Autonomous Robot for In-Pipe Defect Detection

by

Ghadi Alshehri 2112956

Raghad Alghamdi 2111376

Raheed Karah 2110296

Rawnaa Aljohani 2110160

Shahad Alharbi 2111904

Computer and Network Engineering Department

College of Computer Science and Engineering

Jeddah University, SAUDI ARABIA

Supervisor

Dr. Sabra ElFerchichi

(2024)

روبوت ذاتي لاكتشاف المشاكل داخل الأنابيب

يهدف هذا المشروع الى تطوير وسائل استكشاف المشاكل داخل انابيب الغاز والنفط عن طريق روبوت ذا التحكم الذاتي للكشف عن المشاكل داخل الأنابيب (IPIR) ، والتي تعتمد غالب الاحيان على اجهزه استكشافيه تسمى (PIGs) التي تصدرها شركات مثل SGS في السعودية. يهدف هذا المشروع الى تطوير روبوتات ذكيه للكشف عن المشاكل للحد من الطرق التقليديه المكلفه للمال والوقت والمصادر التي تفتقد لعنصر الذكاء الاصطناعي والقدره الدقيقه على تحليل البيانات. احد اهم نقاط قوه (IPIR) هي القدره على توحيد ادوات الذكاء الاصطناعي مع نظام الملاحه الذاتي. يتكون جسد الروبوت من العقل المدبر رازبيري باي ٤، هو المسؤول عن تنفيذ العمليات المتعلقه بنظام الكشف والتصنيف عن المشاكل الموجوده داخل الانبوب باستعمال نموذج (YOLOv9t) للكشف عن المشاكل عن طريق الكاميرا (OV5647). يتم التحكم في حركه الروبوت عن طريق جسر متحكم متصل ب ٤ محركات بعجلات مع ارديونو اونو، مع انكودر لحساب المسافات، بالاضافه الى ٣ مستشعرات للمسافه من اعلى الروبوت لتجنب اي عقبات او اصطدام بحائط الانبوب. يستطيع الروبوت الكشف عن المشاكل المتعلقه بداخل الانبوب منها ، شقوق، عقبات، او حفر بالوقت الحقيقي مع نسبه عاليه الدقه تساوي ٧٧.٠ . يمتلك النظام صفحات على الانترنت ، صفحه الداشبورد الاساسيه تظهر فيها صوره مشكله الانبوب ، مع اسم المشكله وموقعها فالوقت الحقيقي، لمساعده فريق الصيانة لتحديد المشكله ومكانها لوضع خطه لعلاجها. من اهم نقاط قوه الروبوت انه يستطيع الحركه داخل انبوب على شكل (T) و (L) بسهوله ودقه. وفالختام، قدم هذا البحث قد طرق لرفع معدل الانتاجيه وتقليل المشاكل الماليه المتعلقه بالانابيب تحت الارض الى حد كبير . لتطوير المشروع تم اضافة عده نقاط لتقويه المخرجات منها اختيار تقنيات تواصل بعيده المدى لضمان تغطيه اكبر مسافه ممكنه، استخدام خوارزميات متقدمه لضمان دقه ونتائج افضل ، زياده سعه المعالج لزياده السرعه . بهذه الطريقه الروبوت سوف يحقق امكانيات غير متوقعه في عالم الصناعة .

Abstract

This project presents the development of an Autonomous In-Pipe Inspection Robot (IPIR) designed to address the challenges of current pipeline inspection practices, which often rely on smart Pipeline Inspection Gauges (PIGs) offered by companies like SGS in Saudi Arabia. While these advanced technologies utilize electronic and ultrasonic sensors to detect issues such as metal loss and welding defects, their deployment presents significant challenges, including high costs, operational complexity, and inherent safety risks during the pigging process. Additionally, the lack of AI tools and cameras for defect visualization hinders efficient data analysis. The IPIR integrates modern advancements in artificial intelligence and autonomous navigation. It features a Raspberry Pi 4 as the central processing unit running the YOLOv9t-based defect detection model and an OV5647 camera for visual inspections. The robot's movement is controlled by an Arduino microcontroller with an H-bridge motor driver, while a single optical encoder provides accurate navigation feedback. Extensive testing has demonstrated that the IPIR effectively detects and classifies common pipeline defects, including cracks, holes, and obstacles, in real time, achieving a model accuracy of 77%. A web-based dashboard enhances usability by visualizing defect types, locations, and captured images, allowing the maintenance team to make informed decisions. The robot can navigate effectively through T and L junction turns inside the pipe. In conclusion, this research provides a scalable and cost-effective solution for pipeline inspections, addressing critical challenges in underground maintenance. Future work will focus on enhancing wireless communication for reliable data transmission, developing advanced control algorithms, and upgrading the robot's processing capabilities. This study contributes to improving pipeline integrity management and offers a safer alternative to traditional PIG technologies.

Acknowledgments

Our profound gratitude and deep appreciation are owed to Dr. Sabra ElFerchichi, and her invaluable contributions including suggesting the idea, supervising the work, providing expert advice, and offering sincere assistance throughout the project. The sincere gratitude are also to all the faculty members of Computer and Network Engineering Department for constantly motivating and supporting us.

We would like to extend our heartfelt gratitude to Eng. Meshari Alharbi for his invaluable assistance in providing us with detailed information regarding the inspection of pipes for defect detection. His expertise and insights have significantly enhanced our understanding of the processes involved. Additionally, we appreciate his time and effort in completing our project survey, which will greatly contribute to the success of our research. Thank you, Engineer Meshari, for your support and collaboration.

We would like to express our sincere gratitude to Eng. Nader Karah for his invaluable expertise and generous support in providing comprehensive information regarding pipe inspection techniques for defect detection. His insightful guidance has significantly deepened our understanding of the intricacies of this process. Additionally, we are immensely grateful for his dedication and diligence in completing the project survey, which will undoubtedly contribute significantly to the success of our research endeavors. We extend our heartfelt thanks to Eng. Nader Karah for his collaboration and unwavering support.

Eng.Ibrahim Aljohani generously provided us with a collection of pipes, which has been instrumental in enabling us to construct a realistic dataset essential for training our AI model. In addition to this invaluable contribution, his profound knowledge of traditional pipe examination methods has significantly enriched our understanding of the field. We extend our sincere gratitude to Engineer Ibrahim Aljohani for his generous support and expertise.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The oil and gas industry is one of the largest sectors in the world in terms of dollar value, generating an estimated 5 trillion in global revenue as of 2022 [9]. Saudi Arabia is considered a leading company that possesses approximately 17 percent of the world's proven petroleum [10]. Oil and gas pipelines stand as vital conduits, facilitating the transportation of valuable resources across vast distances.To ensure the safety of both the environment and the communities they pass through, maintaining the integrity and efficiency of pipelines is essential. Many companies, particularly in the oil and gas sector, continue to utilize traditional inspection methods for pipeline and equipment integrity. While there is an increasing adoption of advanced techniques like non-destructive testing (NDT), the transition is often gradual. This approach allows companies to effectively combine both traditional and modern methods to ensure safety and efficiency. Most of the industrial companies are still using traditional inspection methods. Non-destructive testing (NDT), is a commonly used method for testing a component without damaging it. An example of (NDT), visual inspection is an external inspection method that relies on human intervention exposing workers to dangerous conditions [11]. Another essential procedure is the internal inspection of pipelines, which assesses the state of pipes from the inside to spot any possible problems like corrosion, blockages, cracks, or leaks. For instance, pipeline inspection gauges (PIGs) are large tools most employed type in oil and gas pipelines, They move with the flow inside the pipeline [12]. It started as a cleaning and displacement tool and then has been developed into a smart PIG [12]. In this chapter, we will elaborate on the main idea of our project which focuses on solving the oil and gas pipeline underground defects. The problem description is presented in section 1.1 then aims are in section 1.2 and objective

in section 1.3. Finally, the report outline is presented in section 1.5 and lastly, the project plan is presented in section 1.6.

1.1 Description of the Problem

Companies like SGS in Saudi Arabia offer advanced smart PIGs for pipeline inspection [13]. These smart PIGs feature enhanced capabilities compared to traditional models, incorporating electronic sensors and ultrasonic sensors that allow for the detection of a range of issues within the pipeline, such as metal loss, welding defects, cracking, bending, and temperature anomalies [14].

Despite their advantages, smart PIGs face significant challenges. Their operation necessitates trained and qualified personnel, as the pigging process which involves launching a PIG—includes steps that pose safety and integrity risks [13]. Furthermore, smart PIGs are often costly, making them financially impractical for many companies. They also lack the integration of AI tools or cameras, which could enhance the detection of defects in pipes.

This project aims to address these limitations associated with traditional PIGs, striving to make pipeline inspection more accessible, safer, and efficient.

1.2 Aims

This project aims to introduce an advanced, cost-effective in-pipe inspection robot that detects and localizes pipeline defects. The robot leverages AI, IOT, and real-time communication and notification to enable smart navigation and autonomous control, reducing manual intervention. This supports the transition to smart cities and helps prevent environmental damage.

1.3 Objectives

1. Use computer vision to classify defects.
2. Semi Autonomous motion control of the robot.
3. Real-time localization of defects in the pipeline.
4. Collect images of identified defects and store them in a database.

1.4 Proposed Solution

In this project, we propose to develop an autonomous In-Pipe Inspection Robot (IPIR) that can inspect the internal surface of oil and gas underground pipelines. By

leveraging AI, using the camera as a non-traditional NDT technique to detect defects that offer more sophisticated and efficient alternatives. Upon detecting a defect, the robot's localization capabilities enable it to locate the defect within the pipeline and wirelessly then transmit the results to a dashboard. (**real-time**) Unlike smart PIGs, Our robot is capable of self-navigating through the pipelines autonomously, without needing to rely on the internal flow for its movement.

1.5 Report Outline

- Chapter 1 (Introduction): In this chapter, an outline of the project concept is provided. This includes the problem to address, how the system will address it, the aims and objectives, and the project plan.

- Chapter 2 (Related Work): In this chapter, an examination of the proposed system is carried out, its importance in the context is determined, and relevant stakeholders are identified. In addition to a precise description of our project Next, a comprehensive analysis of previous research on oil and gas pipeline defects is performed, combining these previous efforts with the proposed solution.

- Chapter 3 (Analysis): In this chapter, a survey was conducted and sent to our target stakeholders to gather requirements. And then define the functional, the non-functional, and hardware requirements. additionally, defining the actors which are the users of the system to model a use case diagram to specify the system behavior from the users' perspective.Lastly, modeling an activity diagram that describes the actions and steps of flow in the system.

- Chapter 4 (Design): In this chapter, we designed different diagrams. We started with a class diagram that described the classes of the system and their relationships. Next, we created a sequence diagram to visualize the step-by-step interactions. Furthermore, we developed a system architecture diagram to present the relations between the software and hardware components of the project. Lastly, we designed a user-accessible system interface.

- Chapter 5 (Implementation): This chapter covers the hardware and software used in the project. It includes an overview of the hardware components and their roles, as well as the software frameworks and applications that enhance functionality. A block diagram will illustrate the system architecture, along with a circuit diagram showing the electrical connections. The developed prototype will be highlighted, emphasizing its features as a representation of the conceptual framework. Finally, implementation details will be discussed, providing insight into the methodologies employed.

- Chapter 6 (Testing): This chapter outlines the testing strategies for the Autonomous In-Pipe Inspection Robot (IPIR). Unit testing focuses on web pages and hardware components, while integration testing examines the interaction of subsystems like motion, defect detection, and localization. Validation testing assesses the robot's navigation, defect detection, and real-time data relay. A comparative analysis highlights key aspects such as autonomy and localization, followed by a performance evaluation of the system's efficiency, accuracy, and reliability, contributing to the overall assessment of the robot's effectiveness.

1.6 Project Plan

The project comprises various tasks that are assigned to each group member in an equal manner, to ensure fairness between the members. Each member was given a specific task that needed to be completed within a certain amount of time. The times and dates listed in the Gantt Chart were followed to ensure consistency and quality of the work done by the group members. The table displays the number of tasks done alongside the number of weeks it took for that specific task to be completed.

TABLE I.I

Tasks and Their Duration

NO.	Task	Start	End	Duration
1	Ch. 1 Introduction	11/2/2024	2/3/2024	3
2	Ch. 2 Related Work Introduction Literature Review Comparison Result and Feasibility Study Conclusion	3/3/2024	23/3/2024	3
3	Ch. 3 Analysis Introduction Requirements Gathering Use case Diagrams Conclusion	24/3/2024	13/4/2024	3
4	Ch. 4 Design Introduction Class DiagramSystem Architecture Conclusion	14/4/2024	4/5/2024	3
5	Ch. 5 Implementation Details of Hardware Block Diagram Implementation Details Conclusion	15/9/2024	19/10/2024	5
6	Ch. 6 Testing Scenario Details Testing Details Comparative Analysis Performance Analysis Conclusion	20/10/2024	23/11/2024	5

1.6.1 Time Constraint

A time constraint is an important limitation in project management that must be considered. It specifies the start and end dates for the entire project and for each individual task[15]. To effectively illustrate this timeline and provide a clear visual representation of the project's schedule, we utilized a gantt chart. This chart helps outline the precise tasks needed for every chapter and step in the project, emphasizing not only what needs to be done but also how long each activity should take. Making it easier to track progress and guarantee that all required tasks are finished on schedule. For Senior Project I, Fig. 1.1 and Fig. 1.2 detail the gantt chart. In this chart, time is represented on the vertical axis, spanning from June 14 to May 5, covering a total of 17 weeks.

As shown in Fig. 1.1, each row corresponds to specific tasks and phases of the project, with horizontal bars illustrating the duration of each task. The initial tasks of Senior Project I include problem identification and team formation, the introduction (Chapter 1), and related work (Chapter 2).

Task ID	Task Name	Start Week	End Week	2024																	
				JAN 14	JAN 21	JAN 28	FEB 4	FEB 11	FEB 18	FEB 25	MAR 3	MAR 10	MAR 17	MAR 24	MAR 31	APR 7	APR 14	APR 21	APR 28	MAY 5
				WK01	WK02	WK03	WK04	WK05	WK06	WK07	WK08	WK09	WK10	WK11	WK12	WK13	WK14	WK15	WK16	WK17
01	Problem identification and team formation	WK01	WK04																	
01.01	Choose team members	WK02	WK02																	
01.02	Talk to potential supervisors	WK02	WK02																	
01.03	Finalaize the topic,aims,and scope	WK03	WK04																	
01.04	Presentation	WK04	WK04																	
02	Introduction (Chapter 1)	WK05	WK07																	
02.01	Describing the scope and the problem	WK05	WK05																	
02.02	Defining aims and objectives	WK05	WK05																	
02.03	Describing the proposed solution	WK06	WK06																	
02.04	Setting the report outline	WK07	WK07																	
02.05	Preparing project plan	WK07	WK07																	
03	Related work (Chapter 2)	WK08	WK10																	
03.01	Defining stakeholders'	WK08	WK08																	
03.02	Describing the project's background	WK08	WK09																	
03.03	Literature review	WK09	WK09																	
03.04	Defining the comparison criteria	WK10	WK10																	
03.05	Comparing the results	WK10	WK10																	

Figure 1.1. Gantt chart for chapters 1 and 2

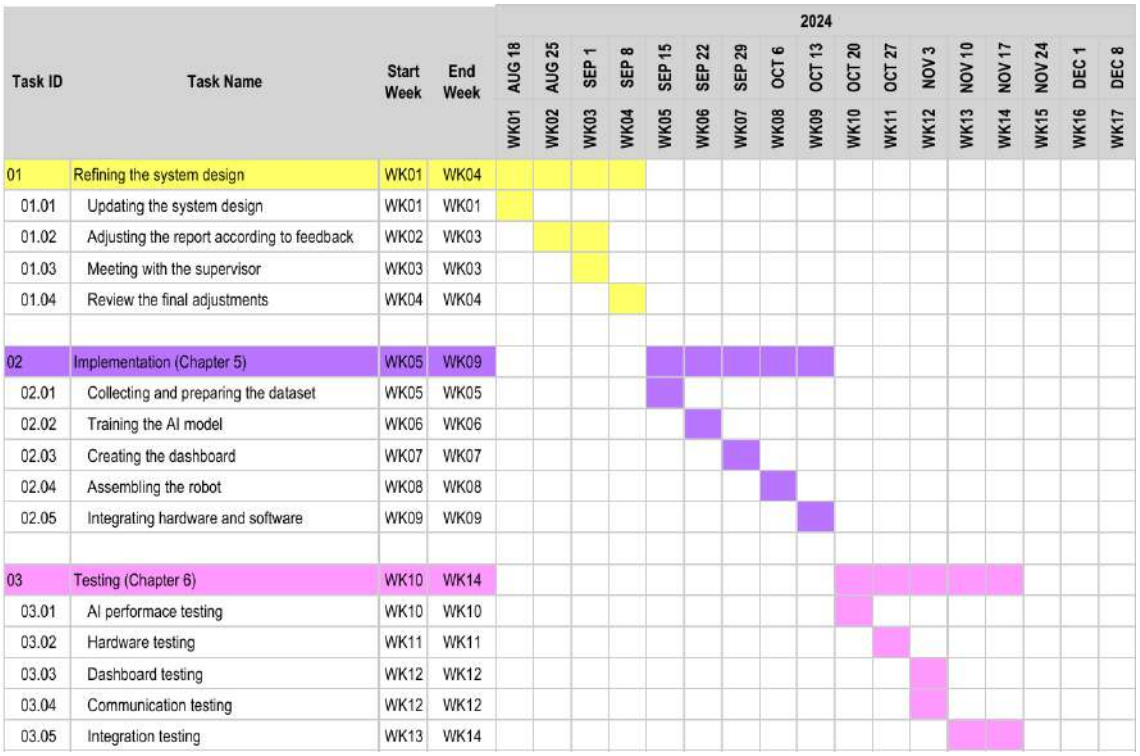
Fig.1.2 presents the subsequent tasks, including analysis (Chapter 3), design (Chapter 4), and the final presentation, each aligned with their respective start and end dates.

04	Analysis (Chapter 3)	WK11	WK13
04.01	Gathering requirements	WK11	WK12
04.02	Defining functional and non-functional req.	WK12	WK12
04.03	Defining hardware requirements	WK12	WK12
04.04	Defining actors	WK12	WK12
04.05	Drawing the use case diagram	WK13	WK13
04.06	Drawing the activity diagram	WK13	WK13
05	Design (Chapter 4)	WK14	WK16
05.01	Drawing the class diagram	WK14	WK14
05.02	Drawing the sequence diagram	WK14	WK14
05.03	Specifying the system archeticture	WK15	WK15
05.04	Designing the interfaces	WK15	WK16
06	Final report and presentation	WK17	
06.01	Submitting the report to our supervisor	WK17	
06.02	Implementing modifications		
06.03	Submitting the report to the coordinator		
06.04	Final presentation		

Figure 1.2. Gantt chart for Chapters 3 and 4

For Senior Project II, Fig. 1.3 and Fig. 1.4 detail the Gantt chart. In this chart, time is represented on the vertical axis, spanning from August 18 to December 8, covering a total of 17 weeks.

Fig. 1.3 illustrates the major tasks and phases of the project. Each row corresponds to specific tasks, with horizontal bars indicating the duration of each. The key tasks include refining the system design, implementation (Chapter 5), and testing (Chapter 6).



Task ID	Task Name	Start Week	End Week
01	Refining the system design	WK01	WK04
01.01	Updating the system design	WK01	WK01
01.02	Adjusting the report according to feedback	WK02	WK03
01.03	Meeting with the supervisor	WK03	WK03
01.04	Review the final adjustments	WK04	WK04
02	Implementation (Chapter 5)	WK05	WK09
02.01	Collecting and preparing the dataset	WK05	WK05
02.02	Training the AI model	WK06	WK06
02.03	Creating the dashboard	WK07	WK07
02.04	Assembling the robot	WK08	WK08
02.05	Integrating hardware and software	WK09	WK09
03	Testing (Chapter 6)	WK10	WK14
03.01	AI performace testing	WK10	WK10
03.02	Hardware testing	WK11	WK11
03.03	Dashboard testing	WK12	WK12
03.04	Communication testing	WK12	WK12
03.05	Integration testing	WK13	WK14

Figure 1.3. Gantt chart for chapters 5 and 6

Fig. 1.4 presents the final tasks, which include the conclusion, creating the poster, and preparing the final report. Each task is aligned with its respective start and end dates. This clear visual representation helps track progress and identify overlapping tasks, ensuring that all components of the project are completed on schedule.

Figure 1.4. Gantt chart for finals tasks

1.6.2 Methodology

the methodology that we adopt is the waterfall model as shown in Fig. 1.5 The Waterfall model [16] is a traditional software development methodology characterized by a linear progression of phases: requirements gathering, system design, implementation, testing, deployment, and maintenance. Each phase must be completed before moving on to the next, making it less adaptable to changing requirements compared to more iterative approaches like Agile.

1.6.3 Project Phases

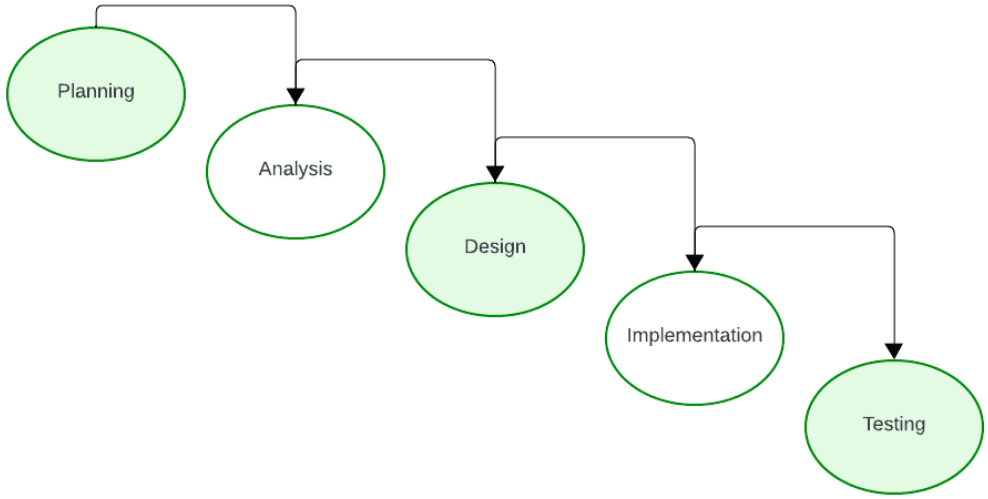


Figure 1.5. Project phases

- Planning
  - Choosing a project supervisor.
  - Building a Team of 5 members.
  - Choosing an idea for the project.
  - Determine aims and objectives.

- Analysis
  - Define stakeholders.
  - Gather data from stakeholders.
  - Define functional and non-functional requirements.
  - Draw use case and class diagram.
- Design
  - Design interface for the system.
  - Design system architecture.
  - Design the prototype.
- Implementation
  - Collect the selected images necessary for datasets.
  - Connect hardware components.
  - Integrate system components.
  - Finalize the system prototype.
- Testing
  - Test hardware connectivity.
  - Test dashboard results.
  - Test communication connection.
  - Test database connectivity.
  - Test the connectivity between software and hardware.
  - Test the hole system.

1.7 Conclusion

This chapter presents a project to identify autonomous robots for pipeline defect detection. We then defined the problem, aims, and project outline and finally finalized the project plan for our system. In the next chapter, we will discuss the stakeholders' Definition and description of the Background, review the related works, and conduct the feasibility study.

Chapter 2

Related work

2.1 Introduction

In this chapter, we will commence by identifying the relevant stakeholders In section2.2, followed by a comprehensive overview of the contextual background In section2.3. This initial framework will set the stage for a thorough literature review In section2.4, where we will critically analyze existing research and methodologies in the field. Subsequently, we will undertake a detailed comparison of the various methods employed in the literature In section2.5 against our proposed solution Section 2.6. This comparative analysis will not only clarify the strengths and weaknesses of the current approaches but also underscore the unique contributions of our proposal.

2.2 Stakeholders' Definition

The main stakeholders are a transportation industry company that works in oil and gas pipelines. these companies could be transporting materials between cities and countries, or manufacturing different types of pipelines.

2.3 Detailed Description of the Background

Pipeline inspection robots (PIRs) are advanced tools designed for monitoring and maintaining pipelines. As shown in Fig. 2.1 Key aspects include communication, which ensures real-time data transfer to operators; a mechanical structure built for durability

and maneuverability in challenging environments; and intelligent inspection features that use sensors and imaging to detect issues like cracks. Additionally, navigation systems enable the robots to move autonomously through pipelines, reaching areas difficult for human inspectors. Together, these capabilities enhance the efficiency and safety of pipeline management.

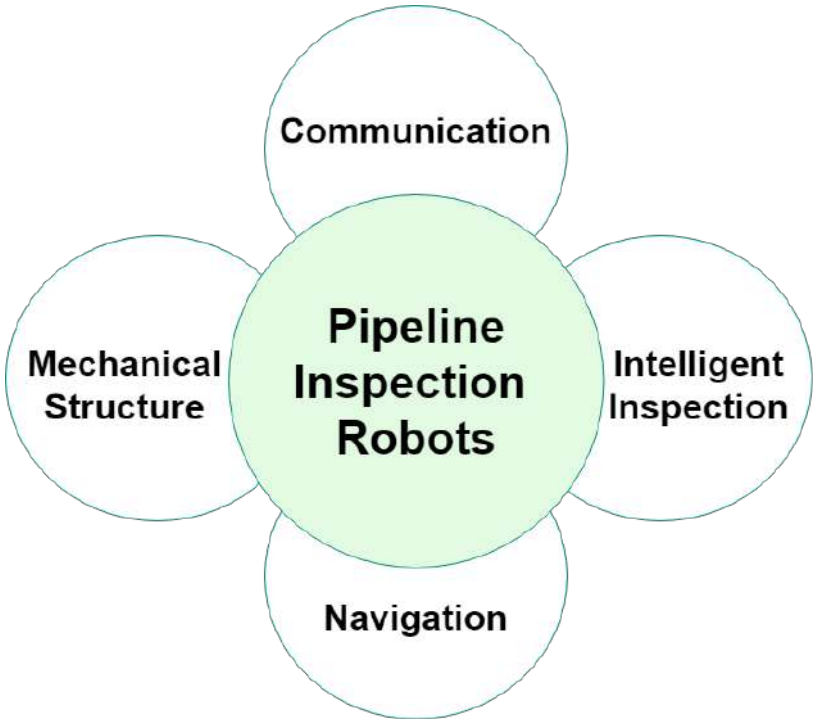


Figure 2.1. Diagram of key aspects of PIRs

2.3.1 Mechanical Structure

The mechanical structure of in-pipe inspection robots (IPIR) is a critical factor in assessing their functionality and effectiveness. The design of an IPIR significantly impacts its ability to navigate complex environments, withstand challenging conditions, and perform inspection tasks. These robots can be classified into six distinct categories, each tailored to specific operational needs [1]:

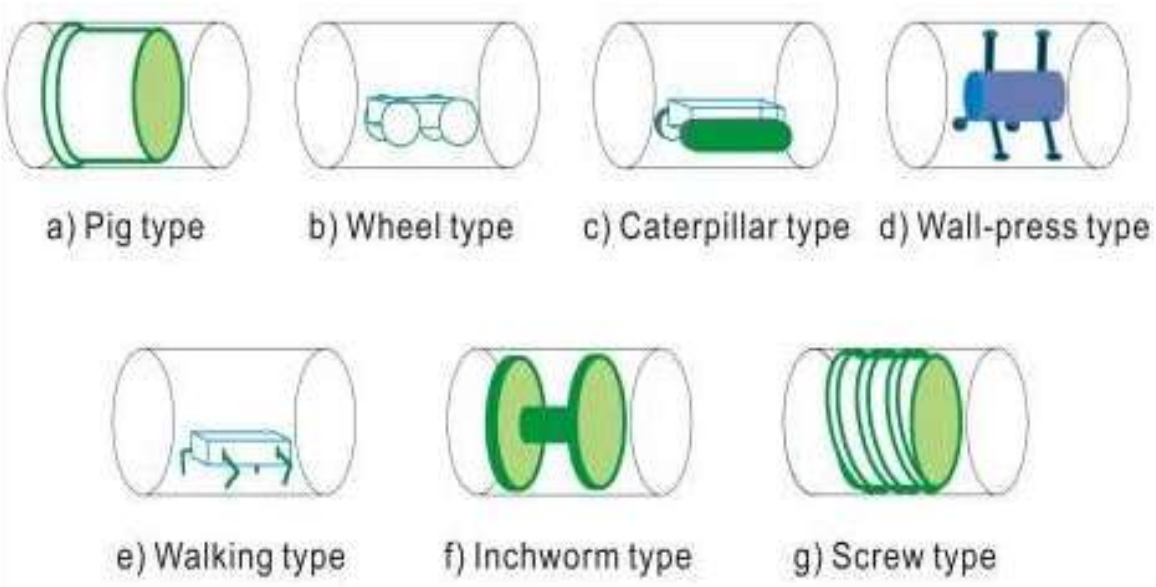


Figure 2.2. classification of in-pipe robots [1]

1. PIG: pipeline inspection gauges are large tools most employed type in large pipelines, they move with the flow inside the pipeline [12].
2. Wheel robot type: these robots are quite mobile in comparison to other types and are similar to simple mobile robots in that they travel inside pipelines by just rotating their wheels [17].
3. Caterpillar robot type: uses track wheels to adapt to the inner pipeline. This type of robot is used in conditions that demand much more grip on the walls of the pipeline [1].
4. Wall-Press robot type: this type has sufficient force to ensure adherence to the pipeline walls. Mostly used in vertical pipelines to prevent the robot from falling down [1].
5. Walking robot type: this type has a complex mechanism and walks on its legs, which reduces slipping and surface damage to the pipeline [17].
6. Inchworm robot type: this type of robot mimics the movement of a worm and is preferred for smaller diameter pipelines [1].
7. Screw robot type: this type resemble a screw motion used in pipelines without damaging the pipeline surface [17].

