding RoI.

When the AGV encounters a junction, its PID controller might not work as expected as it cannot utilize the standard line following method. Hence, an alternative control procedure has been introduced to navigate around intersections. With the ability to identify and classify junctions, when the AGV

encounters the junction, the controller overrides the PID algorithm and switches to the alternative control procedure that relies on the type of the junction and the knowledge on the direction of moving and the delivery information. Based on the current goal as per the delivery information, a predefined sequence of control commands are issued to turn the AGV by 90° or 180°. After turning the AGV at the junction, the control is handed over to the PID algorithm to continue the navigation on the line.

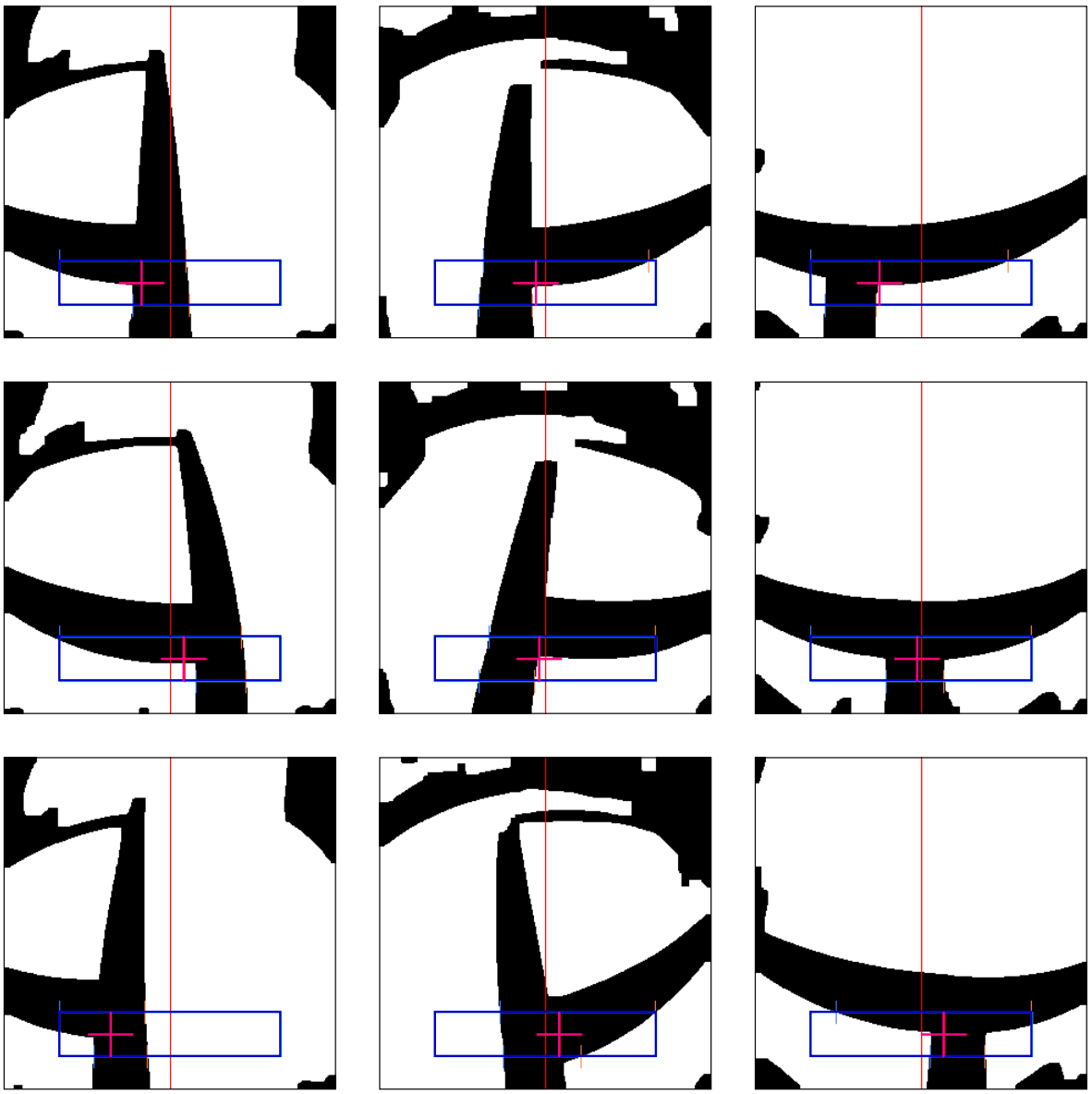


Figure 3.13. Junction identification with the corresponding RoI.

The robot platform consists of terminal points which help the AGV to identify source and destination areas. These terminal points can be identified by analyzing the RoI by following the same procedure which was used to identify and classify junctions. Terminal points are created such that its thickness is larger than the thickness of the path as it could be differentiated from a junction. The AGV identifies terminal points by analyzing the RoI where line is within the RoI and the majority of the pixels in the top and bottom boundary are black pixels. Fig. 3.12 illustrate the terminal points and the corresponding RoI as perceived by the AGV. When the AGV encounters the terminal point, it classifies it as a source

or destination based on the current delivery information and the direction of the move. Then according to the predefined sequence of control commands it if is a source point the AGV will request the delivery information from the edge server and proceed to pick the object, otherwise if it is a destination point, the AGV will proceed to drop the object in the correct position according to the delivery information and then it will return again to the source point for the next job.

When there is no path segment present in the RoI, three scenarios can be assumed.

1. AGV has come to end point of the line which is identified by the terminal points.

2. AGV has drifted off from the line.

3. The line is damaged and the AGV cannot identify the path.

In the first scenario, the terminal points are identified by the AGV and acted accordingly as mentioned in the above section. In the other two scenarios, the AGV is stopped immediately to avoid unwanted motion and to ensure the safety of the surround environment.

The size and the shape of the objects that are transported by the AGV are known beforehand. The end effect of the robotic arm of the AGV is knows as the gripper and it is programmed to grab/loose such objects with a pre-defined size and shape. Also, the place of the object in the source is static and the AGV has the knowledge of that static coordinate. The movement of the 4-DoF robotic arm is programmed to move to a given coordinate along its moving plane using inverse kinematics. The coordinates given in the delivery information are translated to the coordinate system in the AGV and then it is fed to the robotic arm to move to the correct coordinate in the source or destination and pick or drop the object accordingly.

3.3 Intelligence For Object Placement

This section provides a detailed explanation of the software solution that was deployed on the edge server to provide intelligence for object placement. The role of the edge server is to aid the AGV by providing intelligence in terms of computing the delivery information by observing the storage areas, which includes the destination and the drop location. The destination is one of the four square areas defined by cross-hair markers which are known as storage areas. A human operator defines a custom area possibly an irregular flat surface made out of paper for this demonstration. In this work, the custom drop area is made out of dark paper which has a good contrast when compared with the background which is light brown color(see Fig. 2.1). This contrast difference helps to separate the drop areas from the platform. The edge server provides intelligence to find out the coordinates of the largest circle that can be placed inside the user-defined area and shares the coordinates with the AGV, and then it proceeds to place the object on the coordinates with precision.

The edge server goes through several processes to derive the delivery information, which includes communication with the raspberry pi camera observing the storage areas, detection of cross-hair markers, image pre-processing steps to isolate the destination and camera distortion correction, identifying the custom area, and then compute the drop coordinates and translating the coordinates to the actual coordinates and sending the delivery information to the AGV. The flow of the edge server operation is illustrated in Fig. 3.14.