Nearly all in-pipe inspection robots utilize mechanisms derived from one of the above-mentioned types or their combinations. [1]. Our robot's design takes inspiration from the wheel-type mechanism.

2.3.2 Intelligent Inspection

Pipe inspection benefits greatly from machine vision and artificial intelligence (AI), which increase the process's precision and efficiency. It involves gathering and analyzing visual data using cameras, sensors, and the Image Processing Module. This module works by acquiring images of the pipeline, extracting the appropriate color planes, filtering them, and then thresholding the images to remove the background pipe. The remaining part of the image represents the obstacles [18]. Pipeline integrity is ensured through the critical process of machine vision robot inspection, which helps detect flaws. These robots can quickly discover and identify problems by capturing high-resolution images and analyzing them in real-time [19].

2.3.3 Underground IPIR Communication Techniques

The communication capabilities of underground pipeline inspection robots have undergone significant development. Initially, these robots relied on manual recordings, entering pipelines equipped with inspection sensors and cameras to gather data, which could only be reviewed after they exited. This process was similar to the functionality of standard Pipeline Inspection Gauges (PIGs) during their early implementation [20]. This method was time-consuming and labor-intensive, highlighting the need for real-time inspection solutions. To address these challenges, the industry transitioned to wired communication technologies. One approach utilized Power Line Communication (PLC) to transmit both data and power through existing electrical lines[21]. Another development involved a robot that employed customized tether cables for data transmission using optical fiber [22]. This method has become a common form of communication in pipeline inspection robots [23]. Despite these advancements, managing wires within pipelines and navigating different pipe branches remained challenging. The limitations of tethered cables highlighted the need for wireless technology in inspection robots [23]. The increasing demand for real-time data communication has led to the rise of the Internet of Underground Things (IoUT), which extends the principles of the Internet of Things (IoT) to underground applications. IoUT incorporates

various technologies that have been tested across different fields. For instance, it is being explored in smart lighting systems [24], in Agriculture 4.0 to optimize farming practices [25], and in remote health monitoring using LoRa technology [26]. In the oil and gas sector, Patel et al. [27] developed a hybrid communication system utilizing Zigbee and LoRa technologies to monitor oil pipelines, which has proven effective for efficient data transmission. Additionally, a pipeline inspection robot developed for the China National Petroleum Corporation utilized LoRa to control the robot remotely when in manual mode, achieving a communication range of 500 meters [28].

2.3.4 Navigation

Navigation in robotics integrates automation and localization to enable robots to operate efficiently. Effective navigation is essential for enabling robots to perform tasks independently in complex environments.

Automation: autonomous robot is a machine that can perform tasks without direct human control. Automation control of robots involves utilizing advanced technologies and programming to enable robots to perform tasks autonomously without constant human intervention. This can be achieved through the use of sensors and AI algorithms. Overall, automation control of robots plays a crucial role in increasing productivity [29].

Localization: robots having sensors and other technologies fitted to enable autonomous navigation and operation inside a designated area are known as localization robots. These robots locate themselves using a variety of techniques, including ultrasonic sensors [30].

2.4 Literature Review

2.4.1 Sensor-Based Inspection Robots

This article was published in 2022 is talks about "In-Line Detection of Defects in Steel Pipes Using Flexible GMR Sensor Array" introduced an innovative in-line robotic system designed for the detection of defects in steel pipes [31]. This robot employs a non-destructive approach utilizing the magnetic flux leakage (MFL) technique for flaw evaluation, which is particularly effective for ferromagnetic materials and suitable for both quantitative and qualitative defect assessment. The system features a flexible GMR sensor array made up of six sensors, known for their high sensitivity to flux leakage even under elevated temperature conditions. Data from the sensor array is collected using an Arduino UNO, which plots the information as waveform graphs. To test the system, artificial defects such as axial holes, circumferential holes, rectangular slots, Loss of Metallic Area (LMA) defects, and corrosion were intentionally created in a 6-inch-diameter steel tube. The robot's chassis is constructed from polylactide (PLA) filament through 3D printing, significantly reducing its weight compared to a traditional metal chassis. However, despite of this robot using non-destructive techniques it is important to note that this robot does not incorporate artificial intelligence or camera features for defect detection, which limits its automation capabilities. As a result, it requires constant human supervision, reducing overall efficiency by necessitating manual intervention for defect identification.

Another article was published in 2022 addresses the "Long Distance Inspection for In-pipe Robots in Water Distribution Systems with Smart Navigation", and utilizing of a predefined map and the particle filtering algorithm (PF) for robot localization. The robot equipped with three adjustable arms for varying pipe sizes, communicates wirelessly but without mentioning the technique. PF proves valuable for parameter estimation with non-Gaussian noise, with higher accuracy achieved through measuring a large number of random points and updating particle weights based on ultrasonic readings [32]. By integrating these features, offering promising solutions to long-standing challenges in the field. A notable drawback is the limited hardware description, it limits the ability to fully understand the structure of the represented robot. Since the robots lack AI, it can not make real-time adjustments based on sensor data, necessitating constant human supervision and manual control. In addition to disadvantages is the missing dashboard for providing updates and information for the operator.

2.4.2 AI Inspection Techniques

Another significant study is the 2018 paper titled "Defect Detection in Oil and Gas Pipeline: A Machine Learning Application" [33]. This research focuses on the application of machine learning (ML) to enhance defect detection in oil and gas pipelines, specifically analyzing data collected from a Pipeline Inspection Gauge (PIG) device. The study explores several ML algorithms, such as Support Vector Machines (SVM), Gradient Boosting Machines (GBM), and Random Forest, to identify the most effective model for detecting defects. One of the key strengths of this approach is its rigorous focus on accuracy and minimizing false positives, which is vital for maintaining pipeline safety. The paper also emphasizes the importance of cross-validation and iterative fine-tuning to ensure the robustness and reliability of the models. Notably, SVM is highlighted for its ability to effectively distinguish between defects and non-defects, thereby enhancing the overall efficacy of the defect detection process. However, there are some limitations to this approach. The machine learning models are applied post-inspection, meaning that defect identification occurs only after the data has been collected by the PIG device. This can lead to delays between the detection of potential issues and their resolution, which could be critical in dynamic pipeline environments where real-time responses are necessary. In comparison, our robot addresses these limitations by integrating AI directly into the inspection process. This allows for real-time classification and response to defects as they are detected, which is a significant advantage over the post-processing approach used in the 2018 study. While the machine learning models in the 2018 paper provide a robust method for defect detection, our approach enhances this by enabling immediate, on-the-spot analysis, reducing the time between data collection and actionable insight.

In 2022, a paper titled "Robust Sewer Crack Detection with Text Analysis Based on Deep Learning" introduced an advanced automated framework for detecting cracks in sewage pipes using deep learning techniques [34]. The system integrates an attention mechanism, an improved YOLOv5 architecture, and location information from CCTV videos, allowing for crack detection in challenging environments, including pitch-dark sewers. It features a micro-scale detection mechanism and a convolutional block attention module to enhance accuracy and includes a large dataset for the 12 most common crack types. Text recognition is performed using the Pre-trained TPS-ResNet-BiLSTM-Attn (TRBA) framework, involving transformation, feature extraction, sequence modeling, and prediction. Experimental results

achieved a mean average precision (mAP) of 75.9%. Advantages Utilizes an advanced attention mechanism and improved YOLOv5 architecture for enhanced detection; includes a micro-scale detection feature and a convolutional block attention module; large dataset for the 12 most common crack types; effective text recognition with the TRBA framework. Disadvantages Lack details on the hardware used; real-time detection capability was not fully realized; not applicable to oil and gas pipelines.

2.4.3 Real-time Inspection Robots

An article published in 2018 talks about the "design and development of an inspection robot for oil and gas applications", it enhances the caterpillar model type, employs an Arduino Uno microcontroller for processing, along with three DC motors for autonomous movement control and Communicates through Bluetooth modules [35]. Integrating these features offers promising solutions to long-standing challenges in the field. A notable drawback is the limited localization. It limits the ability to localize the robot in the pipeline environment. Since the robots lack AI, they can not make real-time adjustments based on sensor data, necessitating constant human supervision and manual control.

Another research paper that came out in 2018, titled "Design of Reconfigurable In-pipe Exploration Robots" was published, focusing on robots designed to navigate complex pipe structures and adapt to inner deformities [18]. These robots use counter wheels for localization and HD cameras for visual inspection to detect pipe deformities. Equipped with distance sensors and an IMU sensor (MPU6050), they assess the pipe structure and orientation. The robots also feature gas sensors (MQ3) and a temperature sensor (LM35) to detect hazardous gases like methane and butane. Image processing techniques, such as thresholding, help isolate clogs and obstacles, indicating if there is a complete blockage. Encoders measure the distance traveled inside the pipe, and the system is built on the NI myRIO board, allowing wireless to communicate and monitoring. Advantages include their ability to navigate complex pipes, adjust to deformities, and use various sensors for accurate localization and hazard detection. Disadvantages are their limitation to detecting only obstacles without identifying cracks or corrosion.

Another relevant study in the field of defect identification with sensors is the 2021 paper titled "PIRATE: Design and Implementation of Pipe Inspection Robot" [36]. This paper delves into the challenges associated with inspecting pipelines, especially those with

complex configurations such as C, L, and T bends. The PIRATE robot is distinguished by its innovative design, which includes an Octo-naked structure and helical spring-based mechanisms that allow it to maneuver with remarkable precision through intricate pipe layouts. The paper emphasizes the robot's advanced sensing capabilities, which are critical for identifying defects within pipelines. It is equipped with a CCD camera for capturing high-resolution visuals, a Time of Flight (ToF) sensor to detect obstacles, and an ultrasonic sensor for precise positioning within the pipe. These features are complemented by WiFi communication, enabling real-time data transfer during inspections, which is essential for monitoring and decision-making. However, despite these innovations, the PIRATE robot has some significant limitations, particularly in terms of autonomy. The absence of AI limits its ability to adapt to unforeseen changes in the pipeline environment, requiring constant human supervision and manual intervention. This lack of autonomy can be a major drawback in dynamic inspection scenarios where real-time adjustments are crucial. In contrast, our robot addresses these limitations by integrating both caterpillar-type and wheel-type mechanisms, which enhance its versatility in navigating various pipe conditions. Furthermore, our robot employs WiFi technology, which provides a reliable communication solution for real-time data transfer, particularly in challenging environments. A major advancement in our design is the incorporation of AI for intelligent inspection. This allows our robot to autonomously classify defects and make real-time decisions based on sensor data, significantly reducing the need for human intervention. The ability to precisely localize defects and provide detailed reports further enhances the robot's utility in pipeline inspections. Our robot's hardware setup, including the Raspberry Pi 3, VO5647 camera, ultrasonic sensors, and encoders, supports these advanced functionalities, ensuring accurate defect detection and effective pipeline navigation. This combination of AI, robust hardware, and reliable communication technology positions our robot as a more capable and autonomous solution compared to the PIRATE model, addressing many of the challenges outlined in the 2021 paper.

In 2021, another paper titled "Application of Computer Vision in Pipeline Inspection Robot" was published [19], focusing on the development of a screw-type in-pipe inspection robot (IPIR) designed to detect defects within pipelines such as cracks, corrosion, and blockages. The robot employs the YOLOv3 object detection model, enabling real-time identification of defects with an accuracy of 91% even in low-light conditions. It navigates autonomously using a passive screw-based adhesion and locomotion mechanism. However, there are notable drawbacks, as it can only detect defects without classifying their types.

Additionally, while the robot can navigate straight sections and bends of pipelines, it struggles with more complex structures like T-joints and Y-joints.

Another paper came out at 2023 called "Design and Development of an In-Pipe Mobile Robot for Pipeline Inspection with AI Defect Detection System" presents an advanced system that integrates AI for real-time defect classification [37].Central to this system is the YOLOv8n deep learning model, which processes images captured by an ESP32 CAM as the robot navigates through pipelines. This model excels at detecting and classifying various pipeline defects such as corrosion, cracks, holes, and welding flaws, ensuring rapid identification and response. A key component of their system is the extensive database used to train the AI model. The dataset comprises images and videos sourced from diverse environments, including industrial settings and AI-generated imagery. These images are meticulously labeled to represent different types of pipeline defects. The database is stored on Ultralytics Hub, a cloud-based platform known for its secure and scalable storage capabilities. The system includes an Android application that allows operators to control the robot remotely and view live video feeds from the ESP32 CAM. This app not only facilitates real- time monitoring but also enables immediate data capture and defect classification during inspections. The combination of real-time AI processing, cloud-based data storage, and mobile accessibility makes this system highly effective for pipeline maintenance.

2.5 Comparison Criteria Definition

- Automation

  An autonomous robot is a machine or device that is capable of performing tasks or actions without direct human control or intervention Autonomous robots can operate independently, navigate through complex environments, adapt to changing conditions, and complete tasks or missions without continuous human supervision. They are designed to perform specific functions or activities efficiently and effectively, enhancing productivity, safety, and convenience in various industries and applications.
- Localization
  - Defect localization

    The process of determining the root cause of a defect or problem, which includes analyzing the code and other information to determine where the defect exists. This is an important step in discovering defects in the pipes, which leads to their rapid repair.
  - Robot localization

    Robot localization refers to the process of determining the position of a robot in its pipeline. This is typically done using sensors and algorithms or sometimes specific path to estimate the robot's location relative to a known map or landmarks.
- Use of AI in defect detection

  Considering whether AI has a role in defect detection, including its use for defect identification and classification. Moreover, assessing the particular AI technique applied.
- Hardware Used (camera and micro-controller type)
- Type of pipeline
- Dashboard communication

2.6 Comparison Result and Feasibility Study

TABLE II.I

Comparison Result

REF.	Type of pipeline	AI	Type of defect	Localization	Communication	Automation	Microcontroller	Real-Time Detection	Dashboard Availability	Camera
[38]	Oil & Gas	NOT mentioned	crack and corrosion	NOT mentioned	Bluetooth modules	Yes	Arduino UNO	NO	NOT mentioned	NOT specified
[18]	Gas	yes image processing	obstacle	Yes, Encoder	WI-FI, NI myRIO	Yes	yes NI myRIO	Yes	Not mentioned	Yes, webcam
[19]	Industrial	Yes , YOLOv3	cracks, corrosion, and blockage	No	Not mentioned	Yes	Raspberry pi 4	Yes	Not mentioned	Raspberry pi camera.
[36]	Sewer	No	NOT mentioned	Yes, UltraSonic.	WI-FI	NO	Raspberry pi	yes	yes, computer GUI.	Raspberry Pi camera.
[34]	Steel	NO	axial and circumferential holes, rectangular slots, Loss of Metallic Area defects, and corrosion	GMR (Giant magneto resistance) sensors waveform graph represent defect region	Not mentioned	NO	Arduino UNO	NO	No	NO
[37]	Gas	Yes , YOLOv8	cracks, corrosion, holes, waste, and weldings	No	WI-FI	No	esp32	Yes	Available	esp32 cam
[33]	Oil & Gas	Yes, machine learning	Metal loss, shape change and holes	No	WI-FI	No	No	No	No	Yes
[32]	Water	No	Not mentioned	Partical Filtering	Not mentioned	Not mentioned	Not mentioned	Not mentioned	No	No
[34]	Sewer	Yes, YOLOv5	Crack	No	No	No	No	No	No	No
OUR PROJECT	Oil & Gas	Yes	Crack and corrosion	Yes	Yes	Yes	Raspberry pi	Yes	Yes	Raspberry pi camera

A variety of features can be compared between our robot and the previously mentioned systems, as shown in the comparison table. Many existing robots do not include AI-based defect classification is a capability that our robot offers by both capturing and categorizing defect types. Our robot integrates numerous functionalities into one unit, including autonomous movement, real-time defect localization with notifications to the supervisor, the use of a night camera for capturing and classifying defects through image processing, and the ability to present the collected data on a dashboard.

2.7 Conclusion

In summary, this chapter establishes a structured approach to understanding the landscape of stakeholders and contextual factors relevant to our research. By conducting a thorough literature review and comparing existing methodologies with our proposed solution, we aim to highlight both the limitations of current practices and the innovative aspects of our approach. This comprehensive analysis will provide a solid foundation for the subsequent sections, enhancing the understanding of our contributions to the field.

Chapter 3

Analysis

3.1 Introduction

This chapter will concentrate on delineating the project requirements, commencing with an extensive collection of requirements In section 3.2. We will identify both functional requirements In section3.3 and non-functional requirements In section3.4, along with specifying the necessary hardware requirements In section3.5. Additionally, we will explore the definition of actors involved in the project In section3.6, which will provide insight into the various stakeholders and their roles. To further illustrate the system's operations, we will include a detailed description of the use case diagram In section3.7 and the activity diagram In section3.8. By systematically addressing these components, we aim to provide a comprehensive understanding of the project's foundational requirements, ensuring clarity and alignment among all participants. This thorough examination will facilitate informed decision-making and contribute to the successful development and implementation of the system.

3.2 Requirements Gathering

Data gathering is the process of collecting and measuring information on targeted variables in an established system, which then enables one to answer relevant questions and evaluate outcomes. This section delves into the analysis of responses gathered through a Google Survey titled "Autonomous Robot for In-Pipe Defect Detection." This survey was distributed to a targeted audience within the industry, including engineers, supervisors, and workers or anyone related to this field. The primary goal was to understand the views of specialists and experts in the field about the potential application of autonomous robots to detect defects inside pipes.

We inquired about their age, 35.1% of them responded with the range of 40-49 years as shown in Fig. 3.1 .

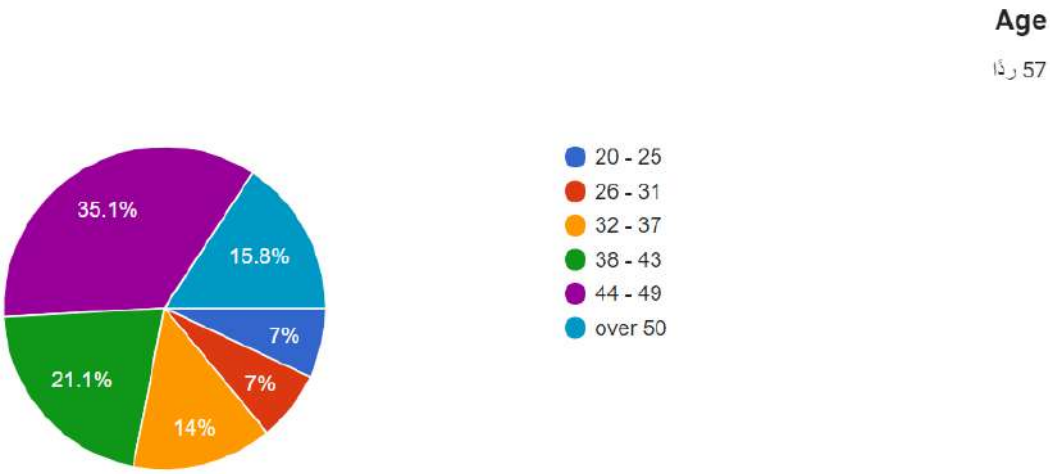


Figure 3.1. Age of survey participants

And 96.5% of them are males as shown in Fig. 3.2.

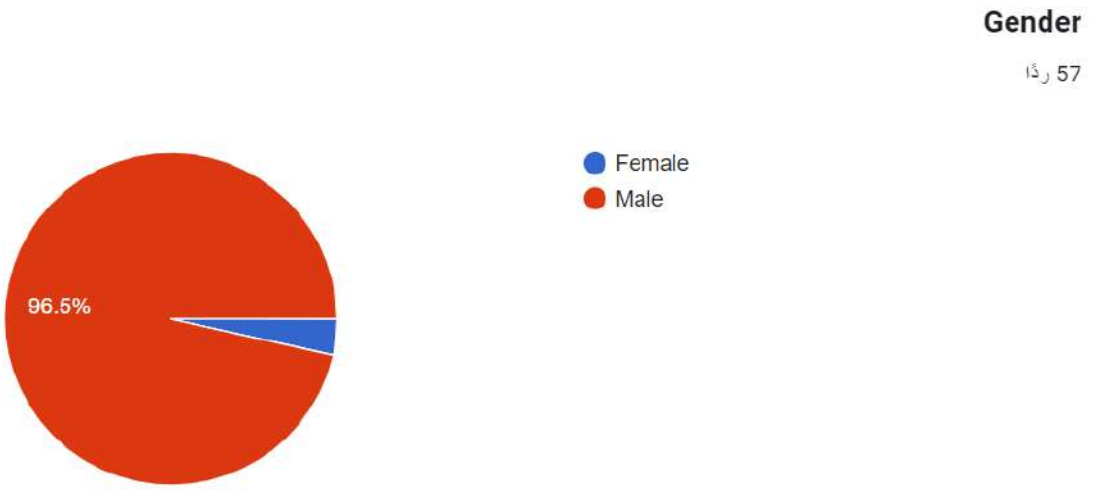


Figure 3.2. Gender of survey participants

38.6% works in the pipeline transportation industry field as shown in Fig. 3.3.

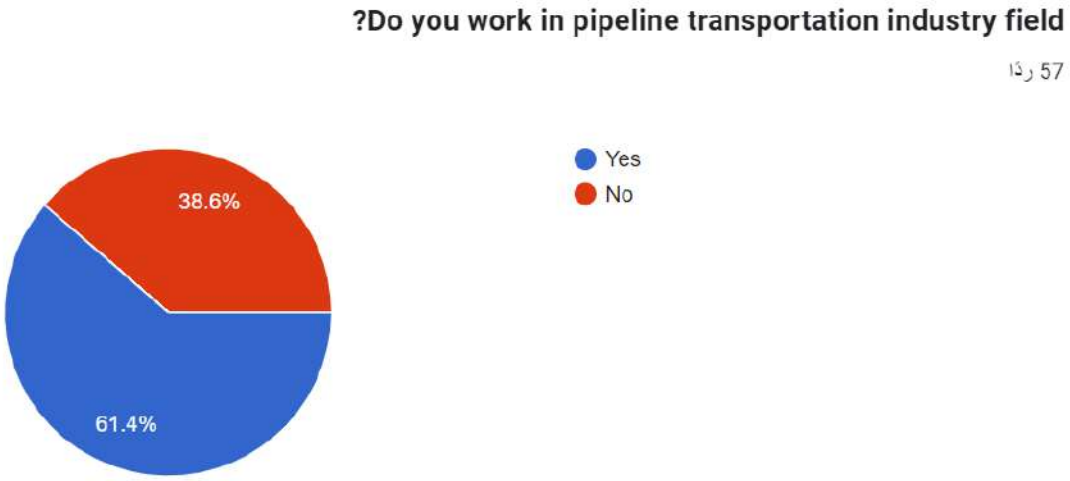


Figure 3.3. Job of survey participants

Besides 80.7% of them are familiar with the pipeline transportation industry field as shown in Fig. 3.4.

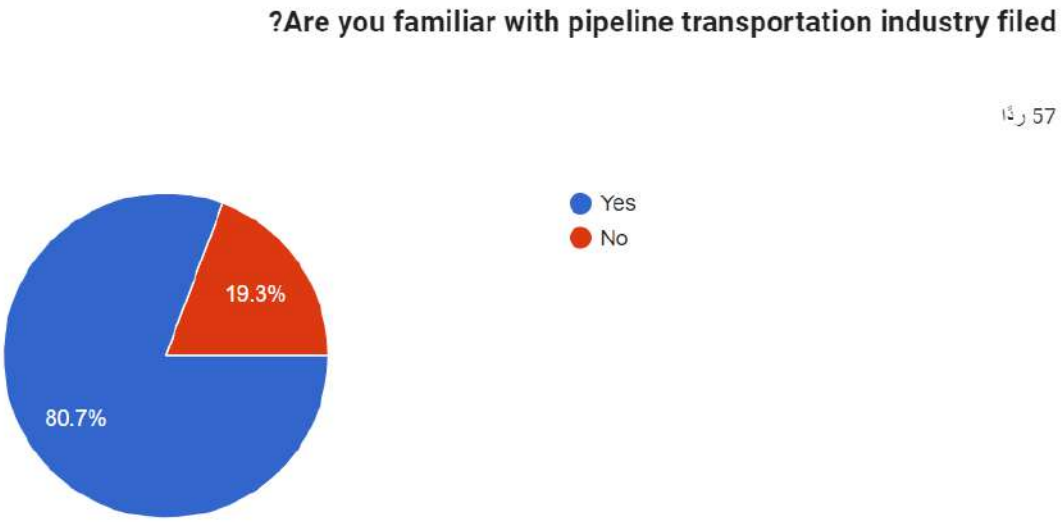


Figure 3.4. Familiarity with the pipeline for most of the survey participants

38.6% as shown in Fig. 3.5 are not familiar with the technology and principles behind autonomous robots used for in-pipe defect detection

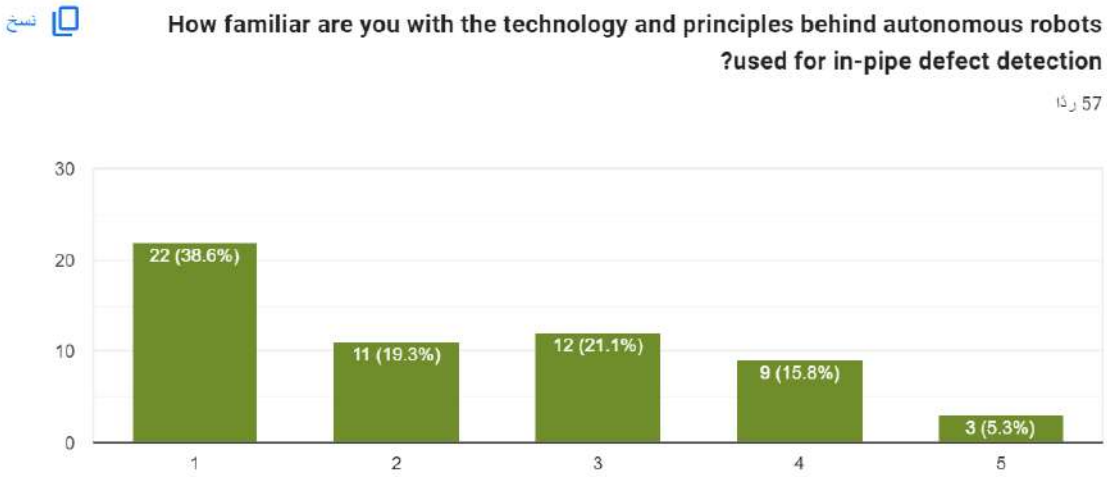


Figure 3.5. familiarity with the technology and principles behind autonomous robots

As shown in Fig. 3.6, The most recognized defects are crack and corrosion, with corrosion receiving 48 votes and crack garnering 44 votes, highlighting a strong familiarity with these issues.

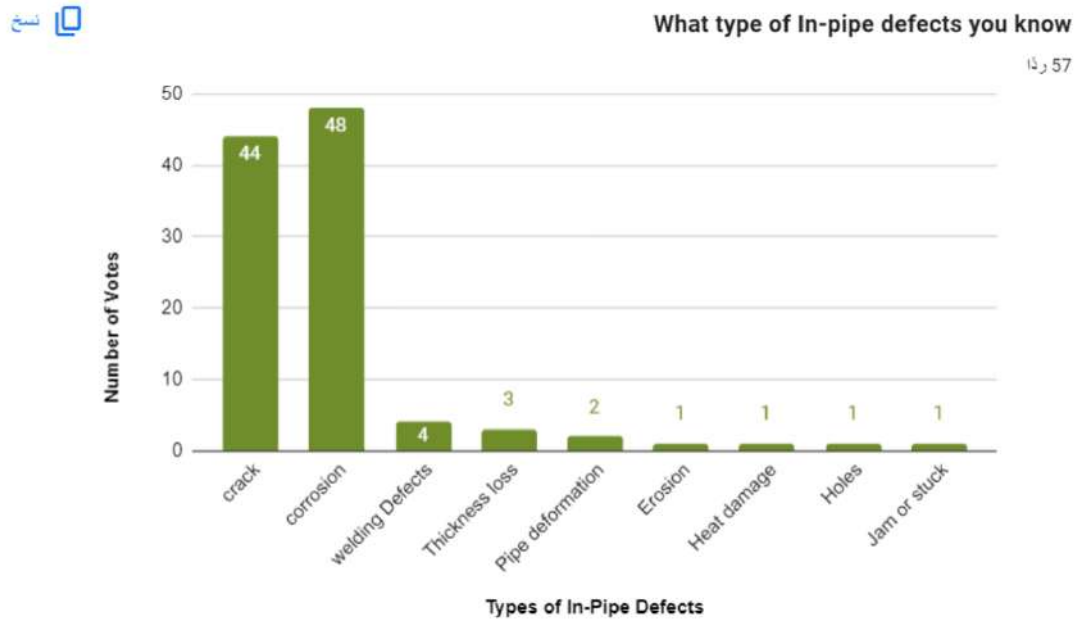


Figure 3.6. Type of defects opinion of survey participants

68.4% as shown in Fig. 3.7 agree that an autonomous robot could improve the efficiency of in-pipe defect detection compared to traditional methods.