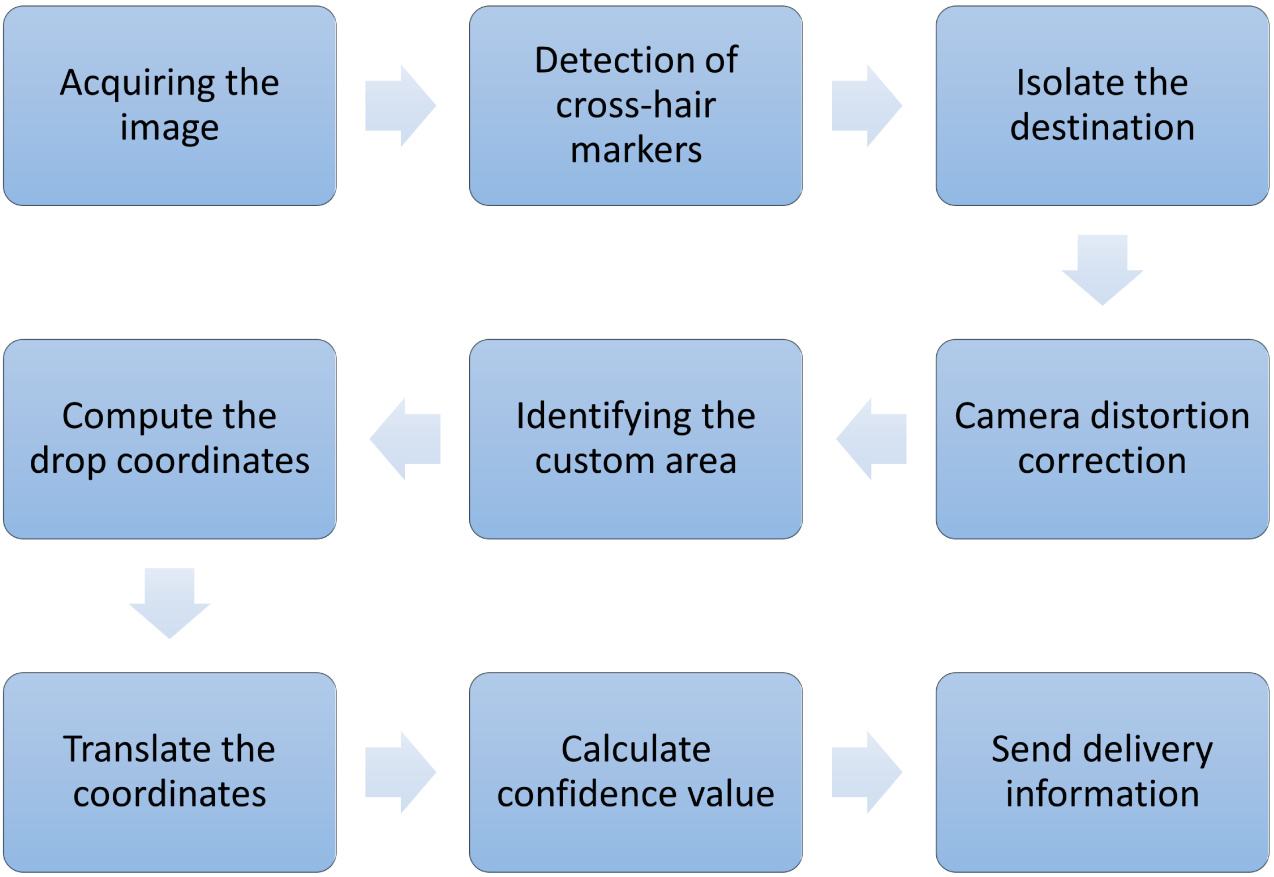


Figure 3.14. Key steps of edge AI service.

When the AGV requests the delivery information from the edge server, as the first step, the edge server

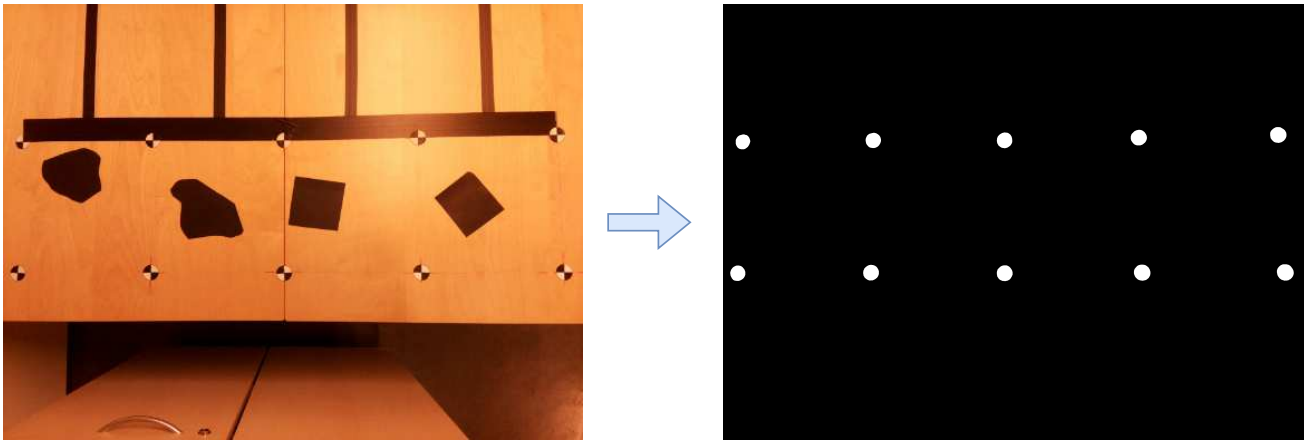


Figure 3.15. Creating the master sample.

sends a request to the Raspberry pi to acquire the bird's eye view of all the four storage areas. A high resolution image of the storage area is required by the edge server to successfully detect the cross-hair markers. Hence, the camera is configured to provide a high quality image which has a resolution of 3280×2464 px. Furthermore, the focal length of the camera is manually adjust to capture sharp and crisp details of the cross-hair markers and the objects that are on the robot platform. A sample image of the bird's eye view of the storage areas are shown in the step 1 in Fig. 3.15. Upon receiving the image from the camera, edge server proceeds to detection of cross-hair markers by using a pre-trained ML model. The ML model was trained under supervised training setting where 256×256 px images extracted from the camera is used as inputs while the labels are the images with the same dimensions having white areas corresponding to the cross-hair markers and black areas reflecting everything else. The labels were created manually by using the MATLAB image labelling tool by drawing a mask on the cross-hair markers.

In order to avoid handcrafting the whole dataset, data augmentation procedure is adopted. First, using the camera image as the master sample, a master label is generated by filtering out the background while keeping the cross-hair markers. The creation of master sample using the raw camera image is shown in Fig. 3.15. In the figure, it can be seen that the areas corresponding to cross-hair markers in the original image are labeled as white circle while the rest is set to black. Then the original image is converted into gray-scale to prepare it for the data augmentation procedure. In order to generate more data samples, different sizes of rectangles with different orientation are extracted randomly from the gray-scale converted master sample and correspondingly from the master label to generate the datasets. Fig. 3.16 illustrates an extracted sample of the dataset created by augmented images using the master label and master sample.

The extracted samples and their corresponding labels are then resized to 256×256 px images to generate the training and testing datasets for the ML model with the sizes of 1024 and 128, respectively. Next, a ML model based on the U-NET architecture [11] with the input and output dimensions of 256×256 is trained and tested with the aforementioned dataset. The trained ML model is uploaded to the edge server and it is used for the inference of the cross-hair marker detection.

The received image is up-scaled into a pre-defined resolution of 4056×3040 px which makes it easier to partition the image as detailed in the next step. Before feeding the camera image to the ML model for inference, it is converted into gray-scale. During the inference, the camera image is partitioned into 10 segments where the size and the shape of the segments are pre-defined by a set of pixel coordinates such that each segment includes one cross-hair marker. This is possible as the prior knowledge of the distance of the cross-hair markers are known by the edge server and the cross-hair markers are laid with a distance of 280 mm. The image segments are then resized into to 256×256 px images and fed to the

pre-trained cross-hair marker detection model.