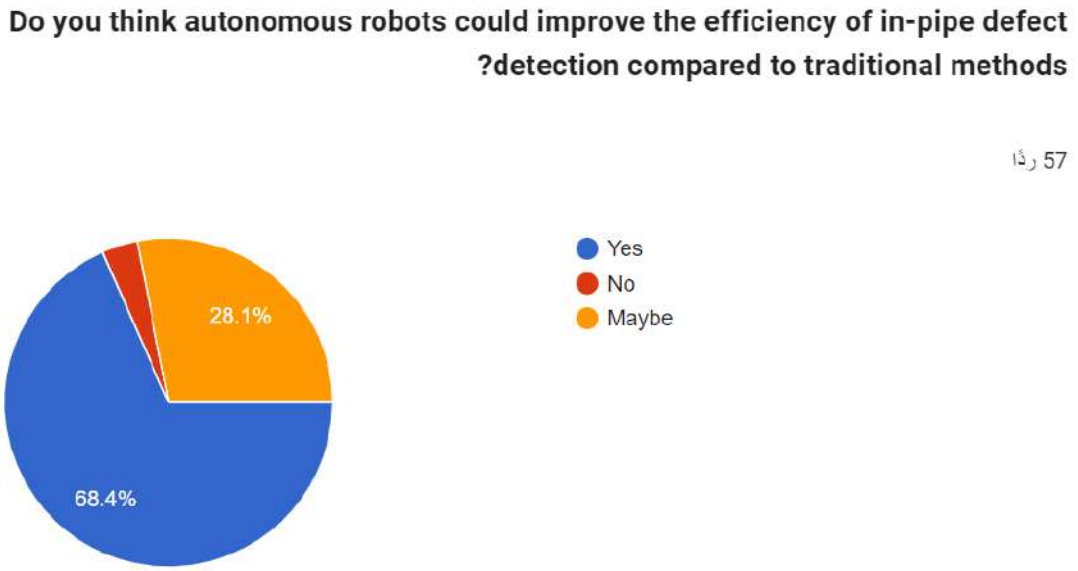


Figure 3.7. Efficiency of in-pipe defect detection traditional vs Autonomous from survey participants opinion

57.9% of the participants as shown in Fig. 3.8, think the dashboard design effectively helps in monitoring the robot's outcomes in real-time.

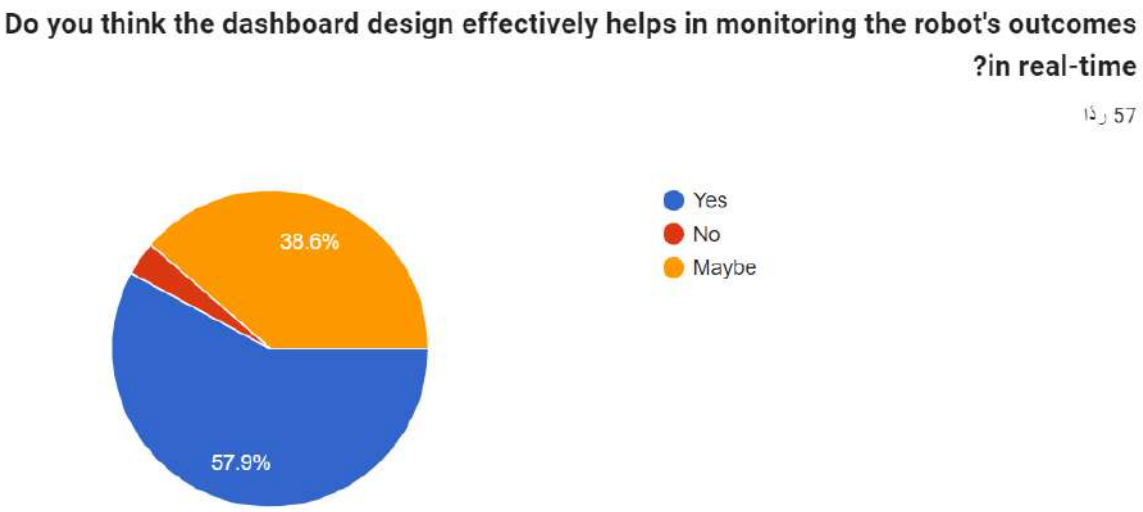


Figure 3.8. Dashboard design efficiently from survey participant's opinion

36.8% as shown in Fig. 3.9 agree that The cost of implementing autonomous robots is significantly lower than traditional inspection methods.

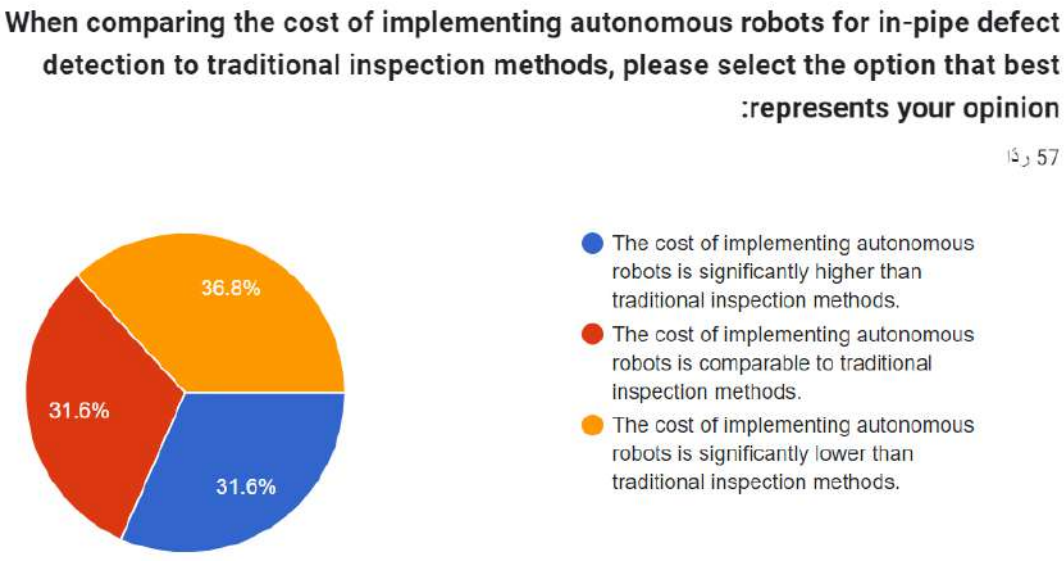


Figure 3.9. Comparing cost from survey participants opinion

56.1% as shown in Fig. 3.10 comply with The time duration of implementing autonomous robots is significantly lower than traditional inspection methods.

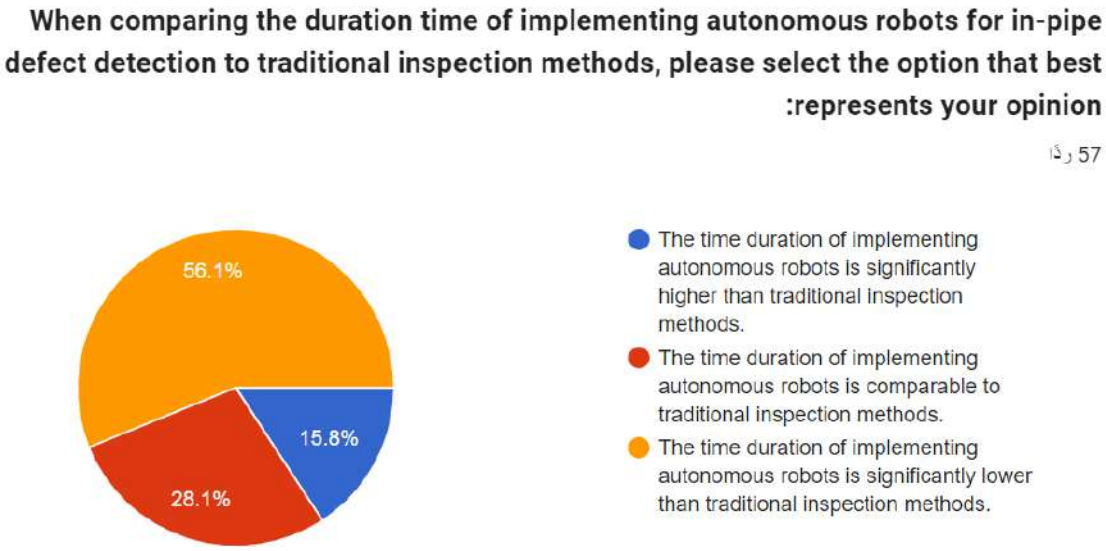


Figure 3.10. Duration time autonomous robot vs traditional methods inspection from survey participants opinion

% 35.1 as shown in Fig. 3.11 chose the scale 3 out of 5 of how confident in the ability of autonomous robots to accurately detect defects within pipes.

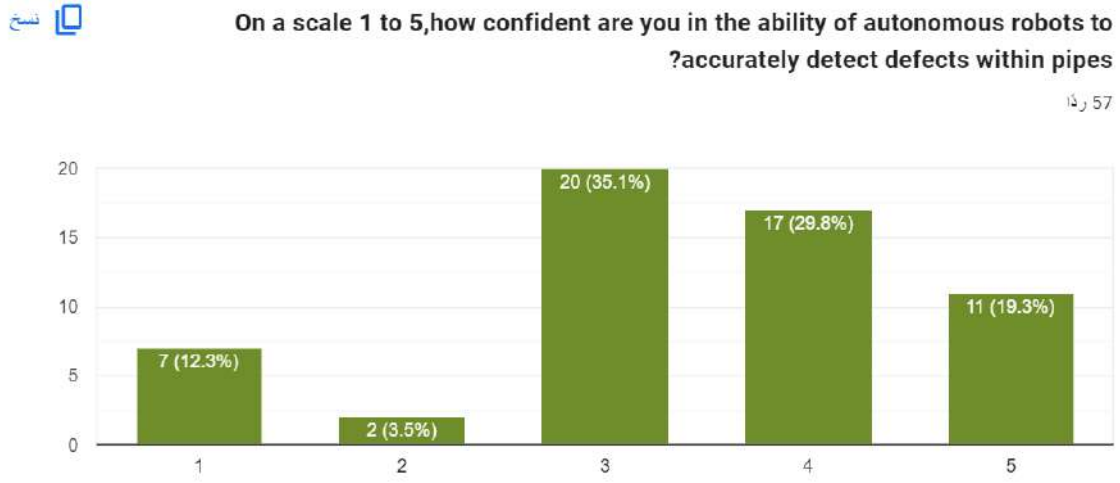


Figure 3.11. Confident of the ability of autonomous robot from survey participants opinion

In Fig. 3.12,a number of people suggested features or capabilities would they like to see or add to the robot or dashboard.

Figure 3.12. Responds for additional features for the robot or the dashboard

In Fig. 3.13 also a number of people suggested features or capabilities would they like to see or add to the robot or dashboard.

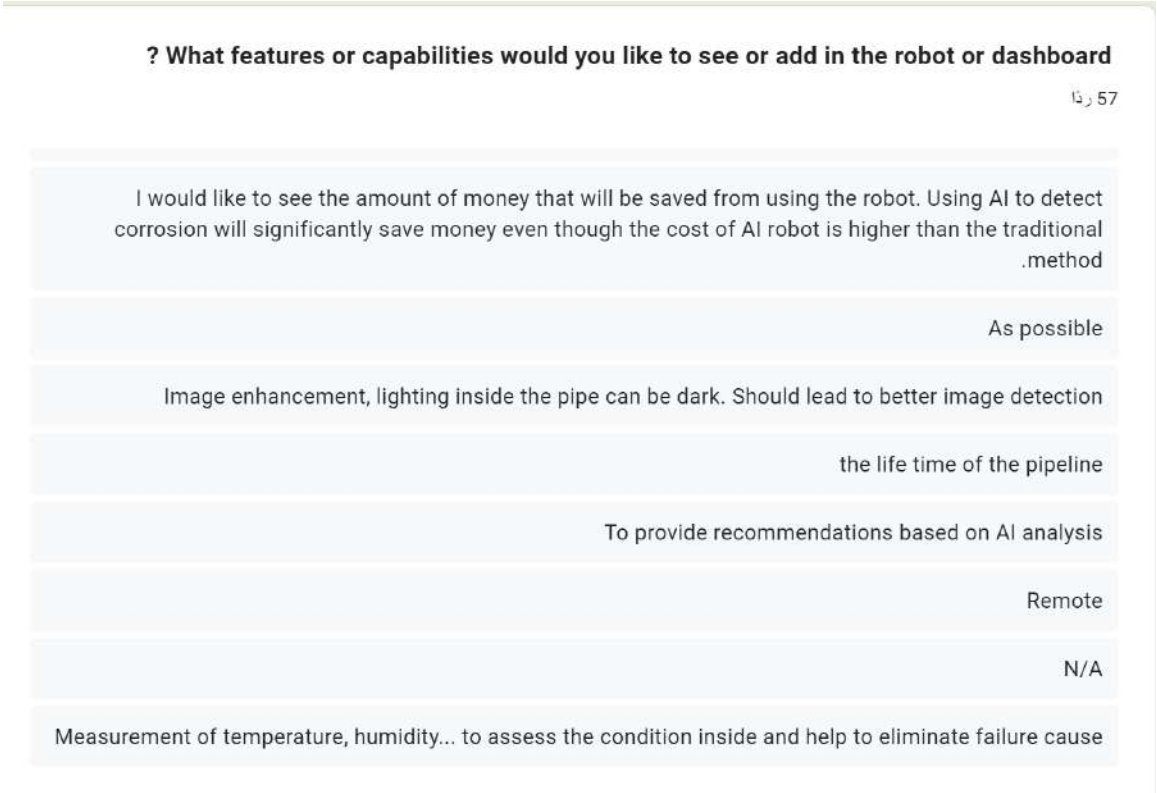


Figure 3.13. Responds for additional features for the robot or the dashboard2

In Fig. 3.14 also a number of people suggested features or capabilities would they like to see or add to the robot or dashboard.

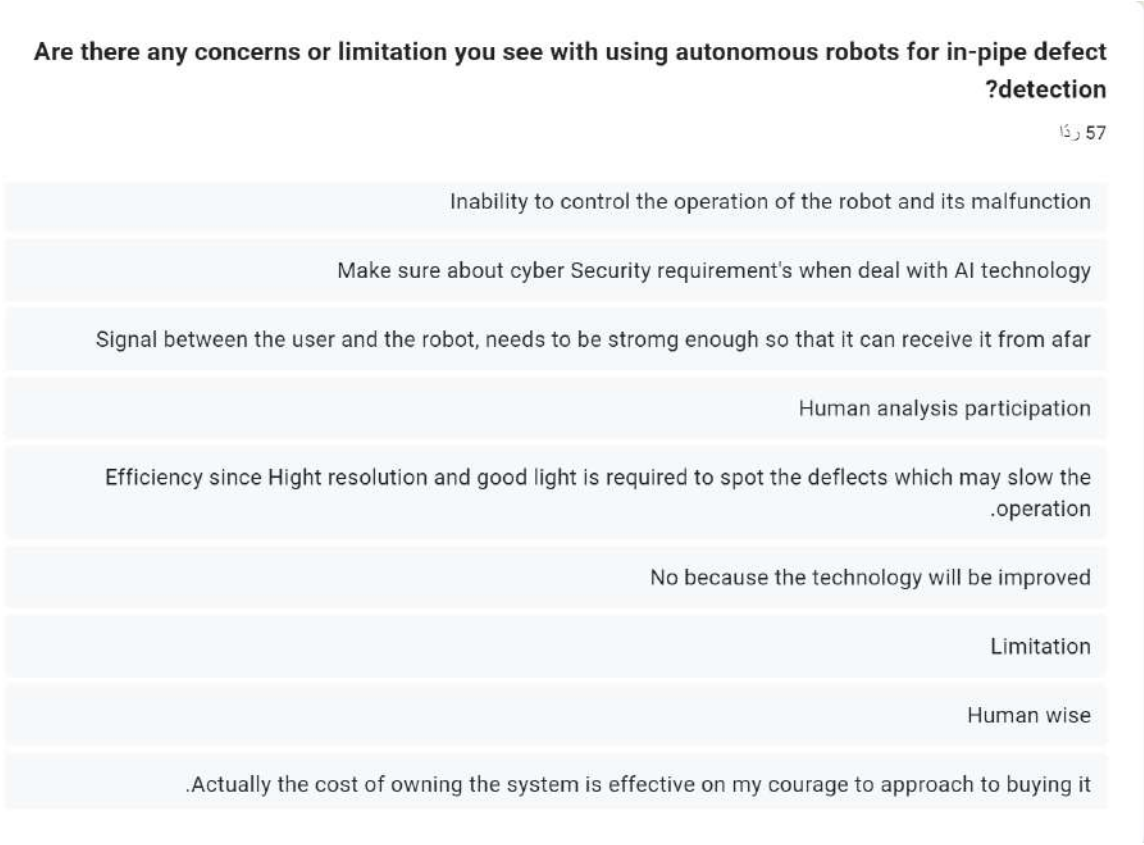


Figure 3.14. Responds for limitations for the robot

In Fig. 3.15, a number of people confess their concerns or limitations with using an autonomous robot for in-pipe defect detection

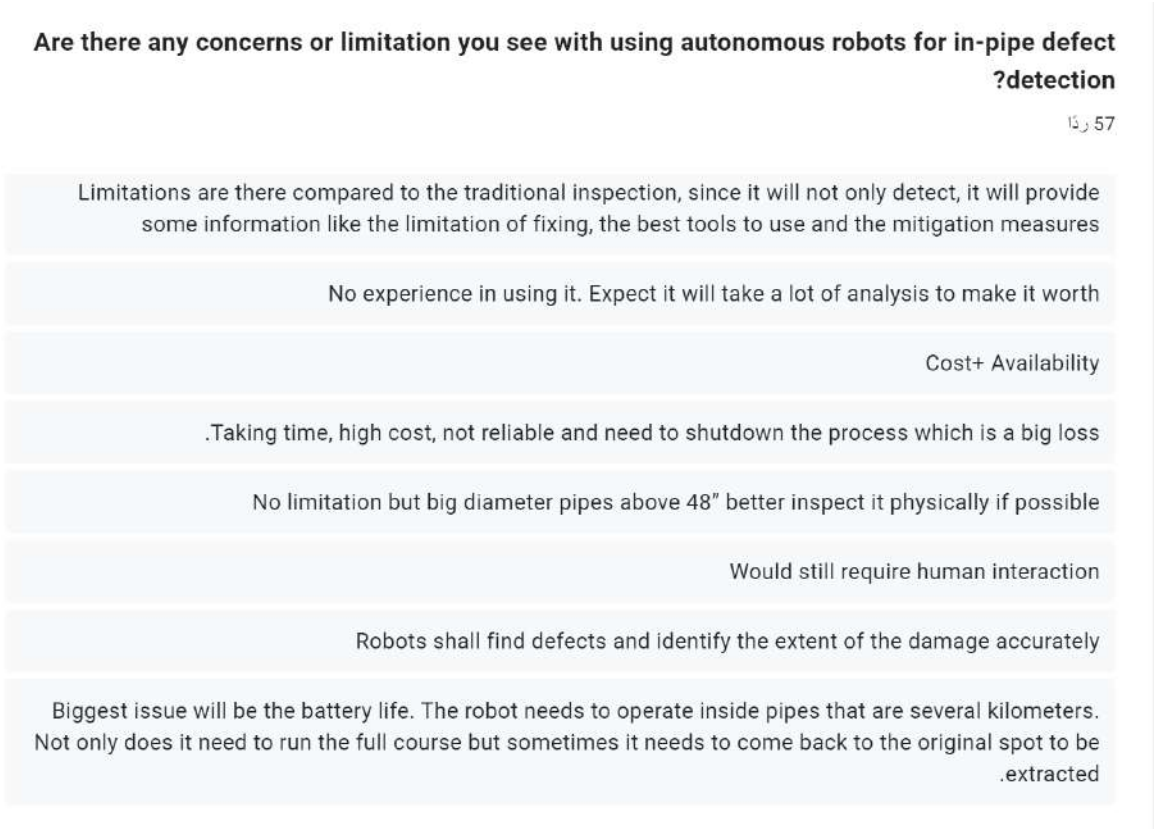


Figure 3.15. Responds for limitation for the robot

3.3 Functional Requirements

- The System :
  - The system shall allow the user and the admin to register.
  - The system shall allow the user and the admin to log in and log out.
  - The system shall be able to take video and process it to detect defects.
  - The system shall be able to classify the identified defects.
  - The system shall be able to store the result and images in the database.
  - The system shall be able to determine the robot's position inside the pipeline using the calculated odometry data.
  - The system shall allow the admin to set the necessary parameters.
  - The system shall be able to determine the robot's next move.
- The Administrator :
  - The admin shall be able to set the pipeline path.
  - The admin shall be able to set velocity parameters.
  - The admin shall be able to set camera settings.
  - The admin shall be able to access the system database.
  - The admin shall be able to view the report.
  - The admin shall be able to view the history of previous results.
- The User :
  - The User shall be able to view the report.
  - The User shall be able to view the history of previous results.
  - The User shall be able to view the notification when a defect is detected.

3.4 Non-Functional Requirements

- **High adaptability:**
  - The device shall be able to operate in the pipeline's harsh environment.
  - The device shall be able to capture the defects in low light conditions.
- **Portability:**
  - The device shall be small and lightweight enough to be easily carried around.
- **Durability:**
  - The device components shall be contained, so that we avoid any component falling inside the pipeline.
- **Affordability:**
  - The device components shall be affordable and available in the market.

3.5 Hardware Requirements

3.5.1 Raspberry pi 4

Figure 3.16. Raspberry Pi 4

The core computational unit of the robot is the Raspberry Pi 4. This miniaturized single-board computer functions as the robot's central processing unit. The Raspberry Pi 4's widespread adoption is attributable to its cost-effectiveness, ease of use, and extensive applicability across various domains, including robotics. It offers a selection of two primary operating systems:

- Raspberry Pi OS(formerly Raspbian): the most extensively utilized, is a Linux distribution based on Debian and optimized for Raspberry Pi's hardware architecture.
- Raspberry Pi Desktop for PC and Mac: unique iteration of Raspberry Pi OS facilitates installation on standard personal computers and Macintosh computers. It delivers a user experience akin to the Raspberry Pi, with the added advantage of seamless execution of native Windows or Mac applications [39].

3.5.2 Encoder

Figure 3.17. Encoder

Encoders play a vital role in robotics by giving critical information on the position, velocity, and orientation of motor-driven devices. They translate motion into electrical signals that the control system use to modify the mobility of the robot [40].

3.5.3 Raspberry Pi Camera

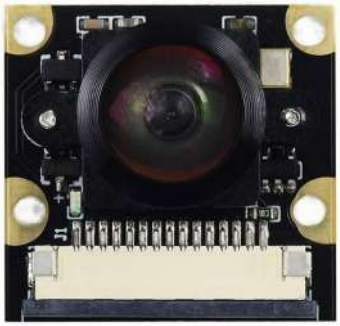


Figure 3.18. OV5647 camera

This camera module utilizes a Fisheye Lens for a 75 Field of View and the OV5647 it Used to produce video inside pipes and 2592 x 1944 images, also compatible with Raspberry Pi 3B+4B. It has IR Fill Light support, with 4 screw holes on the module and a 3.6mm focus length lens that can be adjusted, making it ideal for capturing footage in large areas and perfect for high-quality surveillance of large landscapes [41].

3.5.4 Ultrasonic

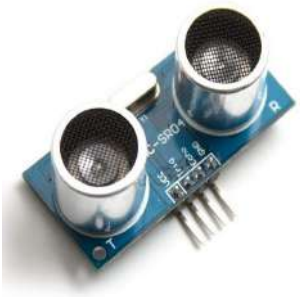


Figure 3.19. Ultrasonic

The control mechanism of the robot uses ultrasonic technology to precisely detect its heading and travel through the pipeline. When sound waves of high frequency are emitted by ultrasonic sensors. Through the analysis of the return time of the sound waves, the robot is able to determine its distance from the pipeline walls and modify its path accordingly. [42].

3.5.5 DC motor

Figure 3.20. DC motor

Robots frequently employ DC motors because of their ease of use and controllability. These motors are flawless for applications demanding precise speed and position control since they run on direct current (DC) power [43].

3.5.6 H-Bridge Motor Driver

Figure 3.21. L298N motor driver

The L298N h-bridge is a motor driver used to control the speed and direction of the DC motor. It can control two DC motors simultaneously [44].

3.5.7 Wires

Figure 3.22. Wires

Wires, made of metal, serve as conduits for electricity and are typically pliable, enhancing their usability. These essential electrical conductors play a crucial role in powering various devices, ranging from computer circuit boards to neighborhood transformers and long-distance power transmission systems. Without wires, electricity would not be accessible to all, underscoring their vital role in modern society. Wires come in different sizes and materials depending on their intended use [45].

3.6 Actors Definition

Actors are the individuals or organizations who can benefit from this project, for instance, the people who will use the system of the robot and manage it. We have two actors in our system :

- **Administrator:** The Admin is the one who is responsible for setting the necessary parameters for the robot, such as uploading a map for the pipe network and updating the speed and velocity of the robot to fit with the pipe surroundings. The Admin can access the database system.
- **End-User/Supervisor:** The supervisor has access to the reports that can be displayed via the dashboard. The supervisor must be able to read and understand the reading on the dashboard.

3.7 Use Case Diagram

A use case diagram is a specific type of Unified Modeling Language (UML) diagram that visually represents how a system can be utilized by users or actors to achieve specific goals or tasks [46]. It provides a high-level overview of the system's functionality by illustrating the various ways users can interact with it [47]. Each use case diagram consists of several key components [48]:

- A rectangular box representing the system boundary
- Actors illustrated as stick figures
- Oval shapes denoting specific tasks or use cases within the system

As illustrated in Fig. 3.23, the use case representation of the Petro system features two primary actors: the Admin, positioned on the left, and the Supervisor, located on the right, as described in Section 3.7. The diagram includes three main use cases: Log In, Set Parameters, and View Data, arranged vertically in the order of execution based on the actors' capabilities.

Three types of relationships are represented in the use case.

- The Association Relationship: It signifies the interaction or communication between an actor and a use case, represented by a solid line [48]. This relationship clarifies the roles of the actors and their capabilities; for instance, only the Admin can set parameters, while both the Admin and Supervisor can log in and view data.
- The Include Relationship: It indicates that the behavior of the included use case is integral to the base use case [48]. This is represented by a dashed line with an open arrow connecting the Set Parameters and View Data use cases to the Log In use case, emphasizing that actors must first log in to perform these functions.
- The Generalization (Inheritance) Relationship: It is represented by a solid line with an unfilled arrowhead, demonstrating inheritance between use cases. In the use case, the Set Parameters use case serves as the parent and is linked to its child use cases: Set Camera Settings, Set Pipeline Path, and Set Velocity Parameters. This indicates that these child use cases inherit the functionalities of the parent use case [49]. Similarly, the View Data use case, also a parent, connects to its child use cases: View Localization Data, View Detected Defect, and View Stored Data.

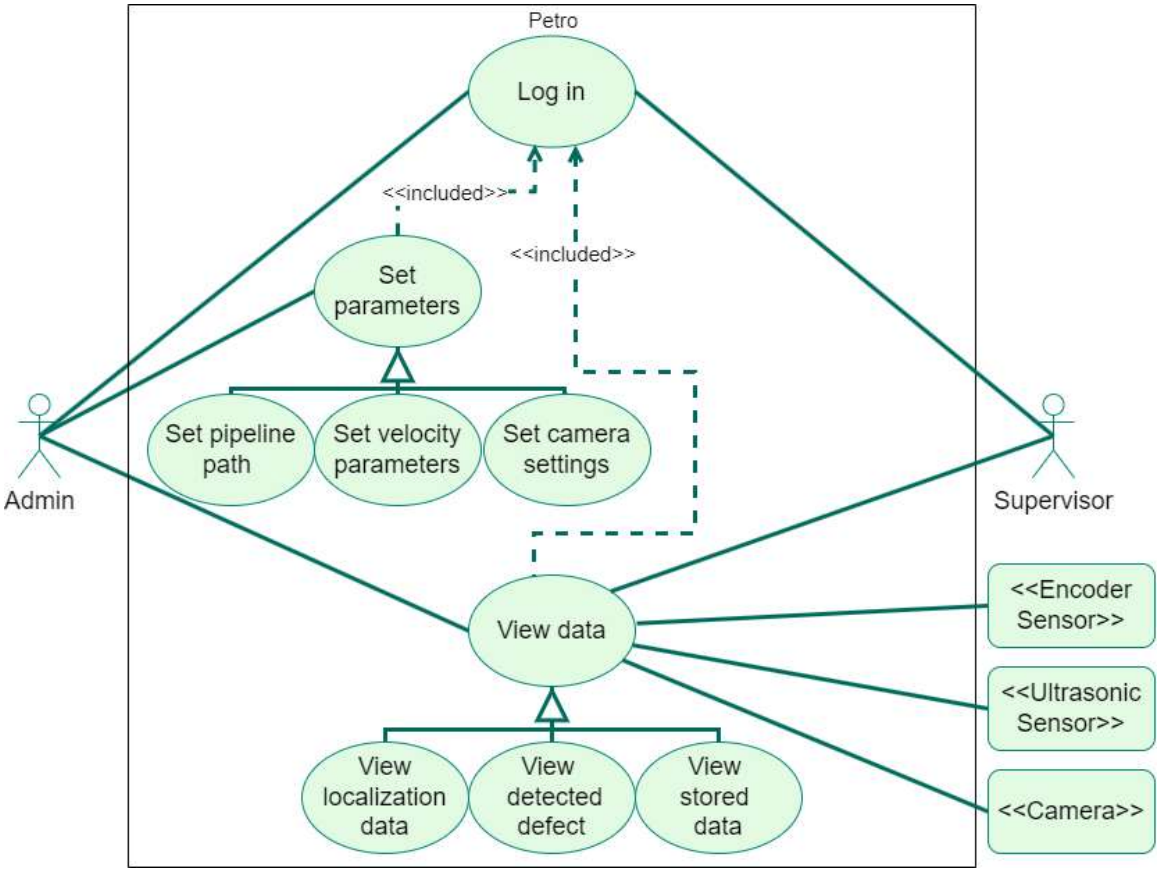


Figure 3.23. Use case diagram

3.8 Activity Diagram

- **System:** An activity diagram is a type of Unified Modeling Language (UML) flowchart that shows the flow from one activity to another in a system or process and depicts the process from the start (the initial state) to the end (the final state). Each activity diagram includes actions, decision nodes, control flows, a start node, and an end node. The system is designed to collect parameters from specified sensors and store them for analysis. If a defect is detected, the system identifies the issue and displays the defect data on a dashboard.