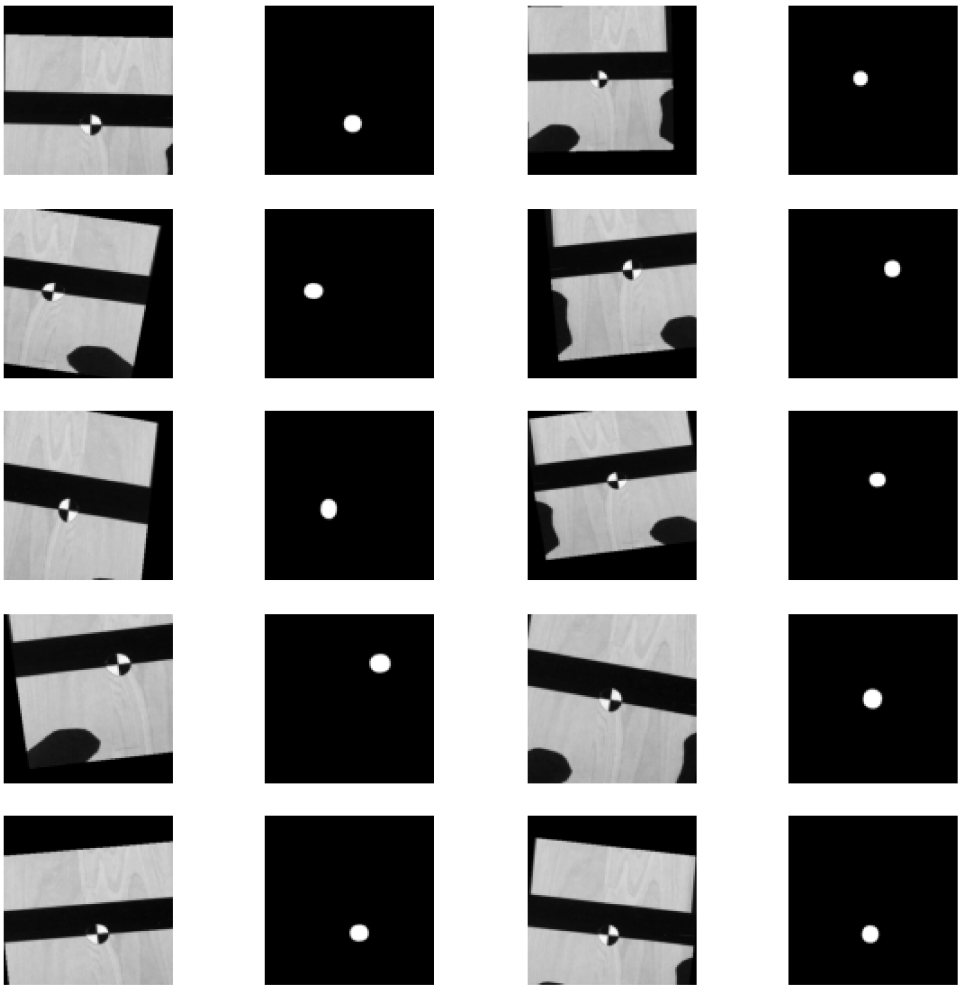


Figure 3.16. Extracted samples of augmented images.

The ML model is capable of providing the label for the given cross-hair marker image segment. But the inferred label is not exactly a circle. There could be slight alterations of the predicted label due to the distortion happen when resizing the images, surrounding lighting conditions and other details in the robot platform. A sample image is shown in Fig. 3.17 highlighting the process till the prediction of cross-hair marker labels. This figure contains 10 image segments that were extracted from the original gray-scaled image which contain one cross-hair marker for each image segment. Then the corresponding resized image segments which were fed to the ML model and the predicted cross-hair marker labels from the ML model are shown in line with the image segments in the same figure. It should be highlighted that, apart from the first and the last image segments, the other image segments are 600×600 px images whereas the first and the last image segments are 457×600 px and 442×600 px respectively. The input and output images for the model are 256×256 px. It can be seen from Fig. 3.17 that some of the predicted labels are somewhat distorted. Since the predicted labels are not complete shapes, we need to resort to a geometric approximation to find the center points of the predicted labels to identify their location in px coordinate system. By doing so, we can then isolate the four storage

areas for further processing.

In order to find the center point of the predicted labels, we can opt for a trivial method like finding the centroid of the label as it was used in the intelligence for navigation part under Sec. 3.2. But there is an inherent issue in finding the center point through the centroid method which is, if the intended shape is a concave shape there could be a chance that the center point can be situated outside of the shape [30]. In order to avoid this, we calculate the visual center point or the center point of the maximum inscribed circle. In other words, this point is known as the furthest point away from an edge which resides inside a polygon. The calculation of the center point is done via a python package known as the `polylabel()` [30] which provides a fast and efficient way with sufficient precision to calculate the center point of the maximum inscribed circle of a polygon. This same tool is utilized in the latter parts of calculating the delivery information process.