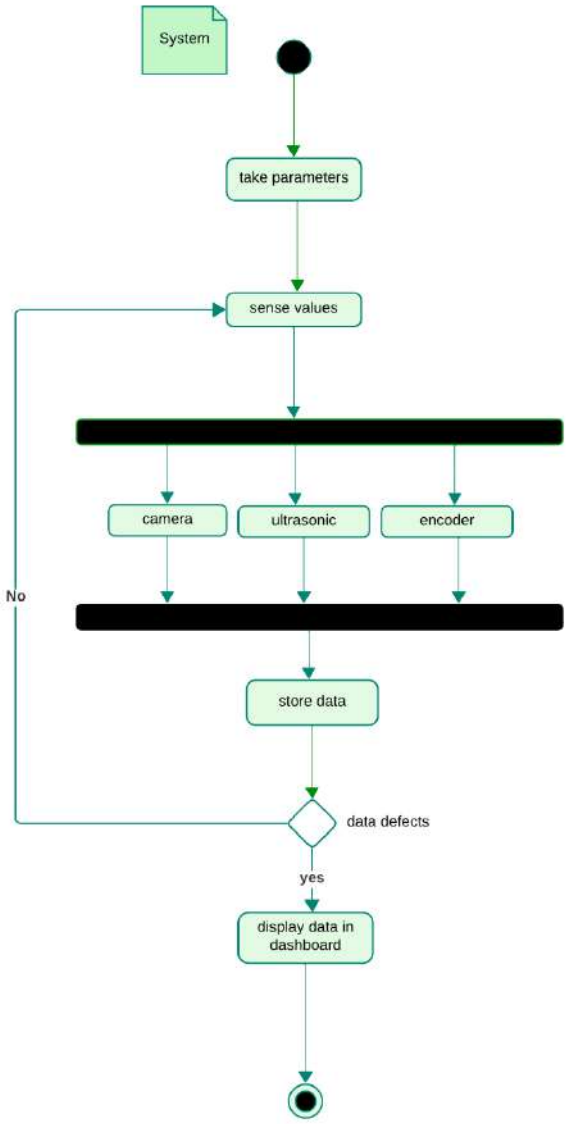


Figure 3.24. System activity diagram

- **Admin:** In the admin's Activity Diagram, the admin begins by attempting to log in to the system. If the admin already has an account, they can proceed to log in; otherwise, they are prompted to create a new, special account with the necessary credentials. Once logged in, the admin has the authority to manage data by setting various parameters, including path, velocity, and video content, into the pipeline for processing. The system then analyzes the data to identify any issues. Upon completing the analysis, it sends a notification to the dashboard. Finally, the system displays a comprehensive report on the dashboard, allowing the admin to review the findings and take necessary actions.

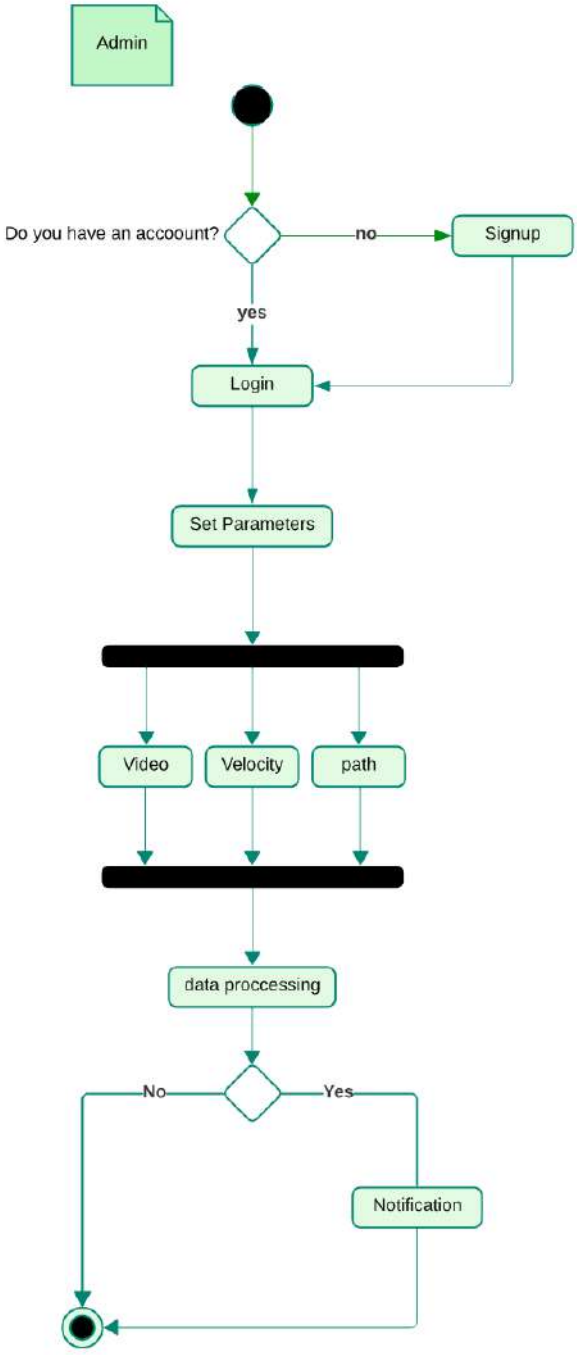


Figure 3.25. Admin activity diagram

- **User:** In the user Activity Diagram, the process begins with the user attempting to log in to the system. If the user already has an account, they can proceed to log in by entering their credentials. If the user does not have an account, they are prompted to create a new account with the necessary information to gain access. Once logged in, the user has the ability to interact with the system's dashboard. They can view the report, which details the type and location of defects. After reviewing the report and ensuring all necessary actions are taken, the user will log out of the system to end the session.

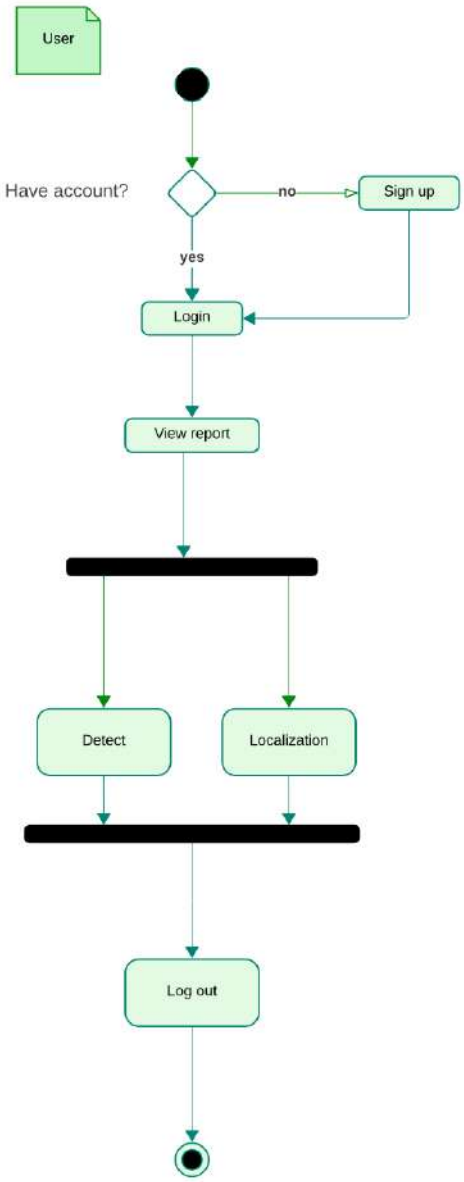


Figure 3.26. User activity diagram

3.9 Conclusion

In conclusion, this chapter provides a structured approach to outlining the project's requirements by gathering both functional and non-functional needs, as well as hardware specifications. By identifying the key actors and their roles, along with illustrating system operations through use case and activity diagrams, the chapter aims to foster a clear understanding of the foundational requirements. This comprehensive analysis will ensure alignment among stakeholders and facilitate informed decision-making, ultimately supporting the successful development and implementation of the system. In the next chapter, we will elaborate more about how the system works.

Chapter 4

Design

4.1 Introduction

This chapter explores the design phase of the project plan as outlined in the waterfall model. This phase forms the basis for how the system will function and how users will interact with it. This chapter provides a detailed overview of the design framework, which includes several key components, starting with Section 4.2, which introduces the class diagram. Class diagrams offer a structured view of the system's design, clearly illustrating the relationships among various entities within the project. Understanding these relationships is essential for a complete grasp of the system's structure, setting the stage for further exploration of the system's architecture. Section 4.3 outlines the system architecture, providing a visual representation of system components, including both hardware and software. This section emphasizes the overall organization of the system and the elements that are part of it, facilitating a clearer understanding of its operational flow. Following this, Section 4.4 includes a sequence diagram that shows how objects interact in specific scenarios within the system, highlighting the dynamic exchanges that take place. Lastly, Section 4.5 addresses the design of interface pages and its elements that contribute to a user-friendly experience. Overall, this chapter aims to give a comprehensive understanding of how each design component works together to create a functional system.

4.2 Class Diagram

A class diagram is a graphical representation of the class entities and their relationship. To represent our system which contains several entities such as :

- Admin: One admin can access the dashboard.
- Supervisor: The dashboard can be viewed by one supervisor.
- Dashboard: Many results will displayed in the dashboard.
- Result: There are 3 types of results in the system, result from a camera, result from the encoder and result from ultrasonic.
- Defects: The camera can detect many defects.

Class diagrams are vital for object-oriented design, clearly describing a system's structure through classes, attributes, methods, and relationships. They improve understanding and communication among stakeholders, ensuring an organized and user-aligned system.

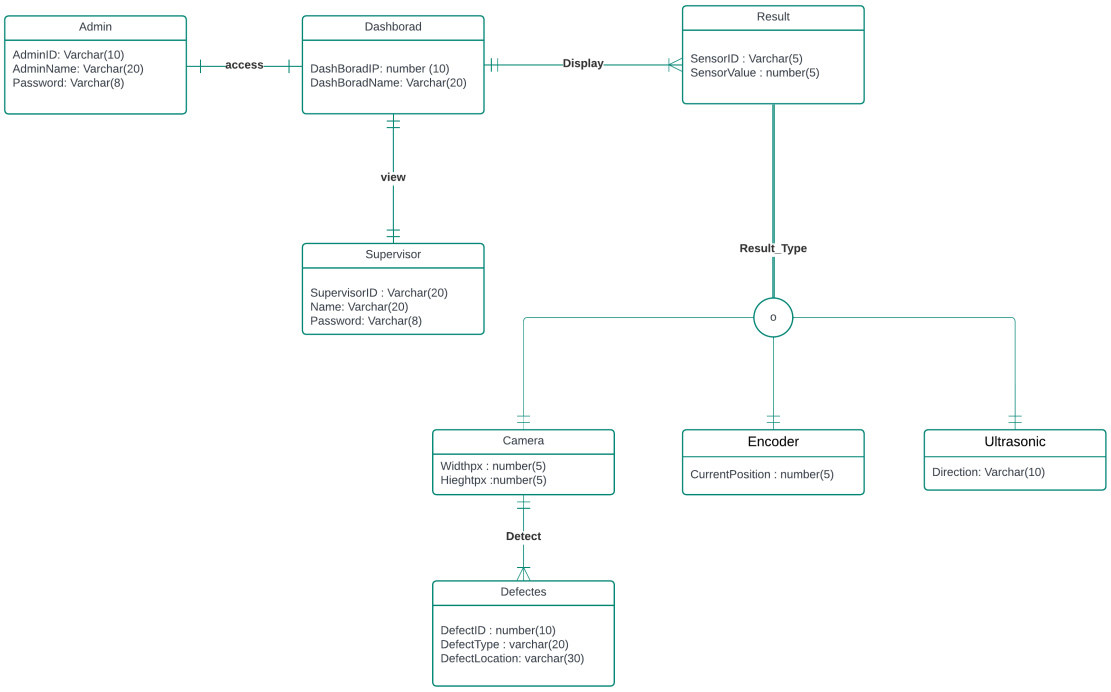


Figure 4.1. Class diagram

4.3 System Architecture

A System Architecture Diagram visually represents the components and relationships within a system. It illustrates how various parts of the system interact with one another, offering a high-level overview of its structure and organization. As depicted in Fig. 4.2, there are two main actors: the admin and the user. This diagram typically includes elements such as robots, sensors, microcontrollers, batteries, actuators, and an external interface dashboard.The system will enable both user and admin registration, ensuring smooth login and logout processes for both roles. It will capture video footage and analyze it to identify and categorize defects, with results and images stored in a database. Moreover, the system will calculate the robot's position within the pipeline using odometry data and permit administrators to configure essential parameters. It will also determine the robot's subsequent actions. Administrators will have access to features such as defining the pipeline route, modifying speed settings, adjusting camera configurations, managing the database, generating reports, and reviewing past results. Users will be able to view reports, access historical data, and receive alerts when defects are identified. In summary, a System Architecture Diagram is an essential tool in software development and systems engineering that improves understanding and communication.

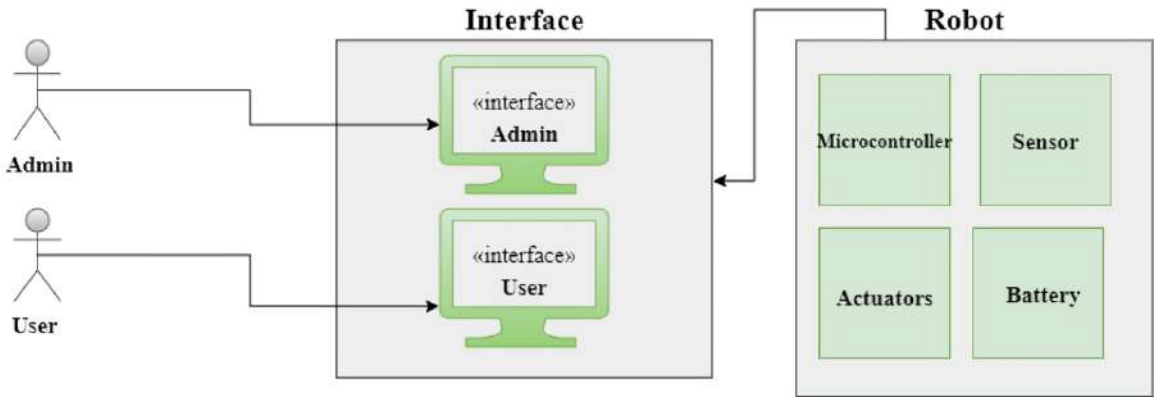


Figure 4.2. System architecture

4.4 Detailed Sequence Diagram

A sequence diagram is a type of interaction diagram in Unified Modeling Language (UML) that illustrates how objects interact in a particular scenario of a system, focusing on the order of messages exchanged over time. As you can see in Fig. 4.3:

- Key Components of a Sequence Diagram:
    1. Actors
        - Admin: Responsible for managing system settings, user access, and overall maintenance.
        - Supervisor: Oversees operations and monitors the performance of the inspection robot.
    2. Hardware Components
        - System: The main control unit that processes requests and interacts with other components.
        - Database: Stores user credentials, system parameters, inspection data, and defect images.
        - Encoder: Converts physical movements of the robot into digital signals for processing.
        - Camera: Captures images and video for inspection purposes.
        - AI: Analyzes data from the camera to identify defects and provide insights.
    3. Massages: Horizontal arrows indicate communication between components, including method calls, responses, or signals.
    4. Return Messages: Dashed arrows indicating the return of control or data to the sender.
- Scenario: User Login Process and Inspection Workflow
    1. Account Verification
        - The Admin or Supervisor initiates the process by requesting to log in.
        - The System checks if the actor has an existing account:
            * **Condition**: If the actor has an account, the System requests login credentials (username and password).

        - **Condition**: If the actor does not have an account, the System requests a sign-up.

2. Admin Login
    - If the Admin has an account, they send a login request to the Login System.
    - The Login System validates the credentials with the Database and returns a success message to the Admin.
    - Upon success, the Admin requests system settings from the Database, which responds with the relevant data.
3. Supervisor Login
    - If the Supervisor has an account, they send a login request to the Login System.
    - The Login System validates the credentials with the Database and returns a success message to the Supervisor.
    - Upon success, the Supervisor requests real-time data from the Inspection System, which responds with the current status of the inspection robot.
4. Saving Credentials
    - After a successful sign-up, the System saves the username and password in the Database.
    - The Database confirms that the credentials have been saved successfully.
5. Setting Parameters
    - The Admin initiates the process to set parameters in the System.
    - The System receives the new parameters from the Admin.
    - The System updates the parameters in the Database.
    - The Database sends a successful feedback message back to the System confirming the update.
    - The System confirms that the parameters have been set successfully and updates the Dashboard accordingly.
6. Inspection Workflow

- The Admin starts the inspection process.
- The Camera is activated to start capturing images for inspection.
- The Camera sends the captured images to the AI for defect detection.
- The AI analyzes the images and identifies any defects.
- The results, including images and defect information, are stored in the Database.

7. Data Request from Encoder
   - The System requests data from the Encoder to gather information about the robot's movements.
   - The Encoder responds with the relevant data and sends it to the Database.
   - The Database stores the encoder data and prepares to send results.
8. Result Delivery
   - The Database sends the combined results (defect analysis and encoder data) back to the System.
   - The System then relays the results to the Admin and Supervisor via the Dashboard for review.

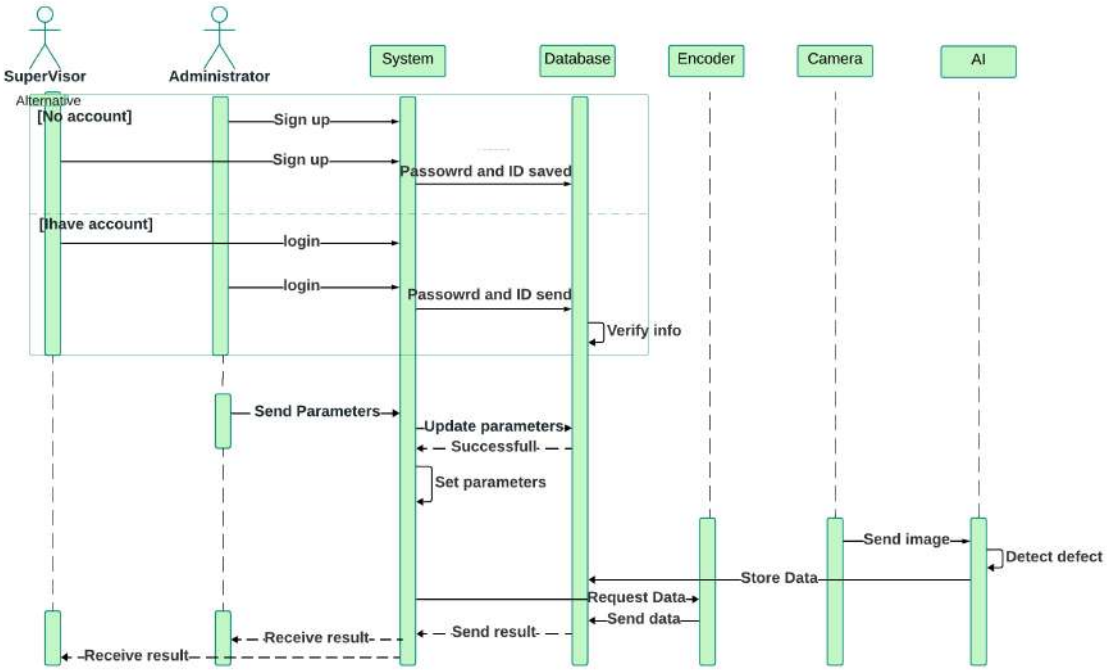


Figure 4.3. Sequence diagram

4.5 Design of Interfaces

This section presents the designed interfaces for the project, showcasing user interaction with the system. The interface design focuses on visual consistency and user-centered design, ensuring that users can easily navigate and utilize the system. There are two types of users: an Admin and a Supervisor. The interface consists of different pages tailored to meet the specific needs of each user, enhancing usability and functionality.
A key interactive feature of the interface is the real-time data update, allowing users to view the most current information available.
The following figures, Fig. 4.4 through Fig. 4.10, illustrate the various pages of the interface, demonstrating how each user interacts with the system to achieve their respective objectives.

4.5.1 Sign Up Page

To initiate the user experience, Fig. 4.4 illustrates the Sign Up page, which allows new users to create an account. It features a welcoming message and fields for essential information such as user role, name, email, password, and user ID, including a "Create Account" button to facilitate the registration process. Additionally, there is a link for users who already have an account to log in.

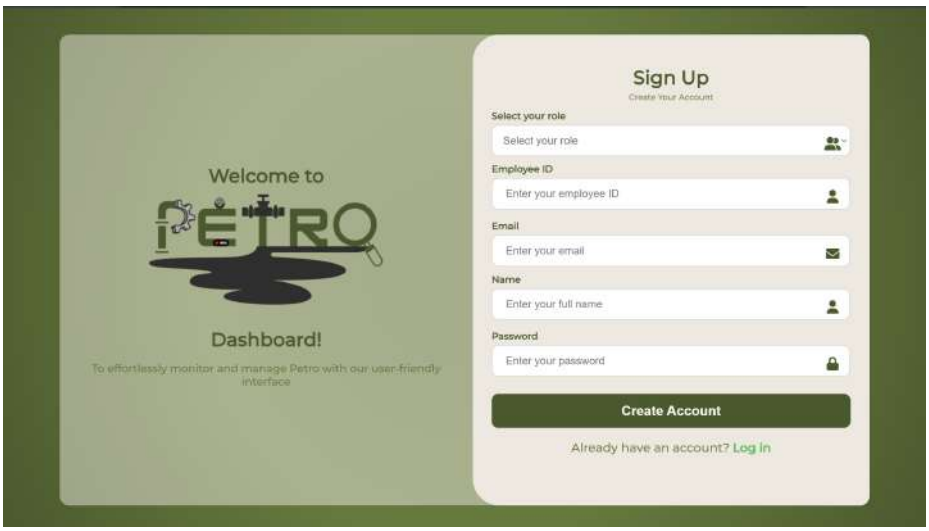


Figure 4.4. Design of the signup page

4.5.2 Log In Page

Fig. 4.5 illustrates the Login Page, which serves as the entry point for users to access the system. This page features a welcoming message and prompts users to enter the Name, ID, and Password they registered with during the sign-up process. The "Access Account" button allows users to login and enter the system. Additionally, a link is provided for users who do not have an account, directing them to the sign-up page for registration.

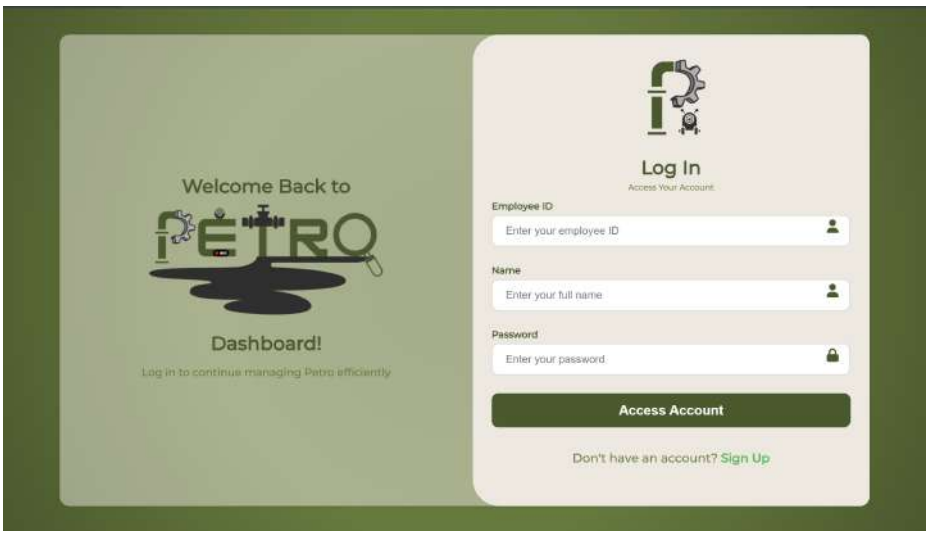


Figure 4.5. Design of the login page

4.5.3 Parameters Page

The Parameters Page is essential for configuring system settings and managing operational parameters, exclusively accessible to the Admin. As shown in Fig. 4.6, this page enables the Admin to set important parameters for the robot, such as determining the pipeline's length and adding the pipe's diameter. The organized layout simplifies access to these features, ensuring the Admin can efficiently manage the robot's operational settings.

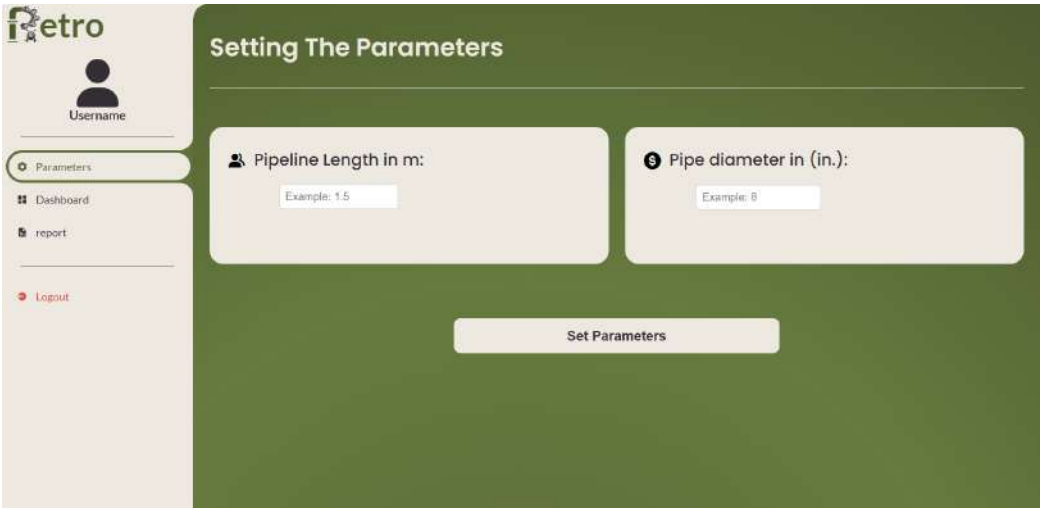


Figure 4.6. Design of the parameters page

4.5.4 Dashboard Page

The dashboard page is designed to provide real-time information. Both the Admin and Supervisor have similar dashboard pages, differentiated primarily by the vertical menu on the left side.

Fig. 4.7 illustrates the Admin dashboard, which displays essential information such as defect type, an image for the defect, and defect location. The menu on the left is used to navigate between pages, including options for Dashboard, Report, and Parameters, allowing the admin to access different sections of the dashboard seamlessly.

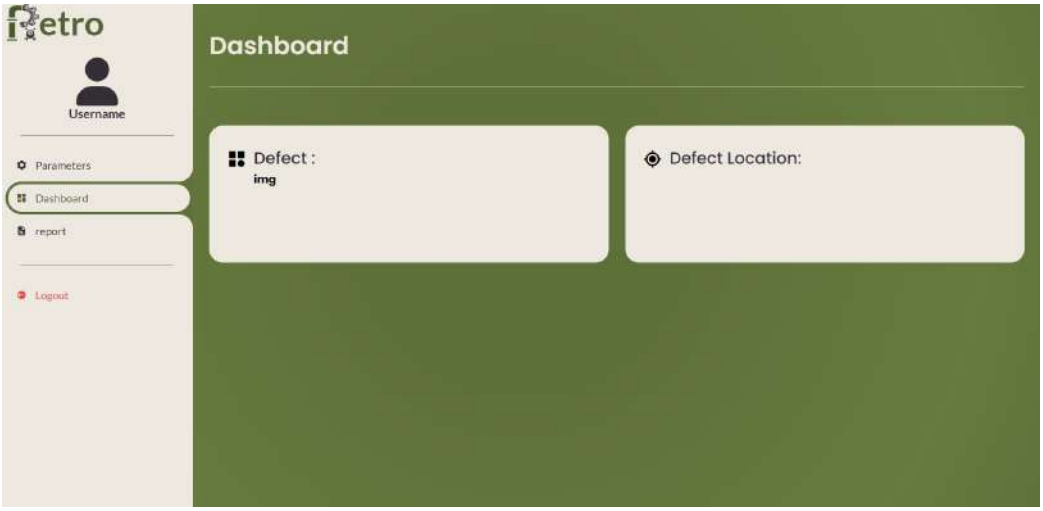


Figure 4.7. Design of the dashboard page

Fig. 4.8 features the Supervisor dashboard, which shares the same layout as the Admin's but includes a menu on the left, consisting only of Dashboard and Report. This design enables the Supervisor to efficiently monitor real-time information without the additional configuration options.

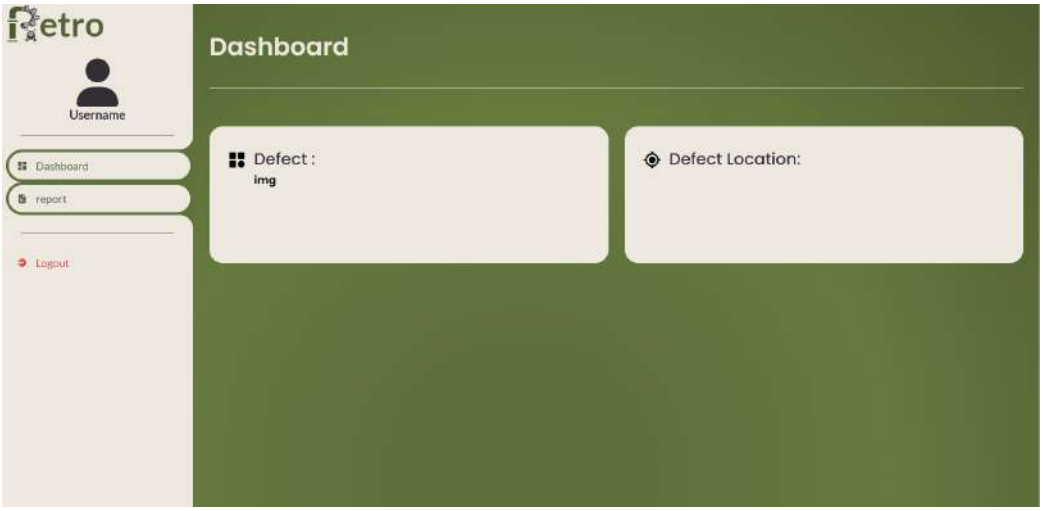


Figure 4.8. Design of the dashboard page

4.5.5 Report Page

The report pages serve a crucial role in the system by providing users with a clear and organized view of defect-related information. Both the Admin and Supervisor have similar report pages, differentiated primarily by the vertical menu on the left side. These pages present information retrieved from the database, including images of defects, defect types, and the locations of these defects within the pipe. This structured format makes it an effective tool for both roles in analyzing defects for future improvements.