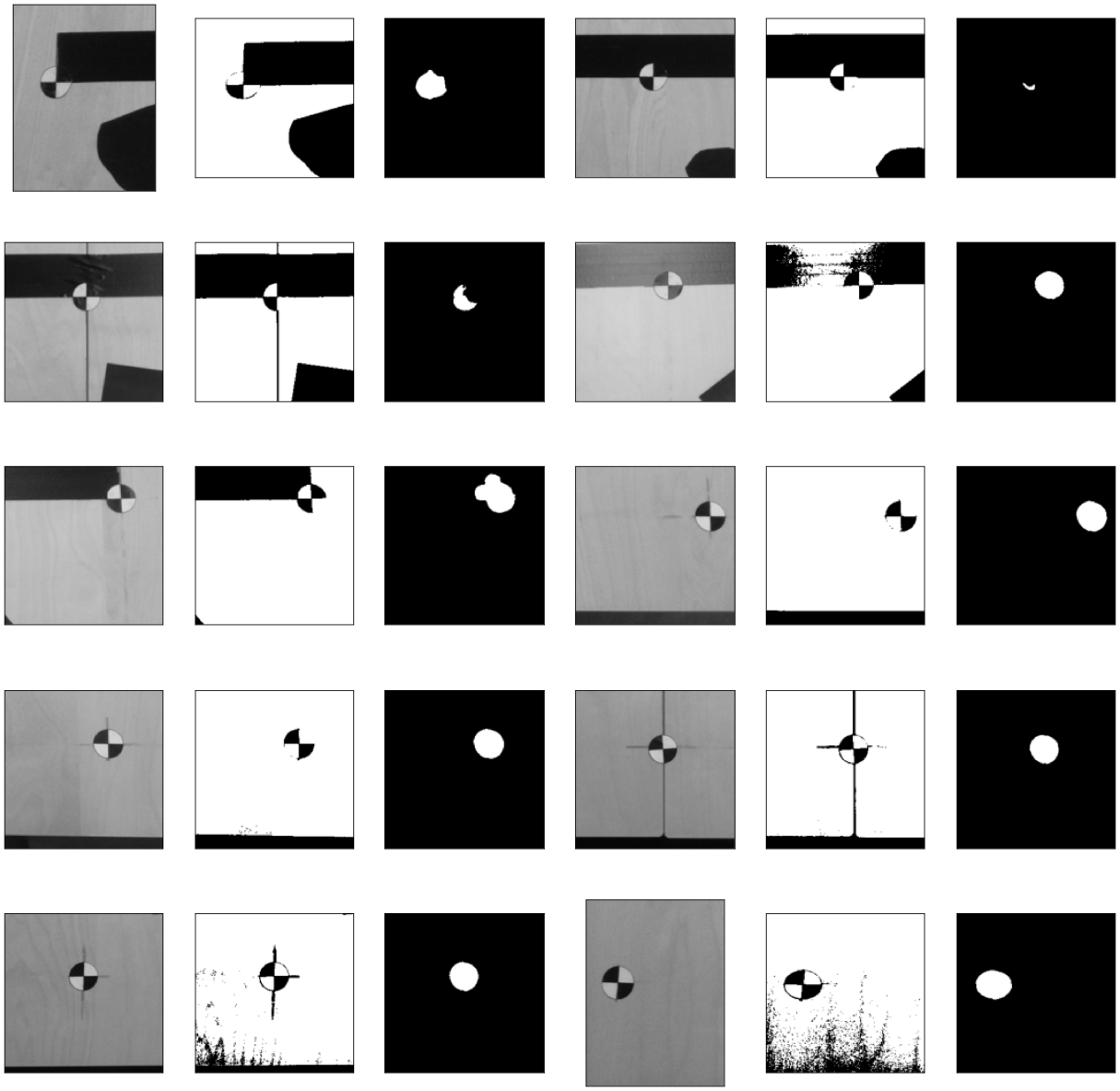


Figure 3.17. Ten examples of image segments, resized image segments, and their corresponding predicted labels (from left to right in each set of three images).

There are a few steps associated with calculating the center point from the predicted cross-hair marker label. First, the label is scaled back to its original size which is the size corresponding to the relevant image segment. The resized predicted label is shown in the step 1 of Fig. 3.18. Then the contours of the objects in the images are detected and localized. The `polylabel` package requires a polygon as an input to find the center point coordinates of the interested area covered by the contour. Therefore, the polygon approximation is carried out for the contours found in the image so that the contour can be defined by the polygon. The detected contour is drawn on the figure in step 2 and the approximated polygon is drawn in step 3 in Fig. 3.18.

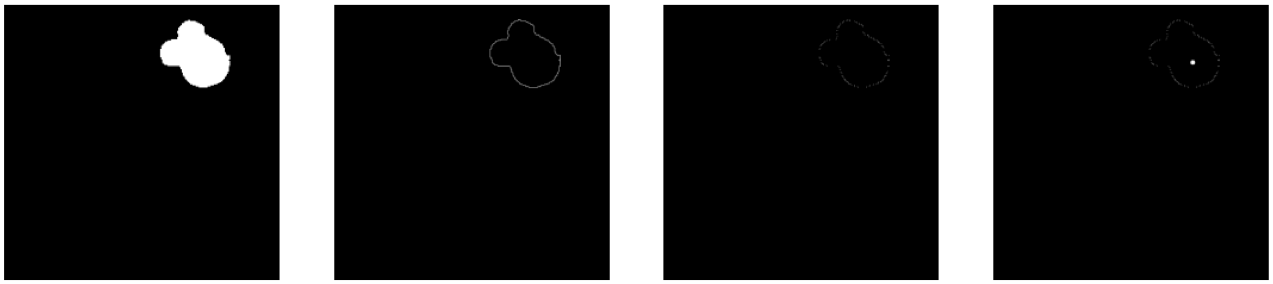


Figure 3.18. Predicted label, contour, approximated polygon, and the center of the polygon.

Then the coordinate points of the vertices that construct the polygon are fed to the function. This polygon approximation can be done for a given precision which then closely resembles with the actual contour. In this work, a precision of 0.001×contour perimeter was sufficient to construct an accurate polygon which serves the purpose. As there could be multiple contours detected in the predicted label, the area inside each of the contours was calculated and the contour with the highest area was selected as the contour that is related with the cross-hair marker. Then the selected contour's polygon was approximated with the said precision level and the coordinate points that define the polygon vertices were fed to the `polylabel` function to get the center coordinates of the predicted cross-hair marker. The detected center point is marked in the step 4 of Fig. 3.18. This process is repeated for each of the 10 cross-hair markers to get the local center coordinates of the predicted cross-hair marker by the ML model. Then these local center coordinates are translated to the global center coordinates with the help of the pixel coordinate points that define the 10 image segments. The local center coordinates refer to the coordinate points that are calculated with respect to the current image segment whereas the global center coordinates refer to the coordinates which are calculated with respect to the original image before the segmentation. Fig. 3.19 shows the global center coordinates plotted on the same figure and it can be seen that the predicted center point closely coincides with the actual center point of the cross-hair marker. There could be instances, that there would be a slight offset from the calculated center point coordinates of the predicted cross-hair marker label and the center point coordinates of the actual cross-hair marker. This would lead to a slight offset of the dimensions of the storage areas when isolating it from the image. But the effects of this slight offset are minimal for the final outcome as the offset would be in the *mm* scale whereas the object that is being delivered has dimensions of several magnitudes in *cm* scale. Hence, the offset of a *mm* level for the final object placement would be negligible.

During this process if there were any missing cross-hair markers or if the ML model fails to correctly predict the cross-hair marker labels due to any unforeseen circumstance, the edge server will terminate the process and return an error to the end user without further processing. After this process is carried out for all 10 cross-hair markers, the edge server has the knowledge of the location of cross-hair markers

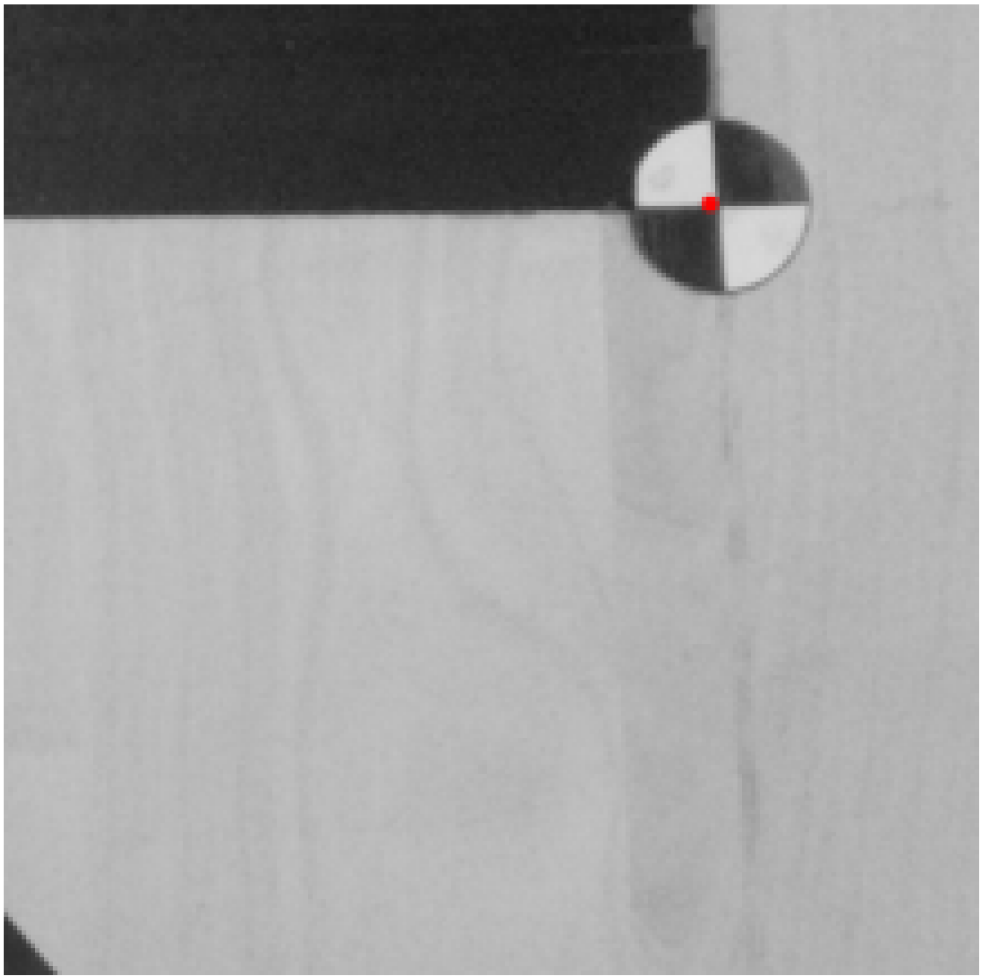


Figure 3.19. Center coordinates of the detected marker.

with respect to the image coordinate system. Then it can be used to successfully segment the image to isolate the storage areas as defined by the cross-hair markers for further processing.

After the detection of all cross-hair markers, the edge server proceeds to isolate the selected storage area by the user. Storage areas are named by the value *storageId* and they are numbered from 1 to 4 where the number 1 storage area denotes the rightmost storage area in the camera image. When requesting the delivery information, the user provides the desired storage area by specifying it in the field *storageId*. With this value, the edge server isolates the correct storage area.

After isolating the correct storage area, the edge server then advances to the next step which is to correct the camera distortion. As the environment that is being simulated in this demonstration is a warehouse, there could be a high chance for frequent changes in the camera angle and its position due to the dynamic nature of the surrounding environment. Hence, the image obtained through the camera is prone to distortions caused by the camera angle and position. So the isolated storage areas may not have the exact bird's eye view. This is directly affecting the center point calculation of the irregular shape placed inside the storage area. Also, due to the way that the storage areas are positioned, the view angle from the camera for each storage area differs slightly. i.e The rightmost and leftmost storage areas have a higher viewing angle whereas the storage areas in the center have a slightly less viewing angle. Hence, the viewing angles for each storage area should be aligned so that all the storage areas have the same viewing angle which will lead to an accurate result. In order to align the camera viewing

angles for each storage area and to avoid camera distortions due to camera position, the perspective transformation which is implemented in the OpenCV package [31] is used.

Perspective transformation is useful in processing images taken by a camera as it allows the user to manipulate the viewing angle of the image. This can be used to correct the distortion of an image due to the camera's angle or to create images with a different perspective. It is done by warping the image so that it appears as if it is viewed from the given perspective. The mathematical form of perspective transformation is expressed as follows [24]

$$\begin{bmatrix} t_i x' \\ t_i y' \\ t_i \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.7}$$

where source image coordinates (x, y) are multiplied by the perspective matrix to apply the perspective transformation. Pixel coordinates of the transformed image (x', y') can be obtained by scaling down the result by t_i factor.

In order to resolve the 8 unknowns in the perspective matrix, we provide 4 pairs of coordinate values from the input image which defines the source quadrangle and its corresponding pixel coordinate values that are in the transformed image which defines the target quadrangle as desired per the use-case. This target quadrangle depends on the required perspective to which it should be transformed. In our case, it is required to change the perspective of the storage areas as seen by the camera to a bird's eye view. The transformation that is needed for the storage areas is graphically illustrated in Fig. 3.20 in which the quadrangle defined by points $ABDC$ is transformed into a rectangle defined by points $A'B'C'D'$ by providing a correction to the distortion caused by the viewing angle. In a similar fashion, the storage areas defined by the cross-hair markers are transformed into an area with a perspective of a bird's eye view as described below.

To facilitate the above transformation, the input coordinate values are chosen such that it represents the center points of the cross-hair markers that define the selected storage area and the coordinates of the target quadrangle are calculated by adding the height and the width of the storage area in pixel values to the top left coordinates of the storage area. The height and the width are added to the top left corner coordinates such that it constructs a rectangle. The height is considered as the length of the left border of the storage area and the width is considered as the top border of the storage area. L1 norm is used to calculate the height h and the width w of the storage area in the original image by using the coordinates values of the corner points, and it is rounded off to the nearest integer (denoted by $\lfloor \cdot \rceil$) as follows:

$$w = \left\lfloor \sqrt{(x_{\text{tl}} - x_{\text{tr}})^2 + (y_{\text{tl}} - y_{\text{tr}})^2} \right\rceil, \tag{3.8}$$

$$h = \left\lfloor \sqrt{(x_{\text{tl}} - x_{\text{bl}})^2 + (y_{\text{tl}} - y_{\text{bl}})^2} \right\rceil. \tag{3.9}$$

After deriving the height and width of the storage area, the coordinate pairs of the corners of the target quadrangle are calculated by

$$(x'_{\text{tl}},\ y'_{\text{tl}}) = (x_{\text{tl}},\ y_{\text{tl}}), \tag{3.10}$$

$$(x'_{\text{tr}},\ y'_{\text{tr}}) = (x_{\text{tl}} + w - 1,\ y_{\text{tl}}), \tag{3.11}$$

$$(x'_{\text{tr}},\ y'_{\text{tr}}) = (x_{\text{tl}} + w - 1,\ y_{\text{tl}} + h - 1), \tag{3.12}$$

$$(x'_{\text{tr}},\ y'_{\text{tr}}) = (x_{\text{tl}},\ y_{\text{tl}} + h - 1), \tag{3.13}$$

where x_{tr}, x_{br} and x_{bl} represent the x coordinate of the top left corner, top right corner, bottom right corner and bottom left corner respectively and y coordinate is defined with the same notation. Then by using the eight coordinate pairs, the perspective matrix given in (3.7) is resolved.