Fig. 4.9 illustrates the Report Page for Admin, which includes a dedicated section for images and fields displaying defect type and defect location. The menu on the left allows the admin to navigate between pages, including options for Dashboard, Report, and Parameters, ensuring straightforward access to various functionalities.

Figure 4.9. Design of the report page

Fig. 4.10 showcases the Report Page for Supervisor, which maintains a similar layout to the Admin's page but includes a menu on the left with only Dashboard and Report options. This design facilitates the Supervisor's ability to efficiently review reports for future analysis without the additional configuration settings.

Figure 4.10. Design of the report page

4.6 Conclusion

In summary, this chapter delves into the design phase of the project, as structured by the waterfall model, establishing the foundation for system functionality and user interaction. By presenting key components such as the class diagram, system architecture, sequence diagram, and interface design, the chapter provides a thorough understanding of the system's structure and operational flow. This comprehensive overview not only clarifies the relationships among various entities but also emphasizes the importance of user experience. Ultimately, this chapter lays the groundwork for a cohesive and functional system, ensuring that all design elements work harmoniously together.

Chapter 5

Implementation

5.1 Introduction

In this chapter, we will provide an in-depth discussion of the hardware and software utilized in our project in section 5.2. We will begin by detailing the specific hardware components employed in section 5.3, outlining their characteristics and roles within the system. Following this, we will examine the software used in section 5.4, including any relevant frameworks and applications that contribute to the functionality of the project. Additionally, we will present a block diagram in section 5.5 that visually represents the system architecture, illustrating how various components interact with one another. This will be complemented by a detailed circuit diagram in section 5.6, showcasing the electrical connections and configurations that underpin our design. Furthermore, we will describe the prototype in section 5.7 developed during the project, highlighting its features and how it serves as a tangible representation of our conceptual framework. Finally, we will delve into the implementation details in section 5.8, discussing the processes and methodologies employed to bring the project to fruition. Through this comprehensive examination, we aim to provide a clear understanding of the technological foundation that supports our project and the rationale behind our design choices.

5.2 Details of Hardware and Software Used

5.2.1 Engineering Standards

Our project conforms to rigorous engineering standards encompassing mechanical design, electrical integrity, software development, communication protocols, and safety measures . These standards ensure system reliability, regulatory compliance, and user satisfaction. This document outlines our commitment to engineering excellence as shown in Figure 5.1 & 5.2.

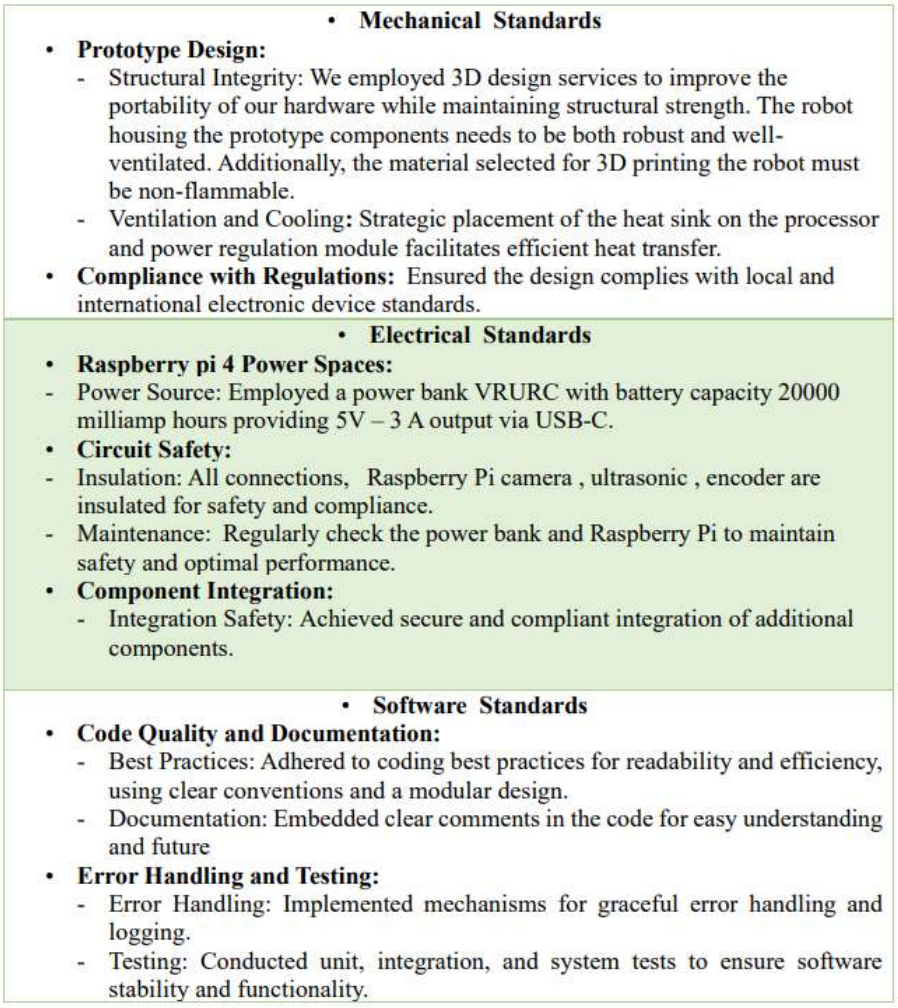


Figure 5.1. Engineering Standards

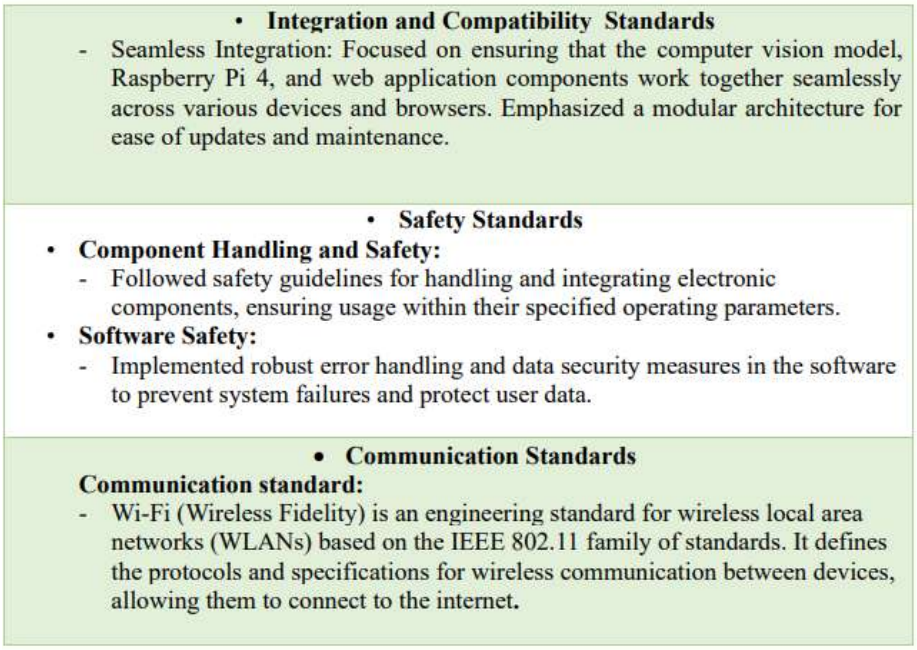


Figure 5.2. Engineering Standard

5.2.2 Constraints

We have different constraints while working along the project:

- Availability dataset: Due to project constraints focused on oil and gas pipelines, we have shifted our attention to prevalent defects in sewer pipes. We systematically cleaned and classified data on these defects, isolating the most pertinent information for our research.To enhance our dataset, we manually documented actual defects in oil and gaz pipes, capturing high-quality photographs as visual references. This integration allows for a more nuanced analysis of real-world
- Economic constraint: to ensure the successful and cost-effective implementation of this project, it is imperative to carefully consider economic constraints.Table V.I presents a comprehensive overview of all components necessary for the construction of the in-pipe inspection robot, emphasizing the critical hardware elements essential for its proper functionality.

TABLE V.I

BILL OF MATERIALS

NO.	Materials	Part Name	Unit Cost	Quantity	Total Cost
1	Raspberry Pi 4	Raspberry Pi 4 Model B	460.00	1	460.00
2	Micro SD Card 128GB class 10	Ultra Microsdxc	41.00	1	41.00
3	HDMI to Micro HDMI	-	23.00	1	23.00
4	Raspberry Pi Camera	OV5647	109.00	1	109.00
5	Heat sink for raspberry pi	-	12.00	1	12.00
6	Type-c cable	-	20	1	20
7	Arduino	Arduino UNO R3	139.00	1	139.00
8	USB cable for Arduino	-	6.00	1	6.00
9	Ultrasonic Sensor	HC-SR04	25.00	3	75.00
10	Optical encoder sensor	HC-020K	14.50	1	14.50
11	H-bridge	L298N	27.00	1	27.00
12	Motor	TT motor	19.00	4	76.00
13	TT motor wheels	-	12.00	4	48.00
15	Lithium-ion battery 3.7v 5000mAh	18650	17.00	2	34.00
16	18650 Battery charger	-	49.00	1	49.00
17	Male-male wires 10cm	-	16.00	2	32.00
18	Male-female wires 10cm	-	16.00	2	32.00
19	Female-female wires 10cm	-	16.00	1	16.00
20	Mini LED spotlight	-	59.88	1	59.88
21	Power Bank	-	129.90	1	129.90
22	Router	E5576	199.00	1	199.00
23	SIM		373.00	1	373.00
24	3D Printing Service (chassis)	-	630.00	1	630.00
			Total	34	2585.28

- Challenging environment: The challenging environment presented significant hurdles for sensor-based image detection, particularly due to variations in luminosity, image quality, and potential obstructions. These factors could significantly impact the accuracy and reliability of the image data collected by the sensors, hindering the overall performance of the system. Fluctuations in ambient light, such as sudden shadows or glare, can drastically alter the brightness and contrast of the images, making it difficult for the system to accurately identify and track objects. Similarly, environmental factors like dust, fog, or rain can obscure the camera's view, degrading image quality and potentially leading to false

detections or missed objects altogether. Additionally, we experienced technical difficulties with the camera, which was displaying a pink hue. To address this issue, we opted to film in a well-lit environment, which helped reduce the color distortion and improve the overall quality of the shoot, mitigating some of the environmental challenges faced by the image detection system.Additionally, We encountered repeated technical challenges with the sensors used in the motion control system, which forced us to replace them with new ones several times during the code rewriting process.

- Time constraint: Time restrictions can have a big impact on the caliber and scope of the work that is delivered. Students frequently struggle to balance their projects with other academic and personal obligations since they have little time to perform research, come up with answers, and finish presentations. The project's overall performance may be impacted by these limitations if they result in hasty decisions, insufficient analysis, or a lack of careful testing and revision.
- Mechanical constraint: The 3D design process for this project presented several challenges, particularly in achieving the necessary accuracy and precision. Designing the chassis and small components, with their intricate shapes and functional requirements, proved particularly difficult. Ensuring the proper fit and functionality of these small components demanded high precision manufacturing techniques. Additionally, limitations in design software and tools hindered the visualization and iteration process, necessitating extensive testing and adjustments to meet the stringent performance and specification requirements. To address this, a key mechanical constraint was to build a fitted-size robot capable of autonomous navigation within the pipeline while effectively avoiding obstacles.
- Communication constraint: Due to communication constraints, we have opted to use a Wi-Fi (Wireless Fidelity) network instead of LoRa technology. The primary reason for this decision is that LoRa is typically sold only to institutions and not available for individual use. Additionally, integrating LoRa with our web platform presents significant challenges, requiring considerable time and resources that are not feasible for our current project. Therefore, Wi-Fi serves as a more practical and accessible solution for our communication needs.

- Regulatory and Compliance Constraints: We encountered limitations regarding the entry of the pipe into the university premises. This issue was addressed by securing a permit that authorized the pipe's access.
- Resource constraint: In any project, the physical environment significantly influences productivity, collaboration, and participant satisfaction. When a project encounters resource limitations due to insufficient workspace, it can result in numerous challenges that obstruct operational efficiency.
- Stakeholder constraint: In any project or organizational effort, effective communication is essential for success. Nevertheless, gaps in communication can arise among stakeholders, resulting in misunderstandings, delays, and diminished project performance.

5.3 Hardware Used

This section provides an overview of the hardware components employed in the design of the in-pipe inspection robot. Each component is integral to fulfilling the project's objectives, such as defect classification, navigation, and data collection. The components are categorized according to their functionality, illustrating how they contribute to the overall system. This structured approach ensures clarity in understanding the role of each hardware element within the robot's architecture, from processing and sensing to movement and data management.

5.3.1 Processing and Control

This section outlines the control and processing units used in the robot. The Raspberry Pi serves as the main processing unit, executing algorithms and utilizing computer vision for real-time object detection from camera data. It also manages data collection and connects to a web-based dashboard for monitoring. The Arduino handles real-time control of actuators and sensors, ensuring precise movement. Together, these components create a robust platform for effective operation of the robot.

5.3.1.1 Raspberry Pi

The Raspberry Pi is a single-board computer that provides a collection of GPIO (general-purpose input/output) pins, enabling control over electronic components. This device was selected as the primary hardware component for the project due to adequate processing power for AI implementation, allowing the system to effectively locate and classify defects in real time. Additionally, the availability of the Raspberry Pi in Saudi Arabia and the extensive resources associated with it make it an ideal choice.
Initially, the Raspberry Pi 5 was considered due to its superior speed. However, the decision was made to opt for the Raspberry Pi 4.The Raspberry Pi 5 requires a 5V/5A power supply, but a suitable portable and lightweight power bank that meets these specifications was difficult to find. Only options providing 5V/3A were available. Therefore, as shown in Table V.II, the Raspberry Pi 4 was selected primarily because it can operate effectively with a lower power supply, making it more practical for this project.

TABLE V.II

COMPARISON OF RASPBERRY PI MODELS

Feature	Raspberry Pi 2	Raspberry Pi 3	Raspberry Pi 4	Raspberry Pi 5
Processors (Quad-core)	ARM Cortex-A7	ARM Cortex-A53	ARM Cortex-A72	Arm Cortex-A76
Clock Speed	900 MHz	1.2 GHz	1.5 GHz	2.4 GHz
Wi-Fi	No	Yes	Yes	Yes
Bluetooth	No	Yes	Yes	Yes
USB Ports	4 × USB 2.0	4 × USB 2.0	2 × USB 2.0 2 × USB 3.0	2 × USB 2.0 2 × USB 3.0
Ethernet Port	Yes	Yes	Yes	Yes
Power Supply	5V/2A	5V/2.5∼3A	5V/3A	5V/5A

Fig.5.3 shows the Raspberry Pi 4 and its ports, providing a visual reference for the specifications.

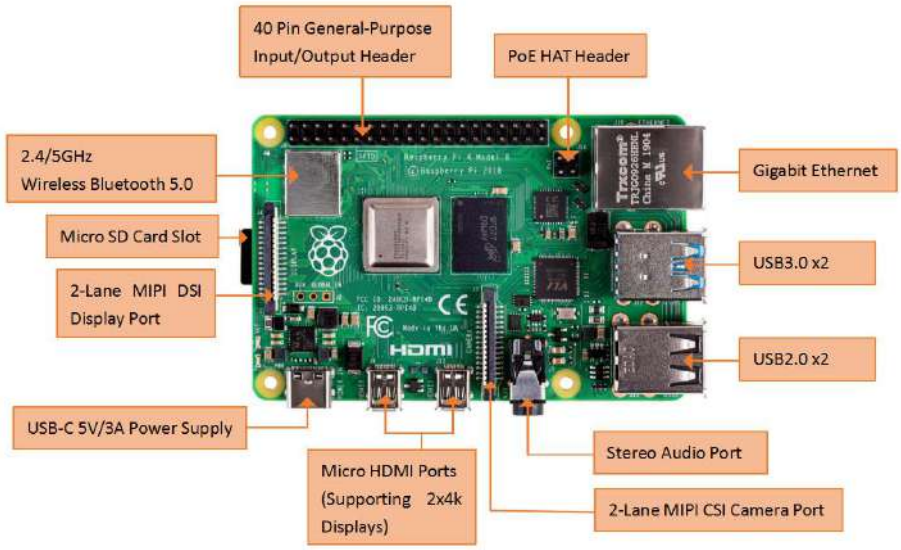


Figure 5.3. Raspberry Pi 4 [2]

Raspberry Pi 4 Additional Specifications:

- Processor: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit System on Chip (SoC) running at 1.5 GHz.
- Memory: 8GB LPDDR4-2400 SDRAM.
- GPIO: Standard 40-pin GPIO header.

- Connectivity: Supports dual-band wireless LAN (2.4 GHz and 5.0 GHz) with IEEE 802.11b/g/n/ac, Bluetooth 5.0, BLE, and Gigabit Ethernet.

5.3.1.2 Arduino

The Arduino Uno is a microcontroller board used in this project to control DC motors and ultrasonic sensors for managing robot movement. This board was selected for its simplicity and the availability of all necessary pins, making it an ideal companion to the Raspberry Pi. Fig. 5.4 below illustrates the pinout of the Arduino UNO.

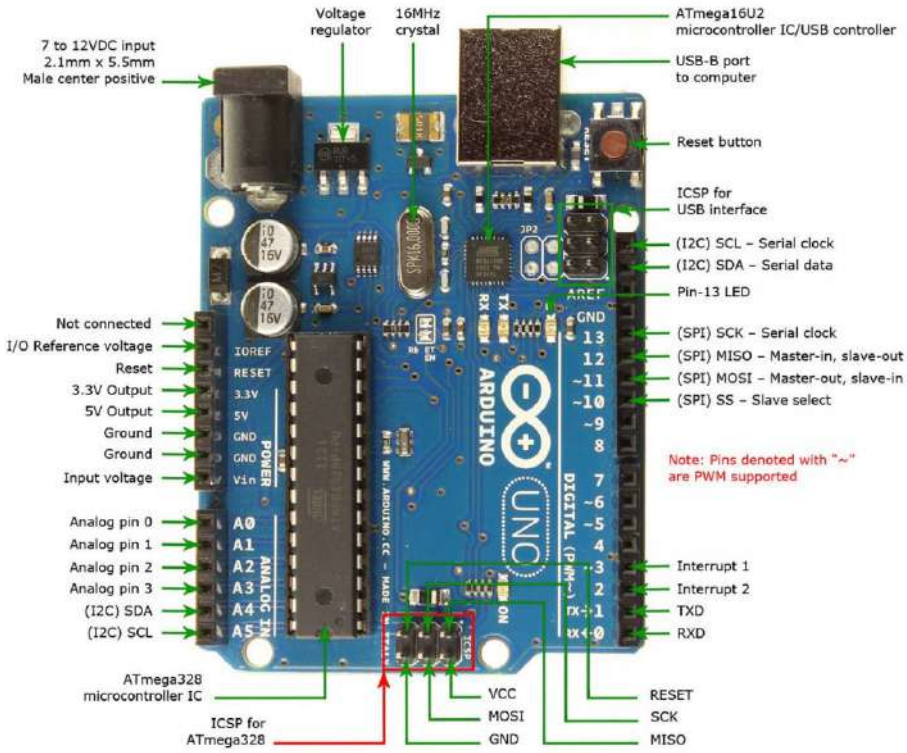


Figure 5.4. Arduino UNO [3]

By integrating the Arduino with the Raspberry Pi, effective coordination of sensors and motors is achieved, enhancing the robot's functionality and responsiveness during operation.

Arduino UNO Specifications:

- Processor: ATMega328P running at 16 MHz.
- Memory: 2 KB SRAM, 32 KB Flash memory, 1 KB EEPROM.
- Pins: 14 digital input/output pins (6 PWM), and 6 analog input pins.
- Input Voltage: 2.7V to 5.5V
- Communication Interfaces: UART, I2C, and SPI.

5.3.2 Inspection Camera

The inspection camera plays a crucial role in defect detection and classification within large oil and gas pipelines. Given the significant diameter of these pipelines, the Raspberry Pi OV5647 Camera, shown in Fig. 5.5, was selected with a wide-angle lens (130 degrees) for its ability to provide the widest field of view (FOV) available among the options. This capability is important for capturing the entire pipe, which is essential for effectively detecting defects.

Figure 5.5. OV5647 camera [4]

Fig. 5.6 shows an image taken by the camera inside a PVC dummy pipe, demonstrating its effectiveness in identifying potential defects and showcasing the wide field of view.

Figure 5.6. Camera view inside a PVC dummy pipe

Camera Specifications [50]:

- Pixel resolution: 5MP (2952×1944).
- Field View Angle:130 degrees.
- Max video resolution: 1080p
- Max frame rate: 30fps
- Connection: Camera Serial Interface (CSI) connector.

5.3.3 Navigation and Localization

Navigation and localization are essential for the effective operation of the robot within underground oil and gas pipelines, where GPS signals are not accessible [51]. This project integrates ultrasonic sensors and encoders to achieve precise positioning and movement control. The combination of these technologies allows the robot to navigate efficiently within the pipeline, ensuring reliable inspections and data collection.

5.3.3.1 Ultrasonic Sensor

Ultrasonic sensors are devices that determine the distance to an object by utilizing sound waves. They function by emitting a sound wave at a specific frequency and then listening for the echo that bounces back. By measuring the time taken for the sound wave to travel to the object and return, the sensor can calculate the distance from itself to the object [5]The formula for calculating distance using ultrasonic sensors is:

$$\text{Distance} = \frac{\text{Speed of Sound} \times \text{Time}}{2}$$

where the speed of sound in air is approximately 343 meters per second (at 20°C), and the time is the duration it takes for the sound wave to travel to the object and back. Fig. 5.7 shows the sensor's pins and the process of emitting and receiving sound waves.

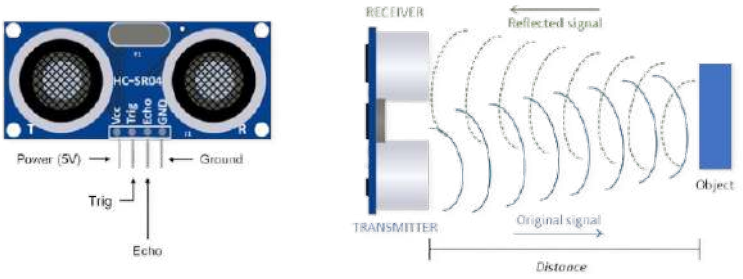


Figure 5.7. Ultrasonic sensor [5]

In this project, ultrasonic sensors play a critical role in navigation by enabling the robot to avoid obstacles while operating in oil and gas pipelines. By employing three ultrasonic sensors, the robot continuously measures the distances to potential obstacles. This real-time data is essential for implementing an obstacle avoidance algorithm, allowing

the robot to dynamically adjust its path and orientation to prevent collisions.
For the navigation system, two waterproof ultrasonic sensors were initially considered: the JSN-SR04T and the A02YYUW. However, the A02YYUW was unavailable, and the JSN-SR04T is heavy for the designed chassis. This led to the selection of the HC-SR04 due to its cost-effectiveness and lightweight design, making it suitable for project needs, especially since three sensors were required. A comparison of the key features of the sensors considered is presented in Table V.III.

TABLE V.III

ULTRASONIC SENSOR COMPARISON

Feature	HC-SR04	JSN-SR04T	A02YYUW
Type	Non-waterproof	Waterproof	Waterproof
Operating Voltage	5V	5V	3.3V-5V
Measuring Range	2 cm to 400 cm	20 cm to 600 cm	3cm m to 450 cm
Accuracy	±0.3 cm	±1 cm	±1 cm
Weight	Lightweight	Heavier	Lightweight
Cost	Lower cost	Higher cost	Higher cost
Availability	Available	Available	Unavailable

HC-SR04 Ultrasonic Sensor Additional Specifications [5]:

- Voltage :+5V DC
- Currnt: 15mA.
- Ranging Distance : 2cm $\sim$ 400 cm.
- Measuring Angle: 30 degree

5.3.3.2 Optical Encoder

In addition to ultrasonic sensors, optical encoders are utilized for localization in this project. They are the most common type of encoders, providing high precision, accuracy, and resolution [52]. Optical encoders operate by using a light emitter and receiver, which require a constant power supply to function. Typically mounted on the rear shafts of motors,

they consist of a code wheel with slits, and a photo sensor, as shown in Fig. 5.8. The primary role of the optical encoders is to determine the robot's position when a defect is detected in the pipeline. By measuring the rotation of the wheels, the encoders provide a positioning data that is crucial for assessing the location of any defects within the pipeline.

Figure 5.8. HC-020K encoder [6]

HC-020K Optical Encoder Specifications [6]:

- Voltage :DC 4.5 $\sim$ 5.5V.
- Conduction tube current: 20 mA
- Output Type: Quadrature output (A and B channels)
- Encoder diameter: 24 mm.
- Encoder resolution: 20 lines (slits).

5.3.4 Movement and Actuation

Movement and actuation are critical components of the robot's functionality, enabling it to navigate and operate effectively within the pipeline environment. This section outlines the key actuators and motor drivers used to facilitate movement and control.

5.3.4.1 DC Motors

The robot employs two DC motors, specifically TT motors, as its main actuators. TT motors are a type of DC motors equipped with a gearbox that decreases the motor's speed while enhancing its torque [53]. As shown in Fig. 5.9 these motors feature a double shaft design, with one shaft driving the robot wheel and the other shaft connected to a code wheel that has slits for the encoder to read.

Figure 5.9. DC TT motor [7]

TT Motor Specifications [7]:

- Voltage : $3 \sim 6$V.
- Current: $80 \sim 150$ mA
- Torque: 0.15Nm $\sim$ 0.60Nm.
- Gear Ratio: 1:48
- Speed: $95 \sim 230$ RPM (depending on voltage).

5.3.4.2 Motor Driver

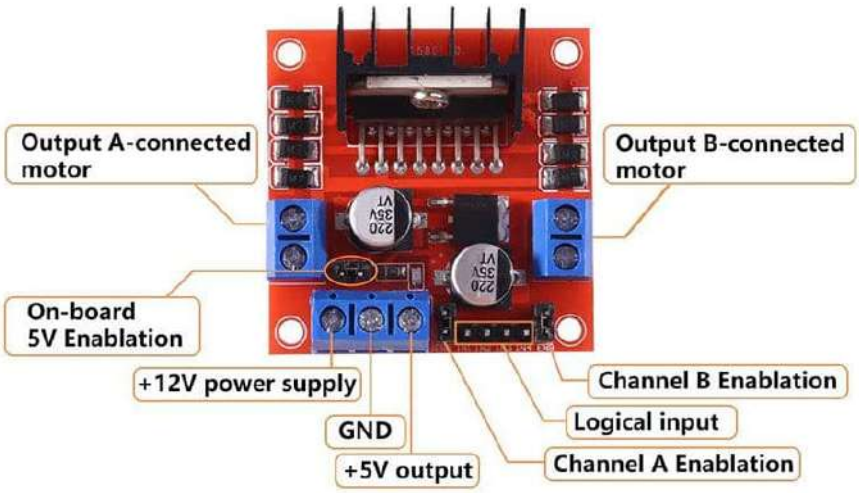


Figure 5.10. L298N H-Bridge [8]

The L298N H-Bridge, in Fig. 5.10 ,serves as the motor driver, managing both the direction and speed of the TT motors. An H-bridge is a versatile electronic circuit designed to control the direction and speed of DC motors [54]. It consists of four switches (commonly transistors or MOSFETs) arranged in an "H" configuration. By controlling these switches in pairs, the circuit allows current to flow through the motor in either direction, enabling bidirectional control [54].

L298N Motor Driver Specifications [8]:

- Voltage : $5 \sim 35$V.
- Rated Current Output per Channel: 2A.
- Peak Current Output per Channel: 3A.
- Max Power: 25W.
- Control Signal Voltage: 5V.

5.4 Software Used

5.4.1 Command-line

Figure 5.11. Command-line

We used the command-line interface (CLI) for dataset augmentation because of its efficiency and flexibility in managing large-scale data transformations. Tools like ImageMagick, OpenCV, and Python scripts allowed us to automate tasks such as resizing, rotating, and adjusting colors. This streamlined approach ensures consistent, reproducible results, making the CLI an essential tool for handling extensive datasets efficiently. Its capabilities are particularly valuable in machine learning and computer vision projects, where rapid and effective data processing is critical.[55].

5.4.2 Draw.io

Figure 5.12. Draw.io

We use Draw.io to design Sequence Diagrams, Block Diagrams, flowcharts, and other illustrations due to its ease of use and versatility. This proprietary tool offers automatic layout options and fully customizable designs, enabling the efficient creation of professional-quality diagrams. Its wide range of shapes and visual elements, combined with drag-and-drop functionality, makes it ideal for effortlessly producing accurate and visually appealing diagrams. [34].

5.4.3 Figma

Figure 5.13. Figma

We use Figma for our web page design because of its powerful cloud-based design and prototyping features. It facilitates real-time collaboration and offers tools like vector editing, design systems, and interactive prototyping, making it an ideal choice for seamless teamwork between designers and developers. As a browser-based platform, Figma provides cross-platform access without requiring local installations, ensuring flexibility and efficiency in the design workflow. Its robust tools and plugins further enhance productivity, making it essential for creating modern, user-friendly web interfaces. [55].

5.4.4 Visual studio code

Figure 5.14. Visual studio code

We used Visual Studio Code (VS Code) for our web page programming due to its versatility and robust functionality. This free, open-source code editor supports Windows, Linux, and macOS, offering features such as debugging, syntax highlighting, code completion, snippets, and Git integration. Its user-friendly interface and extensive customization options through various extensions make it ideal for developers of all skill levels. VS Code's adaptability and powerful tools streamline the programming process, making it a top choice for modern web development. [56].