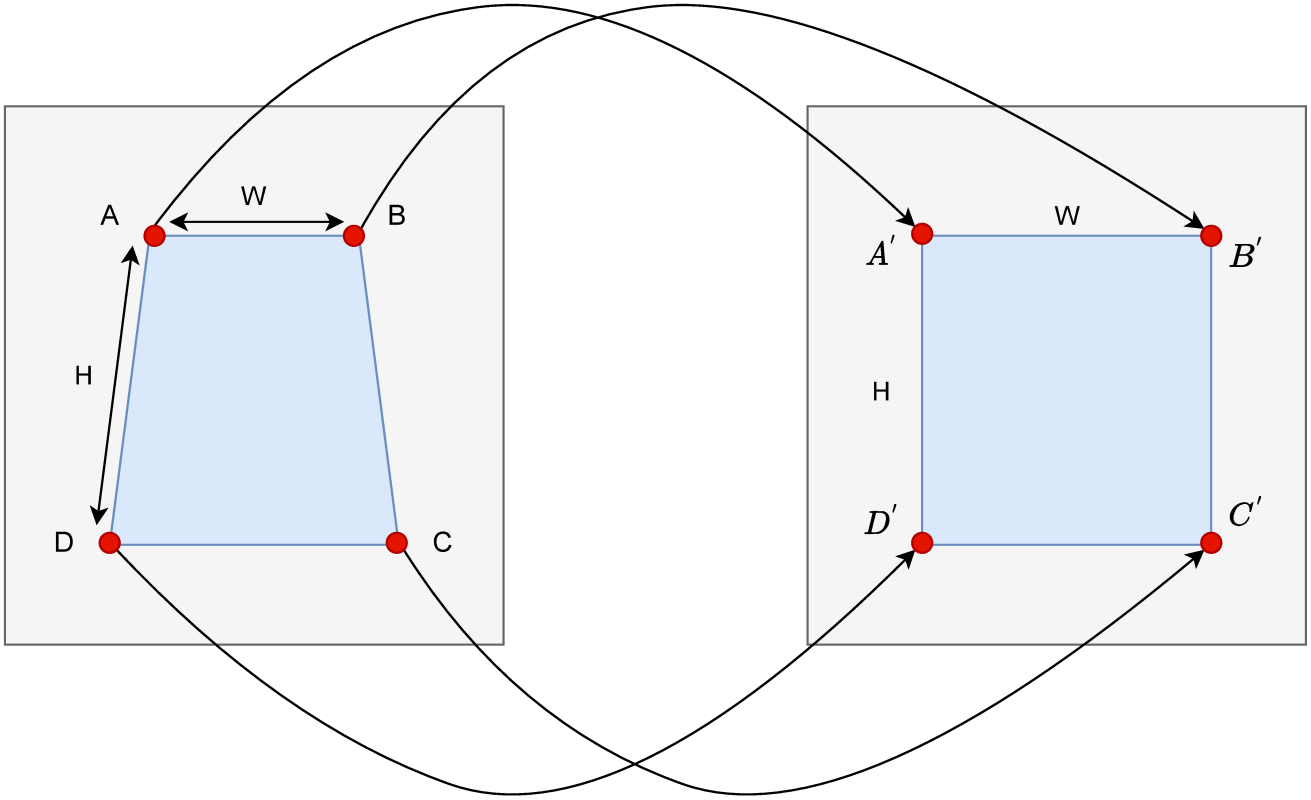


Figure 3.20. Perspective transformation to get the bird's eye view.

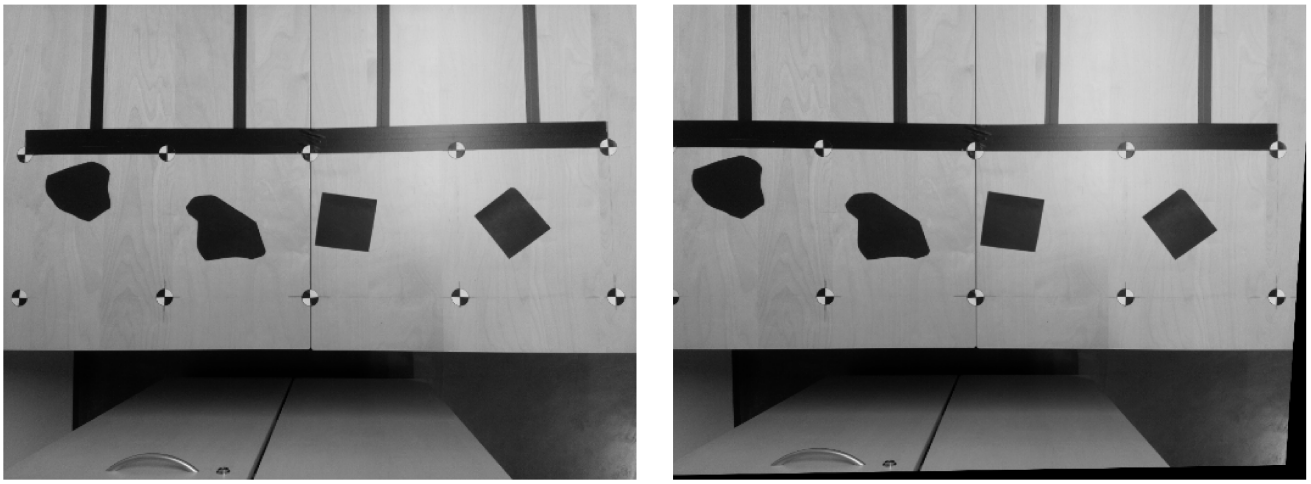


Figure 3.21. Comparison of the original image and perspective transformed image.

After obtaining the perspective matrix, the perspective transformation can be applied to the whole image to correct the camera distortion. Fig. 3.21 shows a side-by-side comparison of the original image and the transformed image in which the perspective transformation is applied to correct the distortion in the rightmost storage area. The image on the right side refers to the original image and the other image refers to the transformed image. The artifacts of the effect of warping that was used to transform the image can be seen on the edges of the transformed image. This distortion correction is applied to all of the 4 storage areas and then the distortion-corrected storage areas are extracted.

Now that the edge server has isolated distortion-corrected storage areas, it can proceed to the next

step to identify the custom area inside the storage area by using an edge detection technique. As edge detection is vulnerable to noise, before the edge detection, the result image goes through a few image pre-processing steps to reduce the noise present in the image and smoothen it out and enhance the edges. Applying a Gaussian filter to remove the noise does not yield good results, as it tends to smoothen out the edges in the image and we may lose the details of the edges of the custom area. Thus, a bilateral filter [32] is used to remove the noise which is known as an edge-aware filtering method. Bilateral filtering is a non-linear image processing technique used to reduce noise while preserving the edge details of the image. The technique is based on a weighted average of pixels within a neighbourhood, with the weights determined by the similarity of the mean intensity of the pixels. This ensures that the edges of objects in the image remain sharp, while still smoothing out random noise. It is particularly useful for preserving edges while smoothing out noise as it allows for a greater degree of control in determining how much noise is removed. The mathematical expression for the bilateral filter is as follows:

$$G(p) = \frac{1}{\lambda} \sum_{q \in S} \mathcal{N}(|p - q|)\mathcal{N}(|I_p - I_q|)I_q \tag{3.14}$$

where $\mathcal{N}$ denotes a Gaussian distribution and I_q denotes the intensity value of the pixel q. S denotes a pixel neighbourhood around p and the normalization factor w is given by,

$$\lambda = \sum_{q \in S} \mathcal{N}(|p - q|)\mathcal{N}(|I_p - I_q|)I_q. \tag{3.15}$$

The performance of the bilateral filter could be adjusted by varying the variances of the two Gaussian distributions to achieve the maximum noise reduction. In this work, it was found out that 25 for each σ value was for sufficient noise reduction while preserving the edges. The bilateral filter with said values was applied for each pixel with a pixel neighborhood of a diameter of 8. Then, the resulted de-noised image was used to detect edges in the custom area. For this purpose, we use a popular edge detection technique known as the Canny Edge Detector [33]. Canny edge detection is widely used in image processing to distinguish edges in a given picture due to its precision and resiliency and it is known as an optimal method to identify edges. It is a multi-stage algorithm that employs four primary stages for edge detection. Initially, noise reduction is achieved via Gaussian filtering, which smooths the image and eliminates any spurious or false edges. Subsequently, the calculation of the magnitude and direction of the intensity gradients of the image is conducted by convoluting the image with two distinct kernels and then determining the magnitude and orientation of the gradients of the image. Then, non-maximum suppression is used to thin the identified edges and eliminate any undesired edges. Finally, hysteresis thresholding is done to identify true edges while discarding false edges by employing two threshold values for the intensity gradient. The final output of the detected edges can be calibrated by adjusting the threshold values. These threshold values for this work were chosen by using the median value of the intensities in the given image and adding a correction to it based on the brightness of the image as the brightness of the image affects the edge detection. Hence, the upper and lower threshold values provided for the canny edge detector depend on the median intensity value of the image and they are calculated on the fly. The detected edges of the storage areas are illustrated in Fig. 3.22.

Then the resulted image from the canny edge detector is further processed to enhance the edges by using the morphological operator dilation which is elaborated in Sec. 3.2. A kernel of size 10×10 with the value of 1 was used as the structuring element in this process and it was done in a single iteration. Then the dilated image was cropped by the boundaries that consist of the transformed cross-hair marker center points. In order to remove any discontinuities that might remain on the detected edge of the custom area, another morphological operator known as closing is performed. Morphological closing is

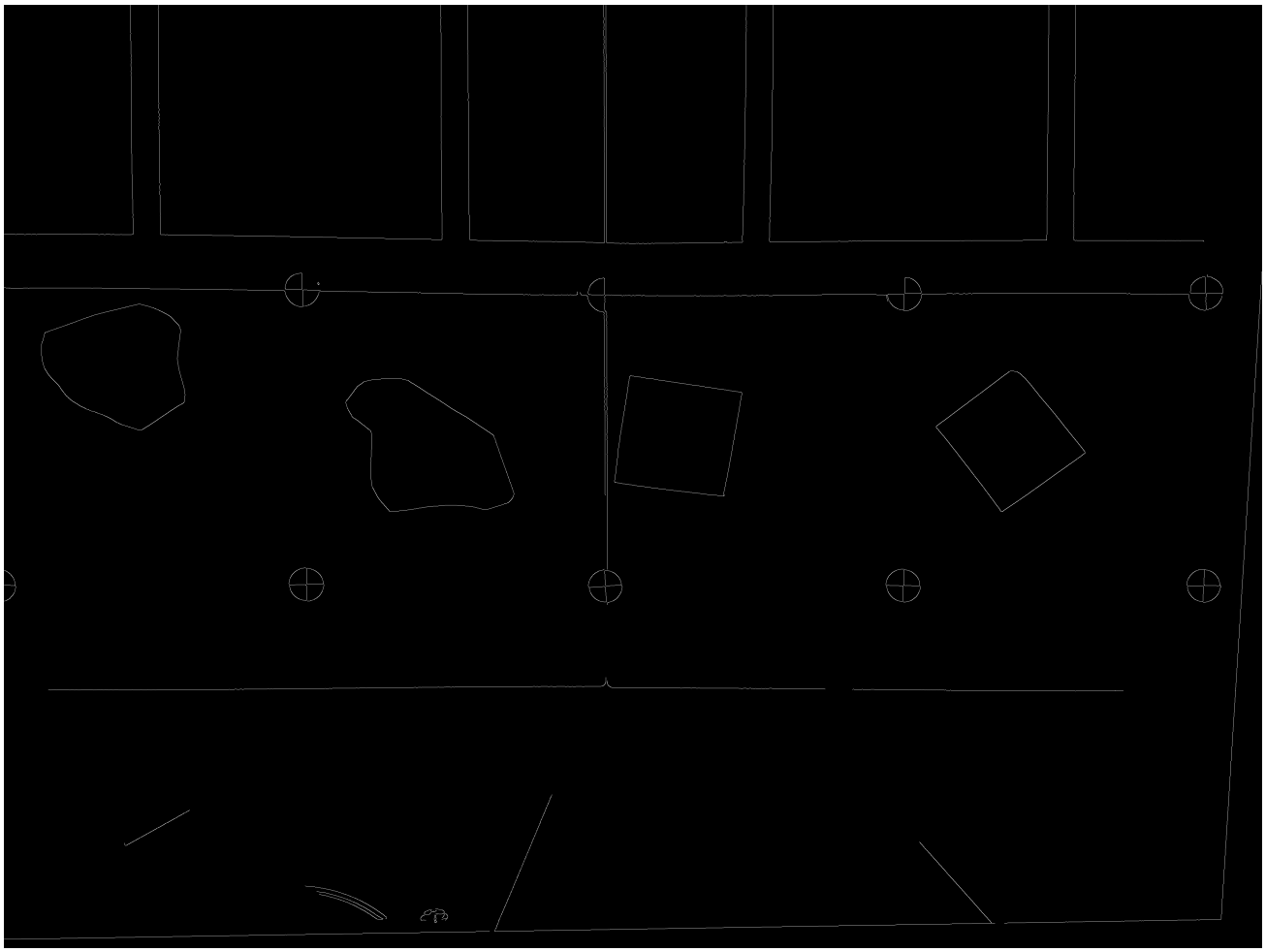
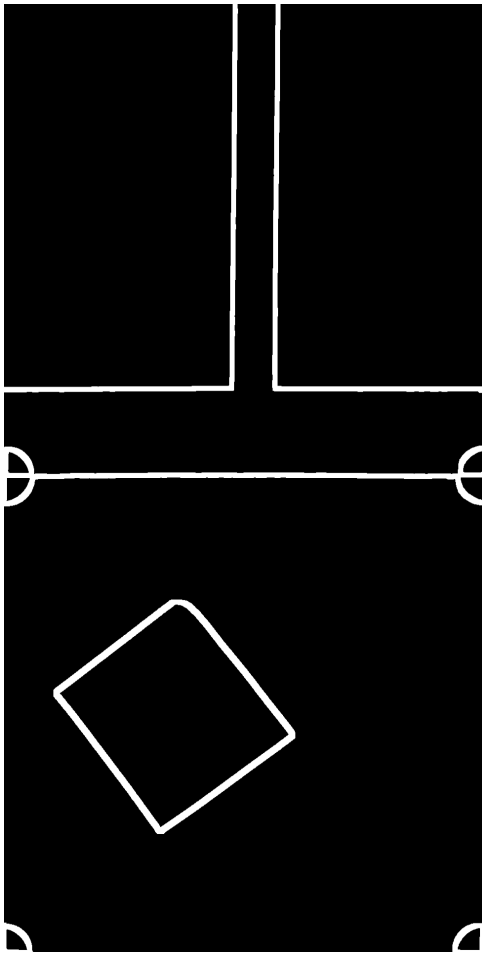


Figure 3.22. Image after the edge detection step.

done by performing the erosion on the dilated image by the same structuring element and it is used to remove any small holes in an image. Small-scale discontinuities along the edges are removed after this process.

Afterwards, the calculation of the delivery information which is to locate the center point of the maximum inscribed circle inside the custom area is carried out. This is done by following the same steps in the process of calculating the center points of the cross-hair markers. A polygon approximation carried out for the contours that were found in the storage area. In this process, when choosing the contours, the contour with the second largest area was chosen for the polygon approximation and subsequently used in the `polylabel()` function to get the center point coordinates of the maximum inscribed circle. The second largest area was chosen to get the innermost polygon of the custom area. The image which was processed by the morphological operators and the final output image where the center point coordinates were derived and localized are shown in Fig. 3.23.

When providing the delivery information for the AGV, the center coordinates should be in *mm* scale with respect to the coordinate system of the storage area where the origin resides at the furthest marker at the bottom left. Hence, the derived local center coordinate points are converted to global center coordinates and they are translated to the actual physical coordinate system where the origin is at the furthest cross-hair marker at the bottom left. With the prior knowledge of the setup dimension and the placement of the cross-hair markers where the distance between each cross-hair marker is 280*mm*, the pixel coordinate values are converted into *mm* scale. Then the confidence value is calculated for the



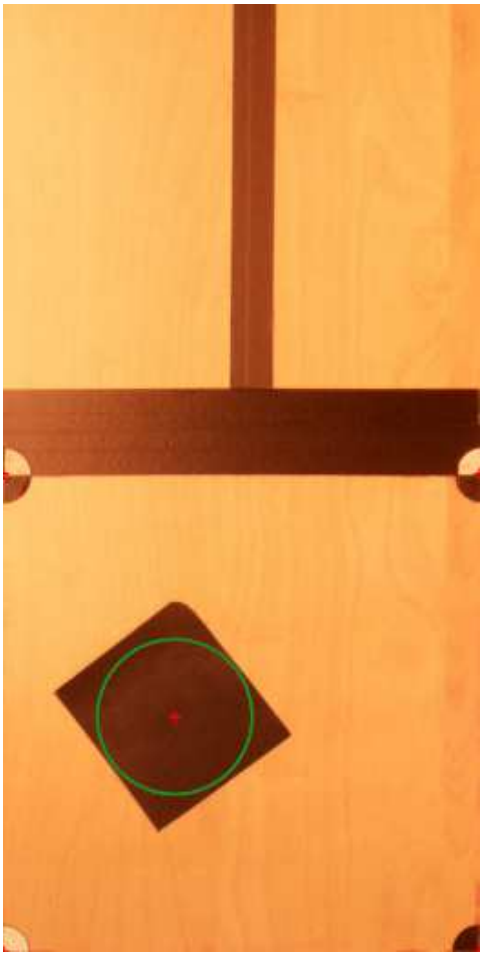


Figure 3.23. Cropped image with enhanced edges and the final image with the center point.

received center coordinates by considering the center point coordinates and the radius of the maximum inscribed circle such that it provides a higher confidence value if the circle is within the storage area up to a given offset value. Then the coordinate pair, confidence value, and radius are parsed into a JSON as shown in Listing 1 in Sec.2.2 and then shared with the AGV.