5.4.5 Xampp

Figure 5.15. Xampp

We used XAMPP for our database setup because of its ease of use and pre-configured environment. This free, open-source software package includes the Apache HTTP Server, MariaDB database, and PHP and Perl script interpreters, making it an all-in-one solution for local web development and testing. XAMPP simplifies the process by eliminating the need for manual server configuration, making it accessible for both beginners and experienced developers. Its cross-platform compatibility further enhances its reliability and versatility for web development projects. [34].

5.4.6 Fritzing

Figure 5.16. Fritzing

We use Fritzing in the design of our circuit because it simplifies the electronics design and prototyping process. This open-source platform offers multiple views, such as breadboard layouts, schematic diagrams, and PCB designs, catering to a variety of project requirements. Fritzing bridges the gap between conceptual ideas and functional prototypes, making circuit design accessible and engaging for both beginners and professionals. Its user-friendly interface encourages creativity and streamlines the development of electronic concepts. [57].

5.4.7 Arduino IDE

Figure 5.17. Arduino IDE

We used the Arduino IDE to program the robot because of its versatility and user-friendly interface. This software enables the writing, editing, and uploading of sketches (programs) to Arduino boards across various operating systems. The Arduino language, based on Processing and similar to C, makes hardware programming accessible for both beginners and experienced developers. Supporting a wide range of Arduino boards, the IDE is an essential tool for prototyping and developing hardware projects, making it ideal for programming our robot. [34].

5.4.8 Imager

Figure 5.18. Imager

We used the Raspberry Pi Imager to prepare SD cards for our Raspberry Pi projects because of its simplicity and efficiency. This official tool supports Windows, macOS, and Linux, offering an intuitive interface to select and install operating systems like Raspberry Pi OS or Ubuntu. It also allows custom .img files to be written for specialized requirements. By streamlining the setup process, the Raspberry Pi Imager makes it quick and accessible for both beginners and advanced users, ensuring a smooth start to our projects.[58].

5.4.9 Blender

Figure 5.19. Blender

We used Blender for our 3D design projects because of its versatility and powerful features. This free, open-source software supports 3D modeling, sculpting, animating, rendering, texturing, rigging, simulation, and video editing. Its adaptability makes it ideal for creating animations, visual effects, games, and architectural designs. Blender's open-source nature allows for extensive customization through plugins and scripts, making it a preferred choice for both professionals and hobbyists in creative 3D projects. [59].

5.4.10 Anaconda

Figure 5.20. Anaconda

We used Anaconda for training machine learning models because it simplifies dependency management through isolated environments and version control, ensuring smooth and conflict-free development. Its conda package manager enables easy installation of precompiled libraries, saving time and resolving compatibility issues. Additionally, Anaconda ensures reproducibility and seamless integration with tools like Jupyter Notebooks, enhancing collaboration and research efficiency [34].

5.4.11 Jupyter

Figure 5.21. Jupyter

We use Jupyter in training models because it allows for interactive coding, real-time execution of model training, and immediate visualization of results. This makes it easier to experiment with different algorithms, tweak parameters, and monitor progress. Additionally, Jupiter's ability to combine code with visualizations and documentation help maintain a clear, reproducible workflow for model development[34].

5.5 Block Diagram

This block diagram, as shown in Fig.5.22, represents a control system managed by a Raspberry Pi and an Arduino microcontroller. The Raspberry Pi connects to a website via WiFi for communication and interfaces with a camera and an optical encoder sensor. The ultrasonic sensors (1, 2, and 3) provide input to the Arduino, which controls a motor driver operating four DC motors. The optical encoder sensor provides feedback on the motor performance of the Raspberry Pi.

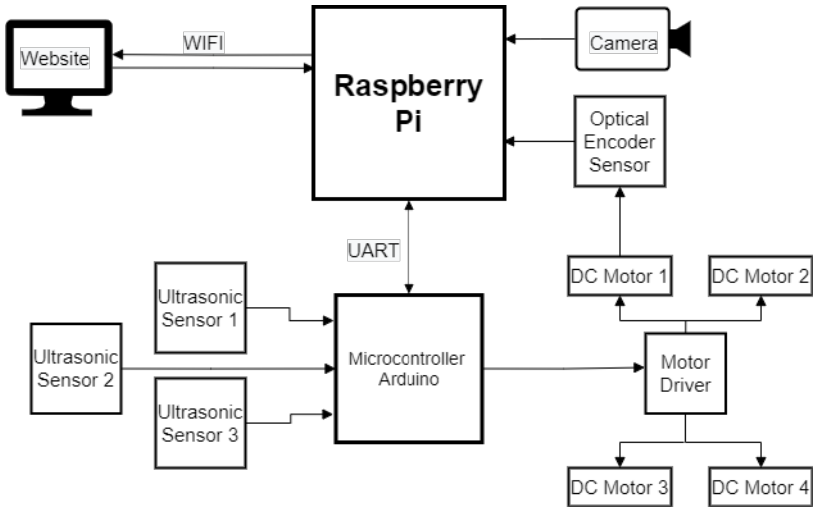


Figure 5.22. Block diagram

5.6 Circuit

This section examines the different circuits essential to the project, each crafted to support specific functionalities. The systems covered include the Camera Inspection System, Serial Communication System, and Robot Motion System. Each circuit is carefully designed to guarantee efficient performance and smooth integration of components. The information provided will encompass connections, components utilized, as well as circuit diagrams,including an overview of the entire system in Fig. 5.23, and real-life examples for better understanding. These circuits constitute the foundation of the implementation, allowing for effective interactions and data processing critical to the project's overall functionality.

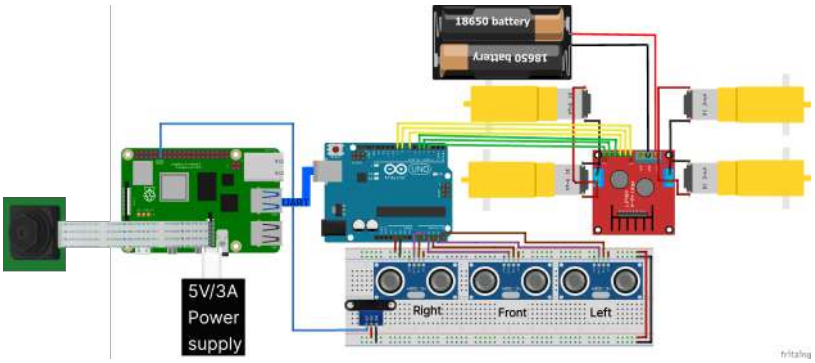


Figure 5.23. Block diagram

5.6.1 Camera Inspection System

The Camera Inspection System is built to work with a Raspberry Pi 4 alongside the Raspberry Pi OV5647 camera module. The camera connects to the Raspberry Pi 4 via a Flexible Flat Cable (FFC), which features multiple flat conductors arranged in parallel and covered with a flexible insulating material. This configuration allows for a compact and lightweight connection, making FFCs particularly suitable for situations where space is constrained.This circuit diagram is illustrated in Fig. 5.24, providing a visual representation of the connections involved.

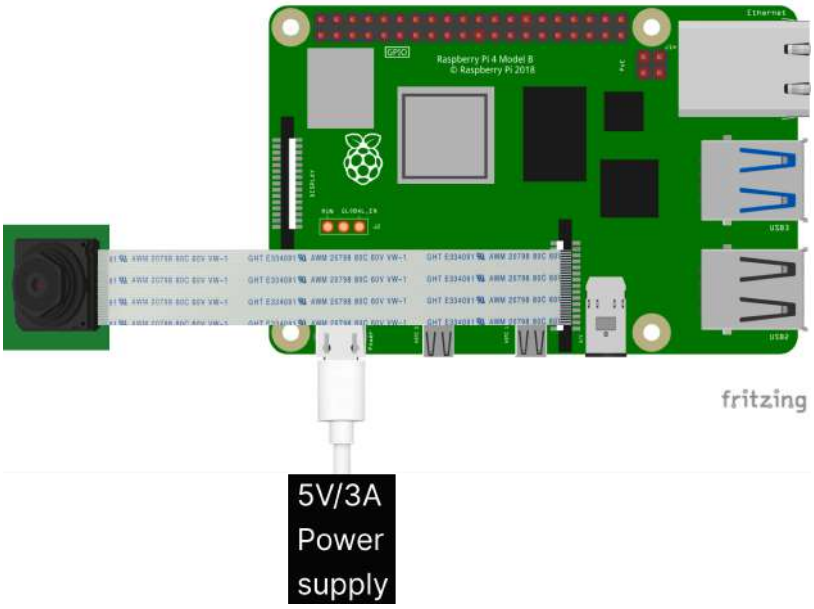


Figure 5.24. Circuit diagram for camera inspection system

Connections

- Raspberry Pi 4 connected to the OV5647 camera module via FFC wire.
- Power bank (5V/3A) connected to the Raspberry Pi.

Furthermore, Fig. 5.25 provides a real-life example of the circuit setup.

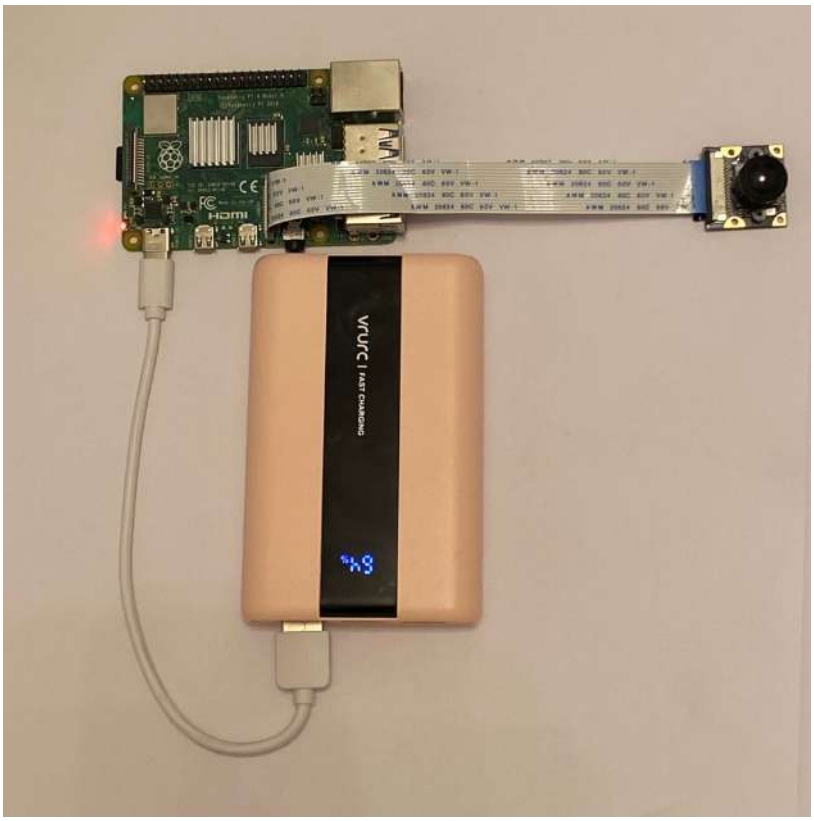


Figure 5.25. Circuit diagram for camera inspection system

5.6.2 Serial Communication System(UART)

The Serial Communication System employs UART (Universal Asynchronous Receiver-Transmitter) to facilitate communication between the Arduino and the Raspberry Pi 4(RPI). This configuration enables data exchange via the USB port of the Raspberry Pi, ensuring smooth integration of both devices. UART is favored for this purpose because of its straightforward implementation with a standard USB cable, which removes the need for extra wiring. It functions without a clock signal,which is a defining feature of asynchronous communication, allowing both devices to operate independently without synchronization. Additionally, UART is compatible with both Arduino and Raspberry Pi 4, requiring no extra components or accessories,as shown in Fig. 5.26.

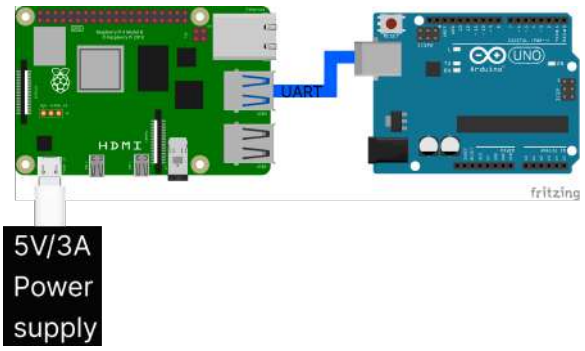


Figure 5.26. Circuit diagram for serial communication between the RPI and Arduino

This setup is illustrated in Fig. 5.27 , providing a real-life example of the circuit connections involved.

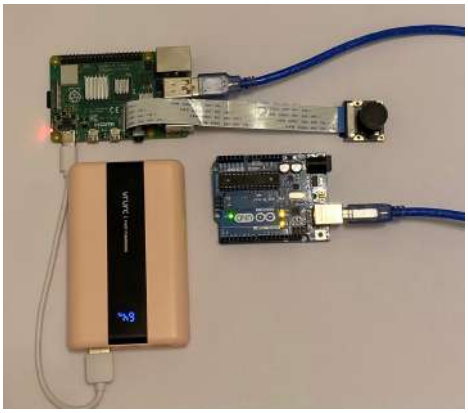


Figure 5.27. Circuit for serial communication between the RPI and Arduino

5.6.3 Robot Motion System

5.6.3.1 Robot Navigation

The circuit layout for the robot motion system includes several components to enable autonomous navigation, as shown in Fig. 5.28. Starting with the Arduino UNO, it is responsible for processing inputs from the ultrasonic sensors and controlling the movement of the robot accordingly. The motor driver module (H-bridge) connects to the Arduino, managing the direction and speed of the four motors to allow effective maneuvering. Three ultrasonic sensors are strategically positioned: one at the front for measuring distance to detect walls or obstacles, and one on each side (left and right) to identify the side walls of a pipe. These sensors connect to the analog pins of the Arduino, allowing the digital pins to be used for the motor driver. They provide real-time feedback to the Arduino, facilitating informed navigation decisions.

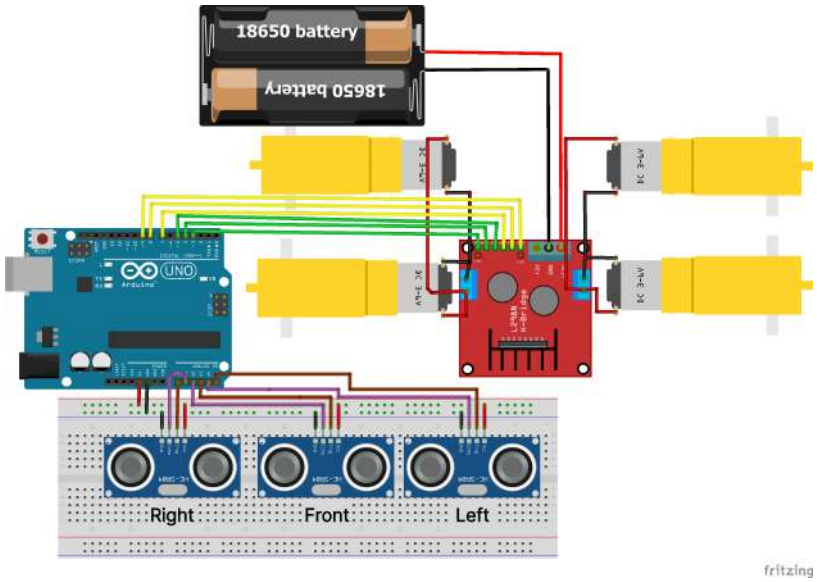


Figure 5.28. Circuit diagram for robot navigation system

Connections

- Power input (12V) and ground of H-bridge connected to the 18650 battery pack.
- Digital pins of the Arduino are connected to the motor driver for motor control signals, with pins 3 and 9 being PWM pins linked to enable pins A and B of the motor driver to control motor speed.

- Analog pins A0 through A5 are connected to the ultrasonic sensors for distance measurement, with the right echo and right trigger connected to A0 and A1, the front echo and front trigger connected to A2 and A3, and the left echo and left trigger connected to A4 and A5, respectively.
- Two motors connected in each H-bridge slot, allowing for control of a total of four motors.
- Common ground connection for all components.
- Power connections made to the Arduino's 5V output.

Fig. 5.29 shows the real-life implementation of the circuit setup.

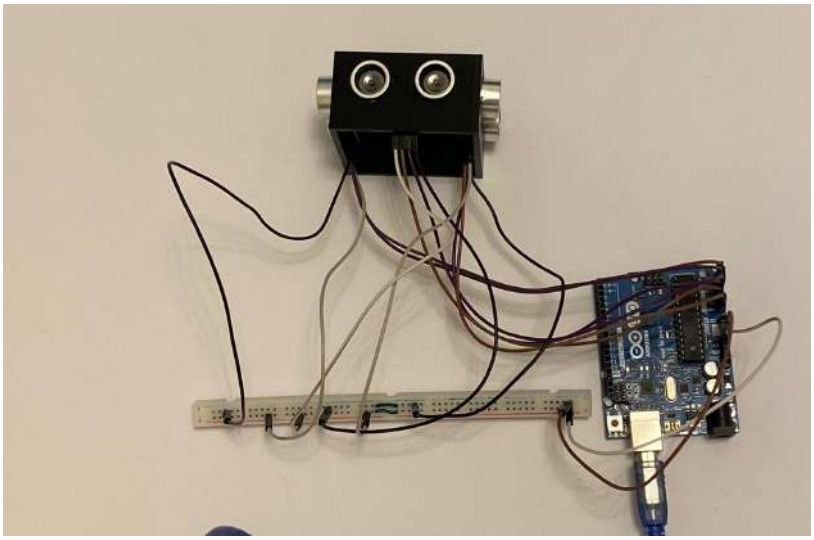


Figure 5.29. Circuit diagram for robot localization system

5.6.3.2 Robot Localization

The robot localization system utilizes an optical encoder connected to a Raspberry Pi to accurately track the distance traveled along a pipe. The Raspberry Pi reads data from the optical encoder, which measures the rotation of the wheels. This setup allows the robot to calculate the distance traveled until it reaches a user-defined target distance, while also enabling the robot to locate defects as they are encountered. Additionally, the circuit includes a power bank that outputs 5V/3A to ensure sufficient power for the Raspberry Pi and connected components,as shown in Fig. 5.30.

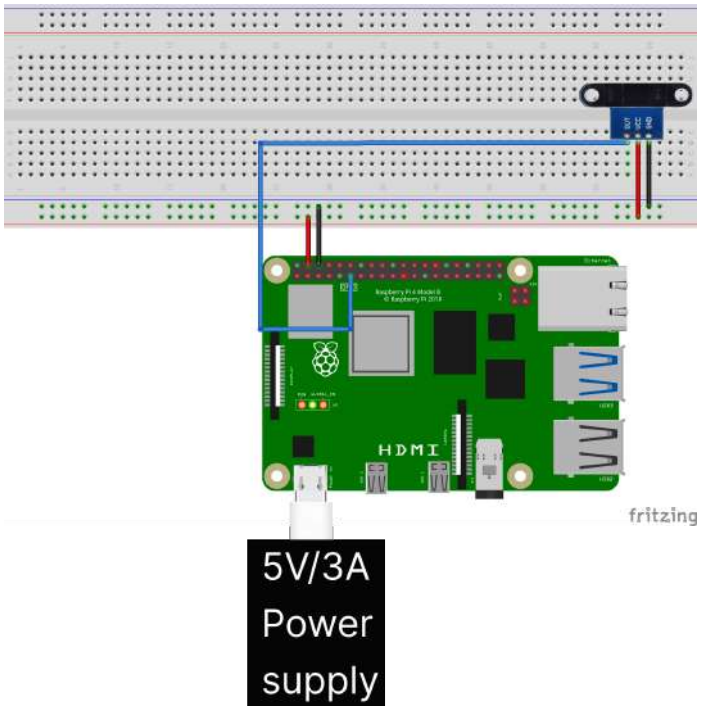


Figure 5.30. Circuit diagram for robot localization system

Connections

- Power bank (5V/3A) connected to the Raspberry Pi.
- VCC of the optical encoder connected to the 5V pin on the Raspberry Pi.
- GND of the optical encoder connected to a ground pin on the Raspberry Pi.
- Signal output of the optical encoder connected to GPIO pin 17 on the Raspberry Pi.

The circuit setup is shown in Fig. 5.31, showcasing a real-life example of the connections used.

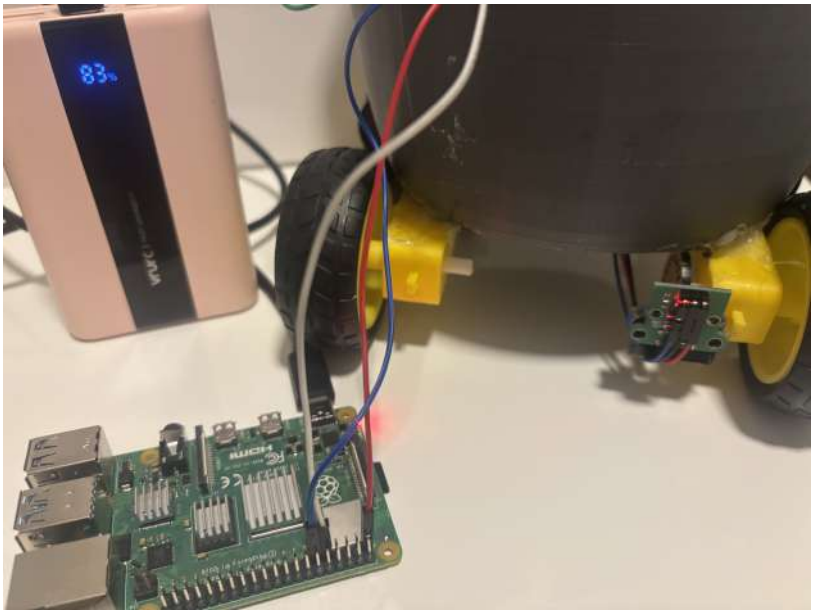


Figure 5.31. Circuit for robot localization system

5.7 Prototype

5.7.1 3D Design

Designing the robot presented significant challenges due to the harsh environment of the pipeline. Many available chassis or robot bodies did not provide adequate protection for the internal components. After extensive discussions,We developed a design, as shown in Fig. 5.32, featuring a spherical body shape that enhances smooth and easy movement within the pipe.

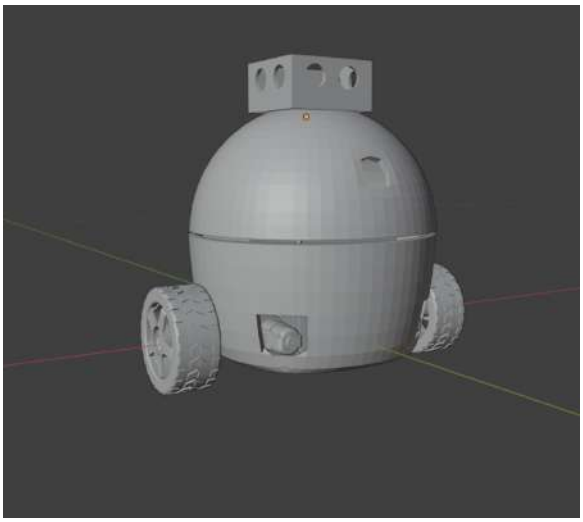


Figure 5.32. The Full Body of The Robot Design

Additionally, Fig. 5.33 shows the 3D printed design implemented in real life, with some adjustments of an additional 2 wheels for balancing.

Figure 5.33. Printed Full Body

To create the 3D model of the robot, we used Blender, a software that offers a comprehensive range of features for design. The initial step in our process involved measuring all the components to determine the optimal dimensions needed to fit everything securely inside the robot. This careful planning ensured that all elements would function effectively within the confined space of the pipeline.

In designing the lower body of the robot, shown in Fig. 5.34, we considered several important factors. We incorporated openings for the shaft extensions connected to the wheels on both sides, along with additional holes for the motor tips to ensure they are centrally positioned. Furthermore, small connectors were included to facilitate easy assembly between the lower and upper bodies

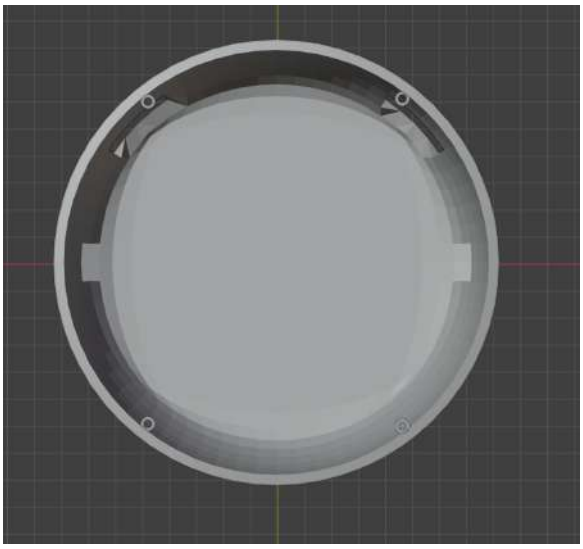


Figure 5.34. The Lower Body of The Robot

Fig. 5.35 shows the complete 3D printed design fresh out of the printing machine, highlighting the practical implementation of our design choices.

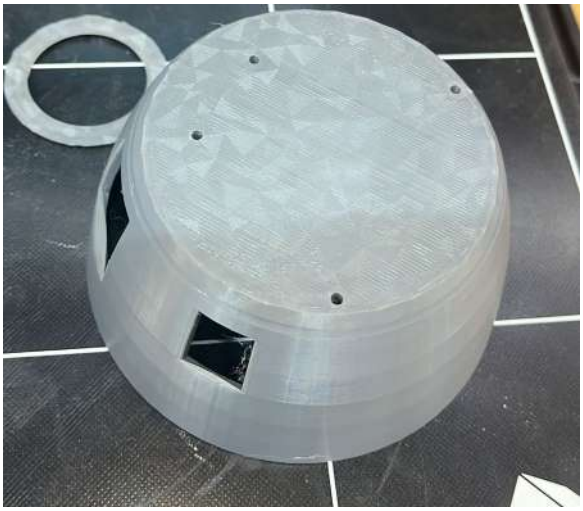


Figure 5.35. Printed lower body

The design of the upper body, shown in Fig.5.36, took into account the placement of the camera and the wire inputs from the ultrasonic sensor holders positioned above the robot. Additionally, we incorporated small connectors to ensure seamless and efficient assembly of the lower and upper bodies.

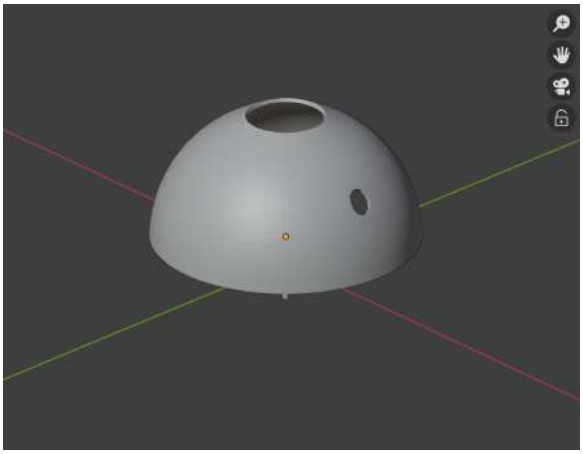


Figure 5.36. The Upper Body of The Robot

Fig.5.37 illustrates the complete 3D printed model fresh out of the printing machine, demonstrating the practical implementation of our design choices.

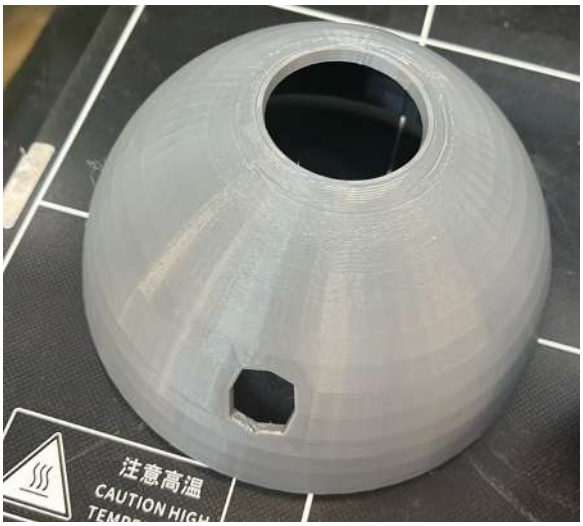


Figure 5.37. Printed Upper Body

In developing the robot's base, our primary goal was to ensure the motors were stabilized for optimal efficiency. This design also aims to support the encoders effectively, enhancing the overall performance of the motor movements.

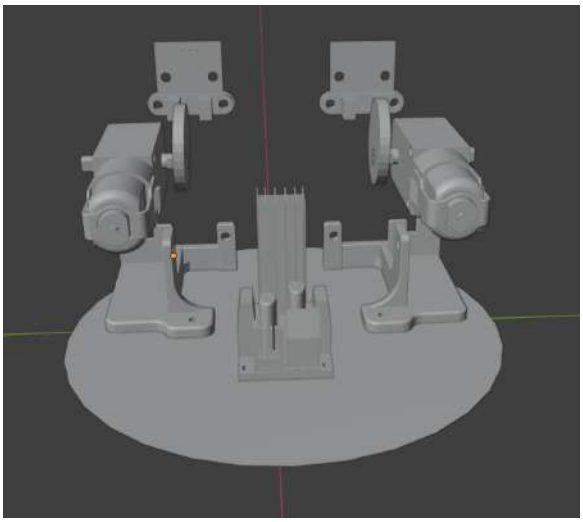


Figure 5.38. The Base Inside The Body

Finally, the ultrasonic sensor holder, shown in Fig.5.39, is positioned on top of the robot. This holder includes ports on the front, right, and left sides to effectively measure distances from three different directions.