4 Conclusion and Future Work

4.1 Conclusion

This demo work is a proof of concept to illustrate how the codesign of edge intelligence and AGV can be utilized to automate repetitive tasks with a high degree of accuracy and efficiency while having the human-in-the-loop and thereby improve productivity. The software related to this demo is available at GitHub `https://github.com/ICONgroupCWC/Demo.Percom23`.

With a similar platform developed as per specifications provided in Sec. 2.2 and the use of the hardware specified under Sec. 3.1 along with the aforementioned software, this demo can be reproduced. Please be mindful that the PID coefficients of the AGV have to be tuned according to the new environment. This demo in action can be seen at `https://youtu.be/DhCSCCZbuHo`. Further, a proposal for this demonstration in the form of a technical paper was submitted to "The 21st International Conference on Pervasive Computing and Communications" (PerCom 2023) which is a premier annual scholarly venue in pervasive computing and communications.

4.2 Future Work

The lighting conditions (over/under-exposure and harsh shadows) directly impact the performance of the AGV. To improve the overall performance, brightness and exposure corrections methods and deep learning models for denoising and filtering are to be investigated in the future. Furthermore, it is expected to investigate the obstacle avoidance of the AGV using deep learning models while navigating on a predefined path.

5 BIBLIOGRAPHY

[1] Abri A.G. & Mahmoudzadeh M. (Mar 2015) Impact of information technology on productivity and efficiency in iranian manufacturing industries. Journal of Industrial Engineering International 11, pp. 143–157.

[2] Tang H., Cheng X., Jiang W. & Chen S. (2021) Research on equipment configuration optimization of AGV unmanned warehouse. IEEE Access 9, pp. 47946–47959.

[3] Li S., Yan J. & Li L. (2018) Automated guided vehicle: the direction of intelligent logistics. In: *Proc. of IEEE Intl. conf. on SOLI*, pp. 250–255.

[4] Shaw J.S., Liew C.J., Xu S.X. & Zhang Z.M. (2019) Development of an AI-enabled AGV with robot manipulator. In: *Proc. of IEEE ECICE*, pp. 284–287.

[5] Qiu T., Chi J., Zhou X., Ning Z., Atiquzzaman M. & Wu D.O. (2020) Edge computing in industrial internet of things: Architecture, advances and challenges. IEEE Communications Surveys & Tutorials 22, pp. 2462–2488.

[6] Bošnak M. & Škrjanc I. (2021) Obstacle avoidance for line-following AGV with local maps. In: *Proc. of IEEE Intl. SACI*, pp. 193–198.

[7] Wang X., Han Y., Leung V.C., Niyato D., Yan X. & Chen X. (2020) Convergence of edge computing and deep learning: A comprehensive survey. IEEE Communications Surveys & Tutorials 22, pp. 869–904.

[8] Murshed M.G.S., Murphy C., Hou D., Khan N., Ananthanarayanan G. & Hussain F. (oct 2021) Machine learning at the network edge: A survey. ACM Comput. Surv. 54.

[9] Ke R., Zhuang Y., Pu Z. & Wang Y. (2021) A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on iot devices. IEEE Transactions on Intelligent Transportation Systems 22, pp. 4962–4974.

[10] Mazzia V., Khaliq A., Salvetti F. & Chiaberge M. (2020) Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application. IEEE Access 8, pp. 9102–9114.

[11] Ronneberger O., Fischer P. & Brox T. (2015) U-net: Convolutional networks for biomedical image segmentation. In: *Proc. of Intl. Conf. on Medical image computing and computer-assisted intervention*, Springer, pp. 234–241.

[12] Bennett S. (1993) Development of the PID controller. IEEE Control Systems Magazine 13, pp. 58–62.

[13] Kodagoda K., Wijesoma W. & Teoh E. (2002) Fuzzy speed and steering control of an agv. IEEE Transactions on Control Systems Technology 10, pp. 112–120.

[14] Amin S.U. & Hossain M.S. (2021) Edge intelligence and internet of things in healthcare: A survey. IEEE Access 9, pp. 45–59.

[15] Xu P., Wang K., Hassan M.M., Chen C.M., Lin W., Hassan M.R. & Fortino G. (2022) Adversarial robustness in graph-based neural architecture search for edge ai transportation systems. IEEE Transactions on Intelligent Transportation Systems pp. 1–10.

[16] Shi Y., Yang K., Jiang T., Zhang J. & Letaief K.B. (2020) Communication-efficient edge ai: Algorithms and systems. IEEE Communications Surveys & Tutorials 22, pp. 2167–2191.

[17] Nain G., Pattanaik K. & Sharma G. (2022) Towards edge computing in intelligent manufacturing: Past, present and future. Journal of Manufacturing Systems 62, pp. 588–611.

[18] Ansari M.S., Alsamhi S.H., Qiao Y., Ye Y. & Lee B. (2020) Security of distributed intelligence in edge computing: Threats and countermeasures. In: *The cloud-to-thing continuum*, Palgrave Macmillan, Cham, pp. 95–122.

[19] Milletari F., Navab N. & Ahmadi S.A. (2016) V-net: Fully convolutional neural networks for volumetric medical image segmentation. In: *2016 fourth international conference on 3D vision (3DV)*, IEEE, pp. 565–571.

[20] Waveshare, Jetank ai kit. Available at `https://www.waveshare.com/jetank-ai-kit.htm` (2022/11/11).

[21] Raspberry pi camera moduke v2. Available at `https://www.raspberrypi.com/products/camera-module-v2/` (2022/11/11).

[22] Raspberry pi 4 model b. Available at `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/` (2022/11/11).

[23] Burger W. & Burge M. (2010) Principles of Digital Image Processing: Core Algorithms. Undergraduate Topics in Computer Science, Springer London.

[24] Gonzalez R.C. & Woods R.E. (2008) Digital image processing. Prentice Hall, Upper Saddle River, N.J.

[25] Otsu N. (1979) A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics 9, pp. 62–66.

[26] Bovik A.C. (2009) Chapter 4 - basic binary image processing. In: Bovik A., editor, *The Essential Guide to Image Processing*, Academic Press, Boston, pp. 69–96.

[27] (2022) Dedication. In: Siegel A.F. & Wagner M.R., editors, *Practical Business Statistics (Eighth Edition)*, Academic Press, p. v, eighth edition edition.

[28] Zaidner G., Korotkin S., Shteimberg E., Ellenbogen A., Arad M. & Cohen Y. (2010) Non linear pid and its application in process control. In: *2010 IEEE 26-th Convention of Electrical and Electronics Engineers in Israel*, pp. 000574–000577.

[29] Ogata K. (2010) Modern Control Engineering. Instrumentation and controls series, Prentice Hall.

[30] Agafonkin V., A new algorithm for finding a visual center of a polygon. Available at `https://blog.mapbox.com/a-new-algorithm-for-finding-a-visual-center-of-a-polygon-7c77e6492fbc` (2022/11/11).

[31] Bradski G. (2000) The OpenCV Library. Dr. Dobb's Journal of Software Tools .

[32] Tomasi C. & Manduchi R. (1998) Bilateral filtering for gray and color images. In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pp. 839–846.

[33] Canny J. (1986) A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8, pp. 679–698.