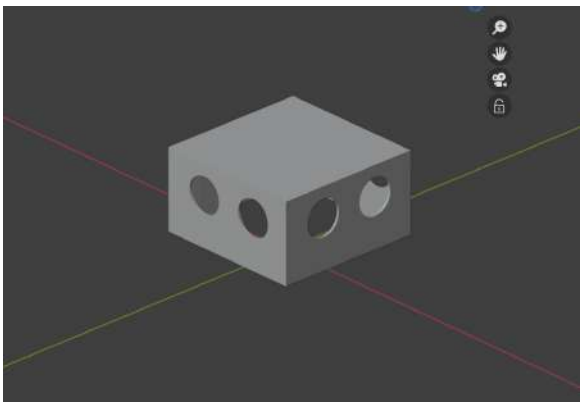


Figure 5.39. The Ultrasonic Sensor Holder

Additionally, Fig.5.40 illustrates the complete 3D printed model fresh out of the printing machine, showcasing the practical application of the design.

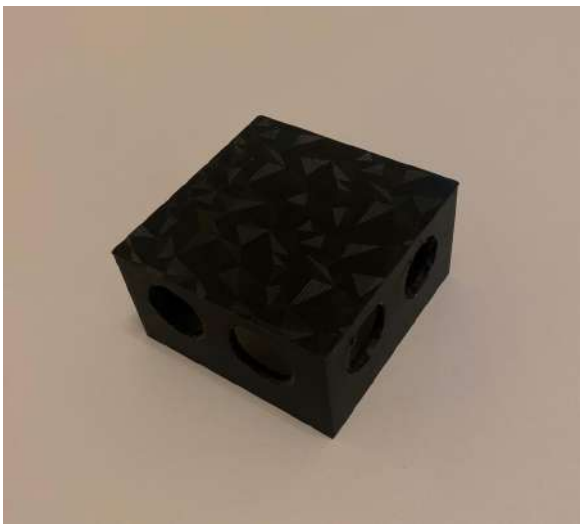


Figure 5.40. Printed ultrasonic sensor holder

5.8 Implementation details

5.8.1 Flowchart

The flowchart shown in Fig.5.41 illustrates the workflow of a robot navigating within an oil and gas pipeline. The process begins with setting the robot in motion, during which it evaluates its orientation. If the path is clear, the robot continues moving forward. If a wall is detected ahead, it rotates to find a new direction. After moving, the robot captures an image, processes it, and utilizes AI classifiers to multi-label (obstacle, crack, holes). It then detects any defects, sends the location of the defects, and stores the analysis results in a database for future reference. Finally, the results are displayed on a dashboard for monitoring. The process repeats as needed until the task is completed.

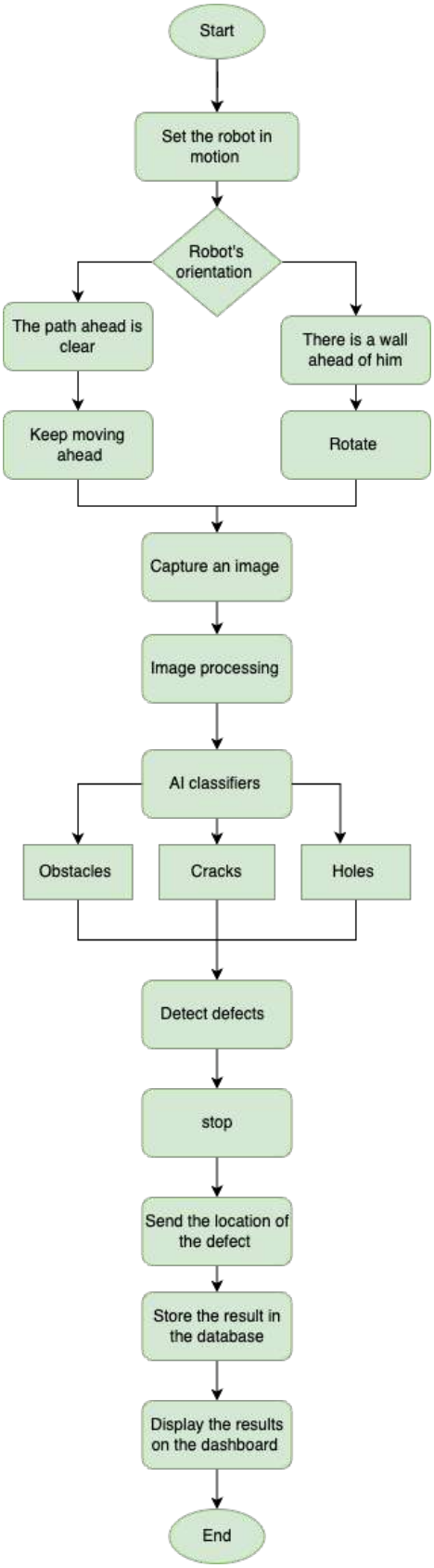


Figure 5.41. Flowchart of the system

5.8.2 Dataset Acquisition

The availability of datasets focused on oil and gas pipeline defects is quite limited. No comprehensive dataset fully encompasses the wide range of surface imperfections in this industry. This lack of resources poses significant challenges for defect detection and model training.

To address this limitation, we initially utilized multiple data sources to compile a more representative sample of defects. By selecting and aggregating images from various datasets, we aimed to ensure coverage of the most common defects associated with oil and gas pipelines. While this approach provided some diversity, it soon became clear that assembling a dataset from pre-existing sources was not an ideal solution. The variability in image quality, context, and environmental conditions within these datasets introduced inconsistencies that diminished their effectiveness for our purposes.

Recognizing the unique internal environment of oil and gas pipelines and the need for highly accurate detection, we created our own dataset by photographing defects directly as shown in Fig.5.42. This approach enabled us to capture high-quality, standardized images, ensuring consistency in lighting, resolution, and defect representation. Importantly, this allowed us to focus on capturing the most prevalent and relevant defects commonly found in oil and gas pipelines, such as holes, cracks, and obstacles. The creation of a custom dataset significantly enhanced the quality and reliability of our training data. By tailoring the dataset to the specific requirements of our detection system, we were able to provide consistent and representative samples that closely resemble real-world scenarios. This improved the performance and precision of our chosen detection model, YOLOv9, which is known for its speed and accuracy in object detection tasks.

Figure 5.42. New dataset

The Raspberry Pi 4 camera, specifically the OV5647 model, was utilized to photograph various types of defects commonly found inside oil and gas pipelines, such as obstacles cracks, and holes. The process began by recording 10-second video clips with the camera. Each video was planned to capture clear and detailed footage of defects from multiple angles. After filming more than 10 video clips As shown in Fig.5.43, these were systematically converted into high-quality images. This method provided a robust foundation for creating a comprehensive and reliable dataset for defect detection.

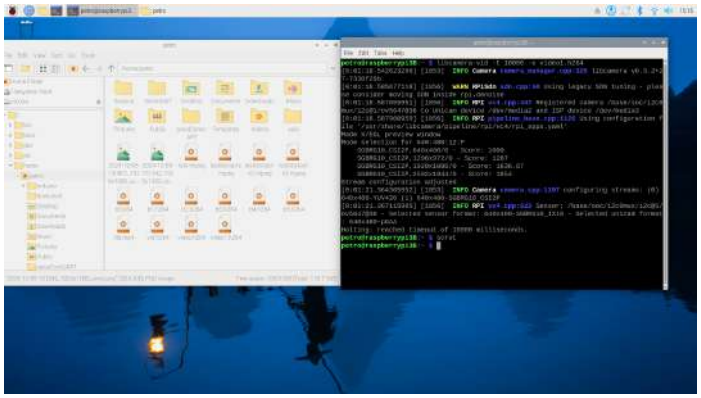


Figure 5.43. code for the image acquisition through the camera

5.8.3 Data preparation

5.8.3.1 Data Annotation

Image labeling is annotating specific objects or features within an image. This step is crucial for teaching computer vision models to recognize and differentiate between various objects by associating visual information with labels. By training models on labeled images, they can later identify and classify objects in new, unlabeled images.
Labeling can be performed using a variety of annotation tools that allow for precise boundary creation around objects of interest. These boundaries, known as "bounding boxes" are drawn around the object, and each box is given a label corresponding to the object's identity. This technique helps the model distinguish between different types of objects within an image, improving its accuracy in object detection and classification tasks.

In our project, we utilized Roboflow for defect identification in oil and gas pipelines. Roboflow enabled us to automatically generate bounding boxes around the defects present in the pipes as shown in Fig.5.44, Each box was labeled with a specific type of defect, such as cracks, holes, or obstacles, allowing the model to learn how to detect and classify these defects.

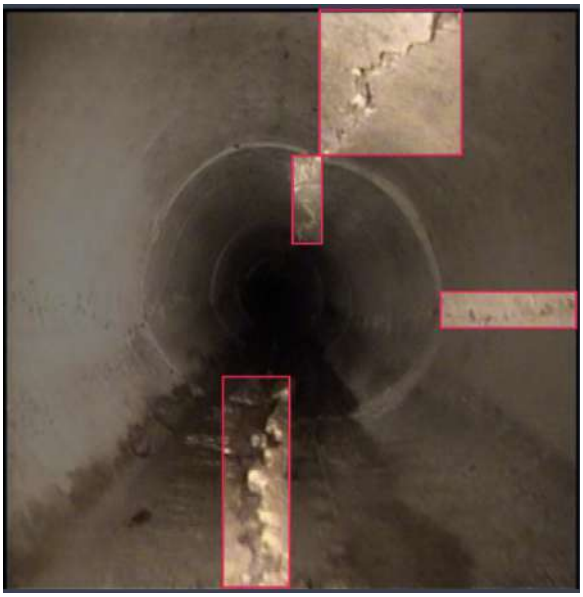


Figure 5.44. Image labeling

5.8.3.2 Data Augmentation

In the model training stage, both the quality and quantity of collected data are essential for improving the model's recognition performance. However, a limited dataset can hinder the model's ability to generalize and accurately detect defects. To overcome this, various data augmentation techniques can be applied, including multi-angle image rotation, saturation adjustment, image flipping, adding salt-and-pepper noise, color dithering, and other
morphological operations. These methods artificially expand the dataset by introducing variations in the images, simulating different real-world scenarios.

In our case, we used the terminal to execute the augmentation process systematically. For each type of augmentation, we created a separate Python file that specified the desired transformation method. These files were then executed through the terminal, allowing us to automate the process and ensure consistency. As a result, we generated three distinct types of augmentation: zoom, brightness, and random distortion. These augmentations not only diversified the dataset but also provided additional training samples, which ultimately enhanced the model's ability to detect a wider range of defects more effectively.

The code shown in Fig.5.45 imports the Augmentor library to perform image augmentation and creates a pipeline for images located in the specified directory. It applies zoom augmentation to the images, adjusting their size based on specified parameters. Finally, the code generates 40 augmented images, which are saved in the same directory as the original images, expanding the dataset for further use in model training.

```
import Augmentor
p = Augmentor.Pipeline(r"C:\Users\HP\Documents\i (1)\train")
p.zoom(probability=1, min_factor=0.8, max_factor=2)
p.sample(40)
```

Figure 5.45. Zoom augmentation

The code shown in Fig.5.46 imports the Augmentor library and creates a pipeline for augmenting images in a specified directory. It applies random brightness adjustments to the images with a probability of 80%, where the brightness is varied between a minimum factor of 0.7 and a maximum factor of 1.5. Finally, the code generates 40 augmented images, saving them in the same directory as the original images

```
import Augmentor
p = Augmentor.Pipeline(r"C:\Users\HP\Documents\i (1)\train")
p.random_brightness(probability=0.8, min_factor=0.7, max_factor=1.5)
p.sample(40)
```

Figure 5.46. Brightness augmentation

The code is shown in Fig.5.47 the image imports the Augmentor library and creates a pipeline for augmenting images from a specified directory. It applies random distortion to the images with a 30% probability. Finally, the code generates 40 augmented images, saving them in the same directory as the original images.

```
import Augmentor
p = Augmentor.Pipeline(r"C:\Users\HP\Documents\i (1)\train")
p.random_distortion(probability=0.3, grid_width=4, grid_height=4, magnitude=8)
p.sample(40)
```

Figure 5.47. Random distortion augmentation

In Fig. 5.48, it is shown that the Augmentor procedure was completed successfully, processing 40 images as intended. The script was executed without errors, and the augmentation task reached 100% completion.

Figure 5.48. command prompt

In Fig. 5.49 , After the augmentation process, the modified images were carefully organized and stored in a dedicated file to ensure systematic management and accessibility for model training and evaluation. The original dataset contained (818) labeled images, divided into (3) classes: cracks (353), holes (151), and obstacles (329), representing critical defect types in oil and gas pipelines. After augmentation, the dataset expanded to (1,612) images, enhancing its diversity and robustness.

In addition to labeled data, unlabeled images of clean pipeline interiors were included to improve model performance. Unlabeled data enhances training, especially in semi-supervised learning, by providing diverse real-world scenarios, improving generalization, and being cost-effective and abundant. Including these clean images reduced false positives and helped the model accurately distinguish between normal pipeline conditions and defects. This comprehensive dataset, combining labeled and unlabeled data, contributed to creating a reliable detection model capable of performing effectively in diverse real-world scenarios. [60].

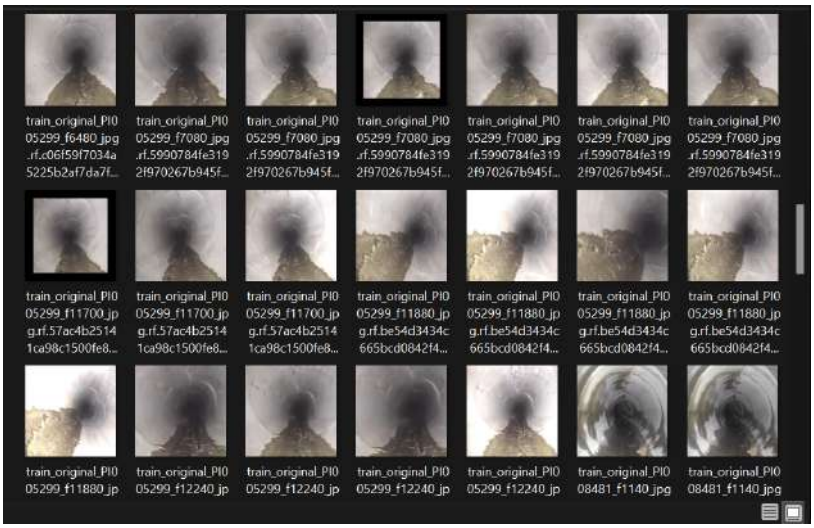


Figure 5.49. images after augmentation

5.8.4 Model Training

5.8.4.1 Object Detection

Object detection is a computer vision task that aims to locate objects in digital images. As such, artificial intelligence consists of training computers to see as humans do, specifically by recognizing and classifying objects according to semantic categories. Object localization is a technique for determining the location of specific objects in an image by restricting the object through a bounding box. Object classification is another technique that defines a detected object's category. The object detection task combines subtasks of object localization and classification to simultaneously estimate the location and type of object instances in one or more images [61].

5.8.4.2 WHY YOLO?

YOLO (You Only Look Once) is one of the most popular models used for detecting defects in oil and gas pipelines. It has been increasingly applied to the surface detection of pipeline weld defects due to its extremely fast detection capabilities, processing images at 45 frames per second (FPS) [62]. Additionally, YOLO achieves more than double the mean Average Precision (mAP) compared to other real-time systems, making it an excellent choice for real-time detection with both high speed and accuracy, while producing very few background errors. The open-source nature of YOLO has allowed the community to continuously improve the model. This collaborative development is one of the key reasons YOLO has seen such rapid advancements in a relatively short period.

5.8.4.3 Model Selection

In the process of model selection for detecting objects in the oil and gas pipeline environment, we undertook a systematic approach to evaluate multiple versions of the YOLO (You Only Look Once) model. The primary objective was to identify the most accurate and effective version suited to the unique challenges and requirements of this specialized environment. Given the critical nature of pipeline infrastructure, it was essential to select a model capable of reliably detecting various potential issues. The versions selected for this evaluation included YOLOv5n, YOLOv8s, YOLOv9t, YOLOv10n, and YOLOv11n, each of which offers distinct features and advancements in object detection.

To ensure a fair and unbiased comparison across all models, we standardized the training parameters for each version. Specifically, we set the number of training epochs to 5 and fixed the image size (imgsz) at 640 pixels. This uniformity in parameters ensured that each model was trained and tested under identical conditions, allowing us to focus on their inherent capabilities rather than differences in training configurations.

During training, we used an isolated and stable environment to run Jupyter Notebooks and ensured that our computer vision projects were executed consistently. As shown in Fig.8.1 in the appendix, this environment allows for seamless management of libraries and their versions without conflicts across projects. It ensures reproducibility, simplifies dependency management and provides a consistent workspace for machine learning and computer vision.

Also, during training, we used the env environment to ensure a controlled and consistent setup for our machine learning tasks. In Fig.8.2 in the appendix, the user activated the env environment and installed the Ultralytics package, commonly used for object detection tasks like YOLO. The output shows that Ultralytics (8.3.28) and its dependencies are already installed, confirming a ready-to-use setup for computer vision tasks.

In the appendix, this script showcases in Fig. 8.3 how to use the YOLO framework from the Ultralytics library to load and train a YOLOv5n model for object detection tasks. The model is initialized with pre-trained weights from the file yolov5n.pt. Training is performed using a dataset configuration file (data.yaml), which specifies the dataset paths and related settings. The training parameters are standardized, including epochs=5 to define the number of training iterations, imgsz=640 to set the image resolution, and device=0 to specify the hardware for execution (GPU). The model achieves a MAP50 = 0.699, as shown in Fig. 8.4 in the appendix.

In the appendix, this script showcases in Fig. 8.5 a custom training setup for the YOLOv8s model, trained for object detection. The pre-trained YOLOv8s weights (yolov8s.pt) are used as a starting point, while the dataset configuration is provided through the data.yaml file located in the dataset's directory. Key training parameters include epochs=5 to define the number of iterations, batch=16 to set the number of images processed in each batch, and imgsz=640 to fix the image size. Additionally, plots=True enables visualization of training progress and results. The model achieves a MAP50 = 0.643, as shown in Fig. 8.6 in the appendix.

In the appendix, this Python script demonstrates in Fig. 8.7 how to use the YOLO framework from the Ultralytics library to load and train a YOLOv9t model for object detection tasks. The model is initialized with pre-trained weights from the file yolov9t.pt. Training is performed using a dataset configuration file (data.yaml), which specifies the dataset paths and related settings. The training parameters are standardized, including epochs=5 to define the number of training iterations, imgsz=640 to set the image resolution, and device=0 to specify the hardware for execution (GPU). The model achieves a MAP50 = 0.69, as shown in Fig. 8.8 in the appendix.

In the appendix, this Python script demonstrates in Fig. 8.9 how to use the YOLO framework from the Ultralytics library to load and train a YOLOv10n model for object detection tasks. The model is initialized with pre-trained weights from the file yolov10n.pt. Training is performed using a dataset configuration file (data.yaml), which specifies the dataset paths and related settings. The training parameters are standardized, including epochs=5 to define the number of training iterations, imgsz=640 to set the image resolution, and device=0 to specify the hardware for execution (GPU). The model achieves a MAP50 = 0.524, as shown in Fig. 8.10 in the appendix.

In the appendix, this Python script demonstrates in Fig. 8.11 how to use the YOLO framework from the Ultralytics library to load and train a YOLOv11n model for object detection tasks. The model is initialized with pre-trained weights from the file yolov11n.pt. Training is performed using a dataset configuration file (data.yaml), which specifies the dataset paths and related settings. The training parameters are standardized, including epochs=5 to define the number of training iterations, imgsz=640 to set the image resolution, and device=0 to specify the hardware for execution (GPU). The model achieves a MAP50 = 0.656, as shown in Fig. 8.12 in the appendix.

After completing the training process for the YOLO model versions, we evaluated their performance based on the Mean Average Precision at 50% Intersection over Union (MAP50), which serves as a key metric for object detection accuracy [63]. The results are shown in the table V.IV:

YOLOv5n achieved a MAP50 score of 0.699, demonstrating strong performance in accurately detecting objects within the pipeline environment.

YOLOv8s performed slightly lower with a MAP50 of 0.643, still showing promising results but not as strong as the other models.

YOLOv9t, with a MAP50 of 0.690, also displayed comparable performance to YOLOv5n, making it a competitive option.

YOLOv10n, however, lagged behind with a MAP50 score of 0.524, indicating that it was less effective at object detection in this particular environment.

YOLOv11n achieved a MAP50 of 0.656, placing it somewhere in the middle of the rankings, showing acceptable results but still trailing behind the top-performing models.

Given these results, we identified the top three performing models—YOLOv5n, YOLOv9t, and YOLOv11n —as the most suitable candidates for further optimization. We decided to focus on fine-tuning these models to maximize their detection accuracy and ensure they perform optimally in detecting critical issues such as cracks, holes, and obstacles in the oil and gas pipeline environment. Fine-tuning involves adjusting model parameters, training with additional data, or applying advanced techniques to improve generalization and robustness in real-world conditions. This step is crucial in achieving the best results for accurate and reliable object detection, which is vital for maintaining safety and efficiency in the pipeline system.

TABLE V.IV

COMPARISON OF THE MODELS

Models name	MAP50
Yolov5n	0.699
Yolov8s	0.643
Yolov9t	0.69
Yolov10n	0.524
Yolov11n	0.656

5.8.4.4 Model Fine Tuning

```yaml
# Ultralytics YOLOv5 🚀, AGPL-3.0 license

# Parameters
nc: 80 # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.25 # layer channel multiple
anchors:
  - [10, 13, 16, 30, 33, 23] # P3/8
  - [30, 61, 62, 45, 59, 119] # P4/16
  - [116, 90, 156, 198, 373, 326] # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [
    [-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
    [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
    [-1, 3, C3, [128]],
    [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
    [-1, 6, C3, [256]],
    [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
    [-1, 9, C3, [512]],
    [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
    [-1, 3, C3, [1024]],
    [-1, 1, SPPF, [1024, 5]], # 9
  ]

```

Figure 5.50. yolov5n architecture

In the model selection subsubsection, as highlighted earlier, we prioritized accuracy as the primary criterion, which led to the selection of three candidate models: Yolov5n, Yolov9t, and Yolov11n. To further evaluate and refine these models, we proceeded with the Model Fine-Tuning stage. This process aims to identify the most suitable model for our objectives by iteratively optimizing their performance. We initiated the fine-tuning process with Yolov5n, setting the number of epochs to 100 to ensure adequate training iterations for the model to learn patterns effectively. The image size parameter (imgsz) was configured to 640, aligning with the recommended size for optimal detection performance. Additionally, we applied a freeze setting of 10, which was determined by analyzing the architecture of Yolov5n as shown in Fig.5.50. This value corresponds to the number of backbone layers in the model, allowing the initial layers to remain unaltered while fine-tuning the remaining layers to adapt to our specific dataset. Furthermore, the training was conducted on device=0, ensuring efficient utilization of computational resources.

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov5n.pt")

# Train the model
train_results = model.train(
    data="C:/Users/pca/Desktop/yolo5/yolov5Dataset/data.yaml",  # path to dataset YAML
    epochs=100,  # number of training epochs
    imgsz=640,  # training image size
    freeze=10,  #backbone layer
    device=0,  # device to run on
)
```

Figure 5.51. yolov5n with new parameters

After completing the training process for the Yolov5n model, we observed a notable performance improvement compared to the previous training iteration. The model's accuracy increased significantly, rising from 0.699 to 0.734. This improvement indicates that the adjustments made during the fine-tuning process, including the number of epochs, image size, and freeze settings, contributed effectively to enhancing the model's ability to detect and classify objects accurately.

These results highlight the potential of Yolov5n as a strong candidate among the selected models, and they serve as a promising step forward in identifying the most suitable model for our specific application. Further analysis and comparison with the remaining models will help finalize the optimal choice.

```
100 epochs completed in 1.983 hours.
Optimizer stripped from runs\detect\train2\weights\last.pt, 5.3MB
Optimizer stripped from runs\detect\train2\weights\best.pt, 5.3MB

Validating runs\detect\train2\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLOv5n summary (fused): 193 layers, 2,503,529 parameters, 0 gradients, 7.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<0
                   all        114        186      0.836      0.686      0.734      0.531
                 crack         44         97      0.755      0.309      0.394      0.246
                  hole         14         14      0.854      0.929      0.945      0.743
              obstacle         38         75      0.898      0.821      0.863      0.602
Speed: 1.1ms preprocess, 7.2ms inference, 0.0ms loss, 3.0ms postprocess per image
Results saved to runs\detect\train2
```

Figure 5.52. yolov5n with new MAP50

```yaml
# YOLOv9

# parameters
nc: 80  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple
#activation: nn.LeakyReLU(0.1)
#activation: nn.ReLU()

# anchors
anchors: 3

# gelan backbone
backbone:
  [
   # conv down
   [-1, 1, Conv, [16, 3, 2]],  # 0-P1/2
   # conv down
   [-1, 1, Conv, [32, 3, 2]],  # 1-P2/4
   # elan-1 block
   [-1, 1, ELAN1, [32, 32, 16]],  # 2
   # avg-conv down
   [-1, 1, AConv, [64]],  # 3-P3/8
   # elan-2 block
   [-1, 1, RepNCSPELAN4, [64, 64, 32, 3]],  # 4
   # avg-conv down
   [-1, 1, AConv, [96]],  # 5-P4/16
   # elan-2 block
   [-1, 1, RepNCSPELAN4, [96, 96, 48, 3]],  # 6
   # avg-conv down
   [-1, 1, AConv, [128]],  # 7-P5/32
   # elan-2 block
   [-1, 1, RepNCSPELAN4, [128, 128, 64, 3]],  # 8
  ]
```

Figure 5.53. yolov9t architecture

Upon completing the training and evaluation of the Yolov5n model, the next step in the fine-tuning process is to transition to the Yolov9t model. Similar to the approach taken with Yolov5n, we have carefully configured the training parameters to optimize the model's performance.

For the Yolov9t model, the number of training epochs is set to 100, ensuring sufficient iterations for the model to learn and adapt to the data. The image size parameter (imgsz) is maintained at 640, which provides a balance between computational efficiency and the level of detail captured in the input images. Additionally, a freeze parameter of 9 is applied, which was determined through an analysis of the Yolov9t model's architecture as shown in Fig.**??**. This freeze value corresponds to the number of backbone layers in the architecture, allowing us to preserve the pre-trained weights of these initial layers while fine-tuning the remaining layers to better align with our dataset.

The rationale behind setting the freeze value to 9 is to leverage the foundational feature extraction capabilities of the backbone layers while enabling the model to learn task-specific features in the deeper layers. This approach helps improve generalization and

enhances the model's performance on the target application. Furthermore, training will be conducted using device=0 to utilize the designated computational hardware effectively.

By adopting a structured and parameter-optimized training process for Yolov9t, we aim to evaluate its performance and compare it with the results from Yolov5n. This systematic fine-tuning will provide insights into the suitability of Yolov9 for our specific requirements and contribute to identifying the most accurate and robust model among the candidates.

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov9t.pt")

# Train the model
train_results = model.train(
   data="C:/Users/pca/Desktop/yolo9/yolov9Dataset/data.yaml",  # path to dataset YAML
    epochs=100,  # number of training epochs
    imgsz=640,  # training image size
    freeze = 9,
    device=0,  # device to run on
)
```

Figure 5.54. yolov9t with new parameters

Upon completing the training of the Yolov9t model, we observed a significant improvement in its performance compared to previous iterations. The accuracy of the model increased from 0.69 to 0.736, reflecting the positive impact of the fine-tuning process. This improvement suggests that the adjustments made to the training parameters—such as the number of epochs, image size, and freeze value—successfully enhanced the model's ability to detect and classify objects within the dataset.

The increase in accuracy demonstrates the effectiveness of the Yolov9t model's architecture, particularly the configuration of its backbone layers. By leveraging these layers through targeted fine-tuning, the model was able to achieve better feature extraction and representation, contributing to its improved performance. These results position Yolov9t as a strong contender for the best-performing model among our candidates, pending further evaluation and comparison with Yolov11n.

```
100 epochs completed in 4.082 hours.
Optimizer stripped from runs\detect\train5\weights\last.pt, 4.6MB
Optimizer stripped from runs\detect\train5\weights\best.pt, 4.6MB

Validating runs\detect\train5\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLOv9t summary (fused): 486 layers, 1,971,369 parameters, 0 gradients, 7.6 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:04<0
                   all        123        197      0.813      0.711      0.736      0.553
                 crack         46         99       0.67      0.323      0.375       0.25
                  hole         14         14      0.889      0.929      0.924      0.745
              obstacle         47         84      0.879      0.881      0.909      0.666
Speed: 1.4ms preprocess, 22.6ms inference, 0.0ms loss, 2.1ms postprocess per image
Results saved to runs\detect\train5
```

Figure 5.55. yolov9t with new MAP50

```yaml
# Ultralytics YOLO 🚀, AGPL-3.0 license
# YOLO11 object detection model with P3-P5 outputs. For Usage examples see https://docs.ultralytics.com/tasks/detect

# Parameters
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=yolo11n.yaml' will call yolo11.yaml with scale 'n'
  # [depth, width, max_channels]
  n: [0.50, 0.25, 1024] # summary: 319 layers, 2624080 parameters, 2624064 gradients, 6.6 GFLOPs
  s: [0.50, 0.50, 1024] # summary: 319 layers, 9458752 parameters, 9458736 gradients, 21.7 GFLOPs
  m: [0.50, 1.00, 512] # summary: 409 layers, 20114688 parameters, 20114672 gradients, 68.5 GFLOPs
  l: [1.00, 1.00, 512] # summary: 631 layers, 25372160 parameters, 25372144 gradients, 87.6 GFLOPs
  x: [1.00, 1.50, 512] # summary: 631 layers, 56966176 parameters, 56966160 gradients, 196.0 GFLOPs

# YOLO11n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 2, C3k2, [256, False, 0.25]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 2, C3k2, [512, False, 0.25]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 2, C3k2, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 2, C3k2, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9
  - [-1, 2, C2PSA, [1024]] # 10

```

Figure 5.56. yolov11n architecture

After completing the training and evaluation of the Yolov9t model, we move on to fine-tuning the Yolov11n model. Following the established methodology, we set the number of epochs to 100 to provide sufficient iterations for effective learning. The image size (imgsz) is configured to 640, balancing computational efficiency with detail capture. A freeze value of 11 is applied, determined by analyzing the Yolov11n architecture as shown in Fig.5.56, where the number of backbone layers is 11. This ensures that these foundational layers retain their pre-trained weights while allowing deeper layers to adapt to our specific dataset.

By freezing the backbone layers, we leverage their robust feature extraction capabilities, focusing the training on fine-tuning task-specific layers for improved accuracy and alignment with our objectives. Training is conducted on device=0.

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolo11n.pt")

# Train the model
train_results = model.train(
    data="C:/Users/pca/Desktop/yolo-3cls/petrov4/data.yaml",  # path to dataset YAML
    epochs=100,  # number of training epochs
    imgsz=640,  # training image size
    freeze=11, # freezing the backbone
    device=0,  # device to run on
)
```

Figure 5.57. yolov11n with new parameters

After completing the training of the Yolov11n model, we observed a significant improvement in its performance. The model's accuracy increased from 0.656 to 0.728, indicating that the fine-tuning process effectively enhanced its ability to detect and classify objects.

This improvement highlights the impact of carefully selecting training parameters, including the number of epochs, image size, and freeze value. By freezing the 11 backbone layers, we preserved their pre-trained feature extraction capabilities while fine-tuning the deeper layers to align with the specific characteristics of our dataset.

```
100 epochs completed in 1.465 hours.
Optimizer stripped from runs\detect\train\weights\last.pt, 5.5MB
Optimizer stripped from runs\detect\train\weights\best.pt, 5.5MB

Validating runs\detect\train\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLO11n summary (fused): 238 layers, 2,582,737 parameters, 0 gradients, 6.3 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<0
                   all        114        186      0.816      0.707      0.728      0.526
                 crack         44         97      0.628      0.366      0.375      0.216
                  hole         14         14      0.974      0.929      0.934      0.739
              obstacle         38         75      0.845      0.827      0.874      0.624
Speed: 0.9ms preprocess, 9.7ms inference, 0.0ms loss, 2.4ms postprocess per image
Results saved to runs\detect\train
```

Figure 5.58. yolov11n with new MAP50

The latest results from the fine-tuning process for the three models are as follows: Yolov5n's accuracy improved from 0.699 to 0.734, Yolov9's accuracy increased from 0.69 to 0.736, and Yolov11n's accuracy rose from 0.656 to 0.728. These results demonstrate the positive impact of the fine-tuning process on each model's ability to detect and classify objects. Among the three, Yolov9t achieved the highest accuracy, reaching 0.736, which makes it the best-performing model based on accuracy. This significant improvement in performance positions Yolov9t as the most suitable choice for the application so far.

After the previous results showed that the Yolov9t model achieved the highest accuracy of 0.736, the next step is to optimize its performance through additional fine-tuning. One potential improvement is enabling multi-scale=true, which introduces multi-resolution training. This technique involves training the model with images of varying sizes, allowing it to learn more generalized features across different resolutions. By exposing the model to these varying image scales, it can improve its ability to detect objects of different sizes and potentially enhance its overall accuracy.

Applying this setting during fine-tuning could lead to further gains in accuracy and overall model robustness, making Yolov9t even more effective.

```
from ultralytics import YOLO

# Load a model
model = YOLO("C:/Users/pca/Desktop/yolo9/runs/detect/train5/weights/best.pt")

# Train the model
train_results = model.train(
    data="C:/Users/pca/Desktop/yolo9/yolov9Dataset/data.yaml",  # path to dataset YAML
    epochs=100,  # number of training epochs
    imgsz=640,  # training image size
    multi_scale=True, #Multi-Resolution
)
```

Figure 5.59. yolov9t with the final parameters

After conducting fine-tuning on the Yolov9t model by enabling multi-scale training, we observed a significant improvement in its performance. The accuracy increased from 0.736 to 0.768, confirming that the addition of multi-resolution training helped enhance the model's ability to detect objects more effectively across different scales. This improvement demonstrates the effectiveness of the fine-tuning process, making this result the approved one. With this boost in accuracy, Yolov9t is now the most accurate and robust model for the task, making it the optimal choice in the application.

```
100 epochs completed in 8.837 hours.
Optimizer stripped from runs\detect\train6\weights\last.pt, 4.6MB
Optimizer stripped from runs\detect\train6\weights\best.pt, 4.6MB

Validating runs\detect\train6\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLOv9t summary (fused): 486 layers, 1,971,369 parameters, 0 gradients, 7.6 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:04<0
                   all        123        197       0.86      0.727      0.768      0.573
                 crack         46         99      0.742      0.384      0.403      0.247
                  hole         14         14      0.939      0.929      0.972       0.78
              obstacle         47         84      0.897      0.869       0.93      0.692
Speed: 0.8ms preprocess, 13.1ms inference, 0.0ms loss, 6.3ms postprocess per image
Results saved to runs\detect\train6
```

Figure 5.60. yolov9t with the final MAP50

This screenshot, as shown in Fig.5.61, displays the classList.txt file opened in a text editor, listing three object classes: crack, hole, and obstacle. These classes are utilized for object detection or classification in a YOLOv9t-based computer vision model.

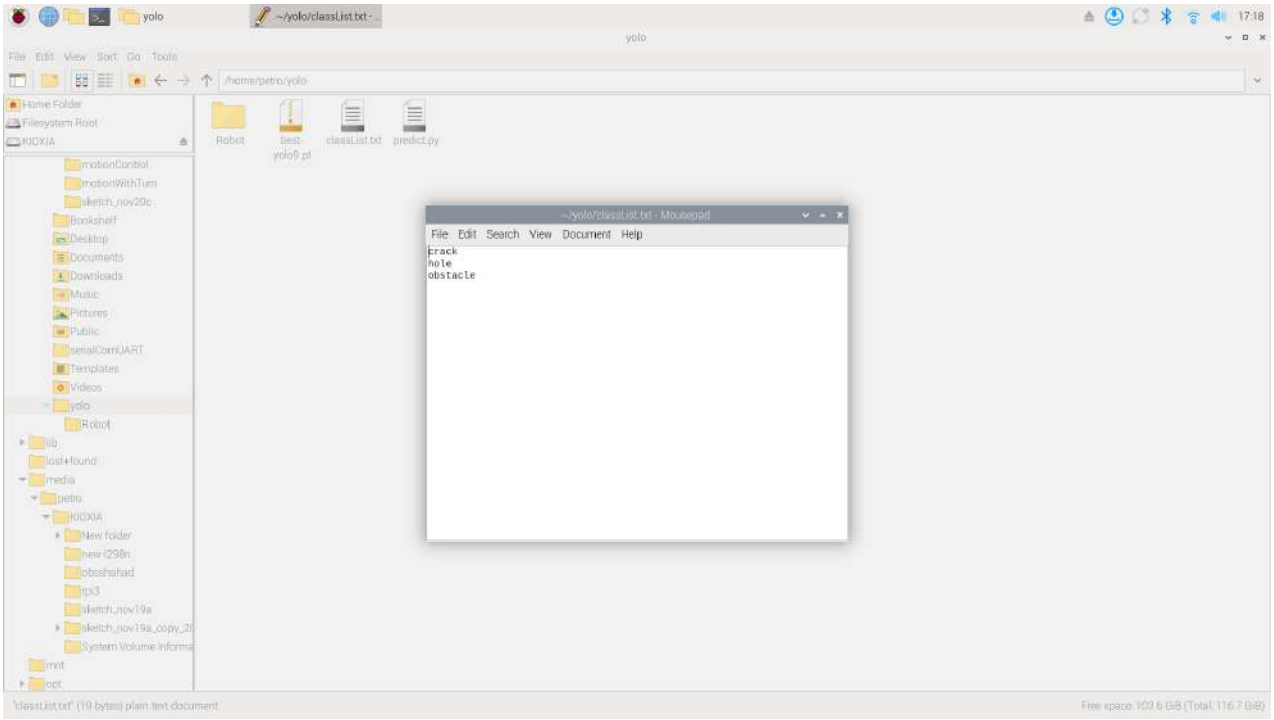


Figure 5.61. YOLOv9t Classes

We used the code, as shown in Fig.5.62, to inspect the camera connected to the Raspberry Pi using the YOLOv9t model. The model detects predefined object classes, such as crack, hole, and obstacle, as specified in the classList.txt file.

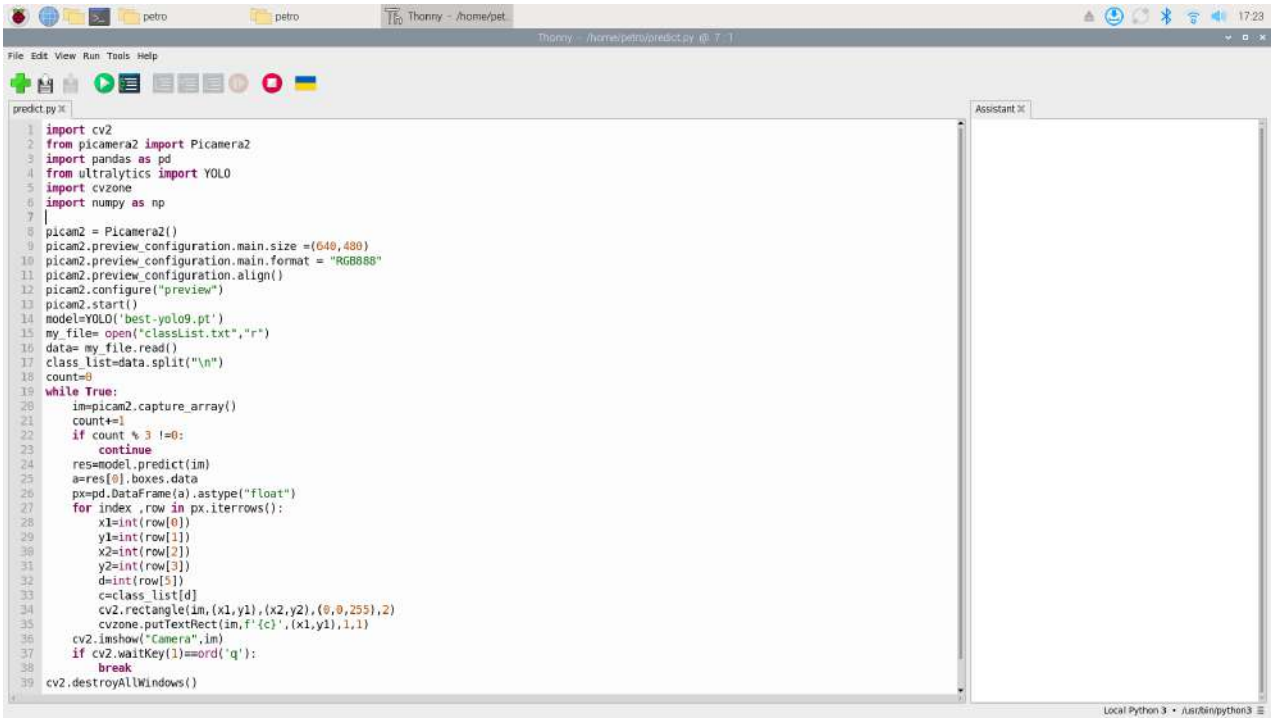


Figure 5.62. Camera inspection code

In conclusion

After evaluating and fine-tuning three models—Yolov5n, Yolov9t, and Yolov11n it was determined that Yolov9t emerged as the most accurate and robust model for the task. Initially, Yolov9t achieved an accuracy of 0.736, but after implementing multi-scale training (multi-resolution), the accuracy improved significantly to 0.768. This enhancement demonstrates the effectiveness of the fine-tuning process and the value of multi-scale training in increasing the model's performance. Therefore, Yolov9t, with an accuracy of 0.768, is the optimal model for the application, providing the best balance of precision and robustness. This result is now considered the approved solution for the project moving forward.

In addition to our work, another article titled "Design and Development of an In-Pipe Mobile Robot for Pipeline Inspection With AI Defect Detection System"[64] utilized the Yolov8 model for their defect detection tasks. Their reported metrics include an object loss of 1.4 and a classification loss of 1.3. In contrast, our implementation using the Yolo9t model achieved significantly better results, with a bounding box loss of 0.57 and a classification loss of 0.40.
These improved metrics highlight the superior accuracy and efficiency of Yolov9t in detecting and classifying defects. The lower loss values indicate better model convergence and a more precise understanding of object boundaries and classifications, making Yolov9t a more effective solution for similar tasks. This comparison further validates the robustness and reliability of our approach.

5.8.4.5 Robot Motion Control

Robot Navigation The navigation of the robot is achieved through a series of steps that enable it to detect obstacles and make informed decisions about movement. The core algorithm for navigation is based on the principles of obstacle avoidance, allowing the robot to maneuver through pipelines by treating the pipe walls as obstacles. This algorithm was chosen because the pipeline environment is relatively simple and predictable, making it easier to implement effective navigation strategies. The structured nature of pipelines allows for predictable movement patterns, enabling the algorithm to function effectively without complex adjustments. Additionally, the inability of humans to enter the pipes ensures that the environment remains static, distinguishing it from typical settings where objects can frequently move or change.

Three ultrasonic sensors are employed to measure distances: the front ultrasonic sensor serves as the primary sensor, continuously measuring the distance ahead. When this distance reaches a specific threshold, it indicates the presence of a wall or obstacle. This detection is then verified by the right and left ultrasonic sensors, which help confirm the nature of the obstruction and guide the robot's response.

Navigation Algorithm Steps:

1. Initialization: This step involves setting up the pins for the ultrasonic sensors and the H-bridge motor driver.
2. Measure Distance : Utilize the front ultrasonic sensor to determine the distance to the front. If this distance falls below a specified threshold, an obstacle is identified. The threshold is calculated as half the pipe diameter provided by the user, allowing the robot to approach the obstacle safely. This buffer space ensures that if the detected object is a wall, the robot can make a turn without colliding with it.
3. Obstacle Detection: if an obstacle is detected at front. Activate the left and right ultrasonic sensors to determine the available paths.
4. Decision Making:
    - If the right sensor detects a distance greater than the left sensor, it indicates a clear path to the right, prompting the robot to turn right.

- If the left sensor detects a distance greater than the right sensor, it indicates a clear path to the left, prompting the robot to turn left.
- If both sensors detect equal distances, this signifies an actual obstacle directly in front, and the robot should come to a stop.

5. Move Forward: If no obstacles are detected, the robot moves forward while continuously monitoring for defects using the camera.

Fig. 5.63 illustrates the function used to measure distance with the front ultrasonic sensor. Although all three functions employ the same calculation to measure distances in centimeters, each function is tailored for a specific sensor: one for the front sensor and two for the right and left sensors. This setup is integral to the robot's navigation system, allowing for effective obstacle detection.

```
int sensorLeft() {
  //pulse output
  digitalWrite(SLeftTrig, LOW);
  delayMicroseconds(4);
  digitalWrite(SLeftTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(SLeftTrig, LOW);

  long t = pulseIn(SLeftEcho, HIGH);//Get the pulse
  int cm = t / 29 / 2; //Convert time to the distance
  return cm; // Return the values from the sensor
}
int sensorFront(){
  //pulse output
  digitalWrite(SforTrig, LOW);
  delayMicroseconds(4);
  digitalWrite(SforTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(SforTrig, LOW);

  long t = pulseIn(SforEcho, HIGH);//Get the pulse
  int cm = t / 29 / 2; //Convert time to the distance
  return cm; // Return the values from the sensor
}
int sensorRight(){
  //pulse output
  digitalWrite(SRigthTrig, LOW);
  delayMicroseconds(4);
  digitalWrite(SRigthTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(SRigthTrig, LOW);

  long t = pulseIn(SRigthEcho, HIGH);//Get the pulse
  int cm = t / 29 / 2; //Convert time to the distance
  return cm; // Return the values from the sensor
}
```

Figure 5.63. Distance measurement function for ultrasonic sensors

Fig. 5.64 presents the code implementation for measuring distances to detect obstacles and make navigation decisions. This code includes the logic for evaluating sensor inputs and determining the appropriate actions the robot should take based on obstacle detection

```
int frontSensor = sensorFront();
if(frontSensor <= threshold){
int leftSensor = sensorLeft();
int rightSensor = sensorRight();
delay(1000);
if(rightSensor>leftSensor){
    right();
    delay(690);
  }
else if(leftSensor>rightSensor){
  left();
  delay(780);
  }
  else{
    Stop();
  }
}
 else{
   frontSensor = sensorFront();
  forward();
  delay(300);
  Stop();
if(frontSensor <= threshold){
  return;
}
else{
  Stop();
  delay(3000);
}
}
```

Figure 5.64. Code for Obstacle Detection and Navigation Decision Making

All code related to the robot navigation and movement is provided in Appendix section 8.1.6. **Robot Localization** Robot localization is critical for determining the robot's

position within a pipeline, especially when tasked with locating defects. In this system, an optical encoder connected to the Raspberry Pi is utilized to localize defects when detected and track the distance traveled by the robot. This combination ensures precise navigation and effective inspection throughout the pipeline. We will describe the cede lines and how the encoder measures distances:

- Import is for libraries such as:
  - Serial for enabling serial communication.
  - Time for time operation.
  - RPI.GPIO is for Raspberry Pi (General Purpose Input/Output) pins.
- Initialize the ‘ser‘ variable to initiate communication with the specific port.
- Set up GPIO mode to Broadcom and configure pin 17 with a pull-up resistor.
- initialize 4 variables for:
  - Track the state of rotation.
  - Holds the current state.
  - Keeps the last state.
  - Track the number of rotations.
- Define the ‘circ‘ variable using the following equation for distance measurement:
  - $C = \pi \times d$ (where C is circumference and d is diameter).
- Use a while loop for continuous reading of the encoder.
- Define the ‘distance‘ variable using the equation for calculating distance: total count multiplied by distance per step (40).

The Raspberry Pi code for the encoder is integrated with the Arduino motion control code through serial communication. All code related to the localization process and validation is provided in Appendix Fig. 8.15.

5.8.5 Database

This section describes the database system developed using XAMPP, which plays a vital role in the project. The database is structured to efficiently manage and store data related to the robot's operations, allowing for seamless interaction among users, parameters, and defect information. It consists of three tables Users, Parameters, and Defects each designed to fulfill specific data needs, ensuring a robust and scalable system. This setup underpins the overall functionality of the robot, facilitating effective monitoring and analysis of its performance in the designated environment.

5.8.5.1 Users Table

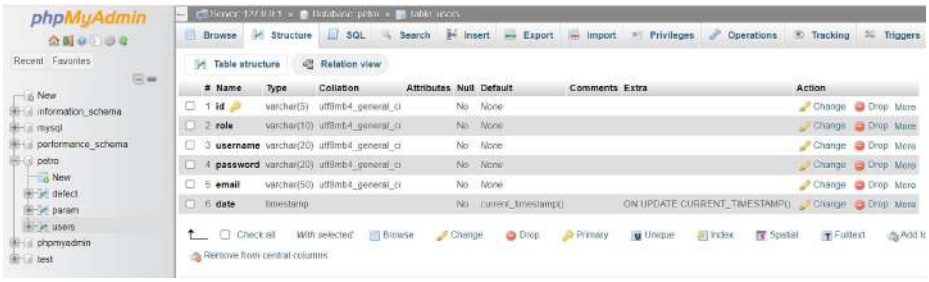


Figure 5.65. Users table from database

This table stores information about the users who interact with the robot system, facilitating user access management and activity tracking. As shown in Fig. 5.65 , it includes the following fields:

- ID: A unique identifier for each user consists of 5-digits (Primary Key).
- Role: The level of access granted to the user (admin or supervisor).
- Username: The name chosen by the user for login purposes 20 characters maximum.
- Password: A secure password for user authentication 20 characters maximum.
- Email: The user's email address for communication and notifications.
- Date: The date when the user account was created.

5.8.5.2 Parameters Table

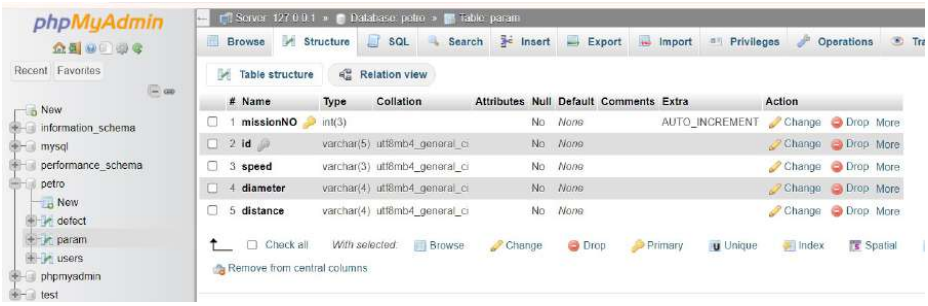


Figure 5.66. Parameters table from database

The Parameters table is designed to store various operational settings for the robot, which are provided by the admin. This allows for easy adjustments and configurations of the robot's functionality. As shown in Fig. 5.66 , it includes the following fields:

- MissionNO: A unique identifier for each mission (Primary Key, auto-increment).
- ID: A foreign key referencing the admin id who initiated the mission (Foreign Key).
- Speed: The speed setting for the robot during the mission.
- Diameter: The diameter of the pipeline being navigated.
- Distance: The distance to be traveled by the robot during the mission.

In the Parameters Table, the foreign key constraint for ID creates a link to the Users Table.

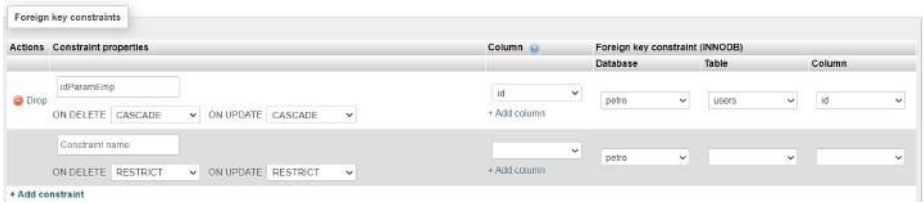


Figure 5.67. Parameters table foreign key constraints from database

As shown in Fig. 5.67, this constraint includes the following actions:

- ON DELETE CASCADE: If a user record is deleted from the Users table, all related entries in the Parameters Table will also be automatically deleted. This prevents orphaned records and helps maintain data integrity.

- ON UPDATE CASCADE: If the ID is changed in the Users table, those changes will be automatically applied to the Parameters Table, ensuring consistency throughout the database.

5.8.5.3 Defects Table

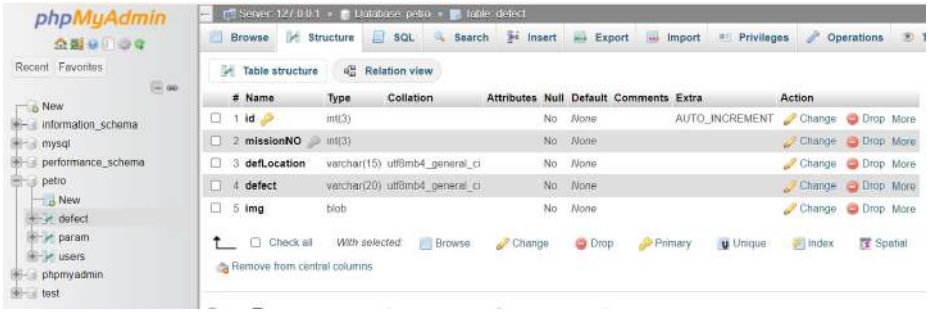


Figure 5.68. Defects table from database

The Defects Table is designed to record any defects detected by the robot during its operation, allowing users to track issues and maintain performance. As shown in Fig. 5.68 , it includes the following fields:

- DefectId: A unique identifier for each defect entry (Primary Key, auto-increment).
- MissionNO: A foreign key referencing the MissionNO from the Parameters Table, indicating the mission during which the defect was detected (Foreign Key).
- DefLocation: The location of the defect within the pipeline.
- Defect: The type of defect identified.
- Img : An image file associated with the defect for visual reference.

In the Defects Table, the foreign key constraint for MissionNO establishes a link with the MissionNO in the Parameters Table.

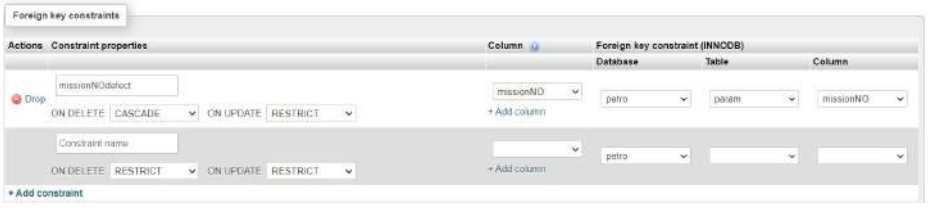


Figure 5.69. Defects table foreign key constraints from database

As illustrated in Fig. 5.69, this constraint includes the following actions:

- ON DELETE CASCADE: If a record in the Parameters Table is deleted, all associated records in the Defects Table will also be automatically removed. This ensures that there are no orphaned entries, thereby maintaining data integrity.
- ON UPDATE RESTRICT: If there is an attempt to modify the MissionNO in the Parameters Table, the update will be restricted if there are related records in the Defects Table. This prevents changes that could create inconsistencies in the database.

The code used to establish a connection to the database is shown in Fig. 5.70 below.

connect.php

```php
<?php
//DB info
$host = "localhost";
$dbusername = "root";
$dbpassword ="";
$dbname = "petro";
$dbn = "mysql:host=$host;dbname=$dbname";

try {
$pdo = new PDO($dbn,$dbusername,$dbpassword);// connect to db
$pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);//error handling
}
catch (PDOException $e){
  echo "connection failed: " . $e->getMessage();
}
```

Figure 5.70. Connect to database Code

5.8.6 Webpage

This section outlines the functionality of the webpage designed for managing user interactions within the system. The webpage features a structured signup and login process, enabling one Admin and multiple Supervisors to access the platform. The Admin has full control over the system, with access to all tabs, including setting parameters, viewing the dashboard, and accessing reports. In contrast, Supervisors have limited access, with the ability to view the dashboard and access reports. This role-based access ensures effective management and oversight of the robot's operations while maintaining security and organization within the system.

5.8.6.1 User Registration

In the user registration scenario, an Admin begins by filling out the registration form, as illustrated in Fig. 5.71, which includes fields for their username, password, email address, and role designation as Admin.

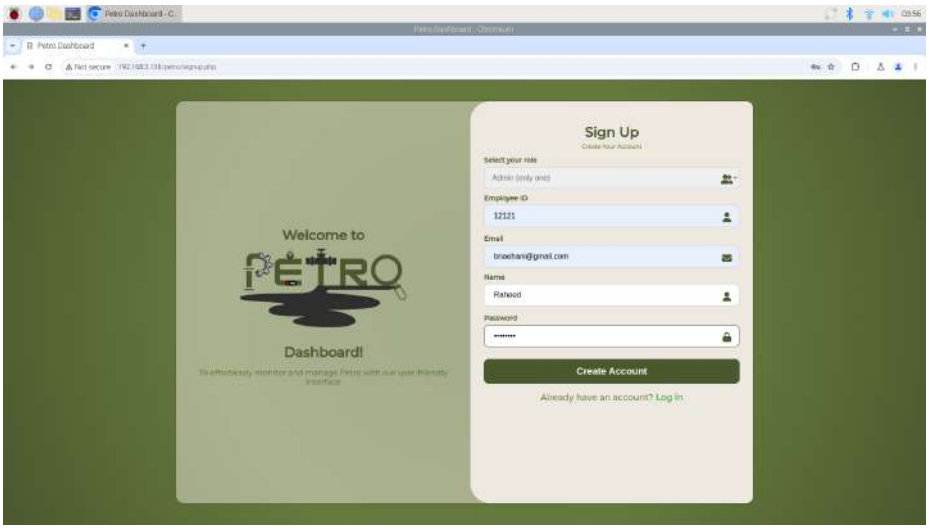


Figure 5.71. User registration form

Once the Admin submits the form, the system verifies the information and successfully creates the Admin account, granting full access to all system functionalities. After registration, the system directs the Admin to the login page.

Following this, multiple Supervisors can register by completing the same form, as illustrated inFig. 5.72, with their own unique usernames, passwords, and email addresses. Upon submission, the system checks for existing accounts and confirms their status as Supervisors, allowing them to create their accounts. After all registrations are complete, Supervisors are also redirected to the login page.

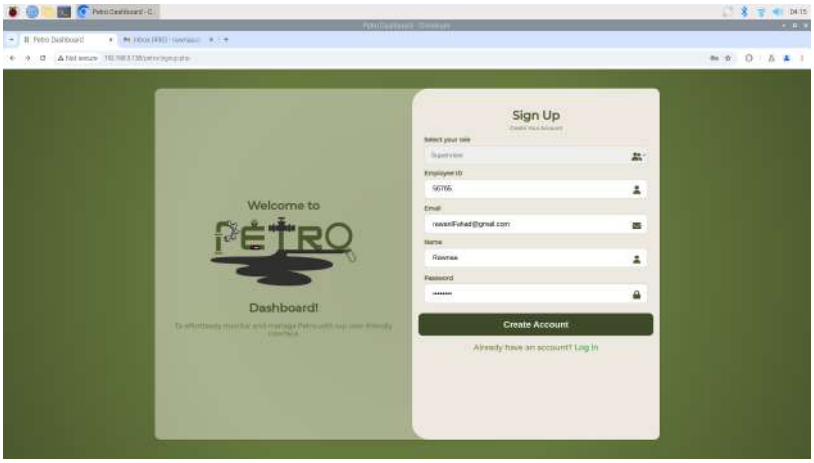


Figure 5.72. User registration form

After all registrations are complete, Supervisors are also redirected to the login page, where they can enter their credentials to access the platform. This structured process ensures that the Admin has control over system parameters while Supervisors can access the dashboard and reports, maintaining a clear distinction between user roles.

The details of both the Admin and Supervisor registrations are saved to the database, as shown in Fig. 5.73, ensuring that user information is securely stored and easily retrievable for future logins.

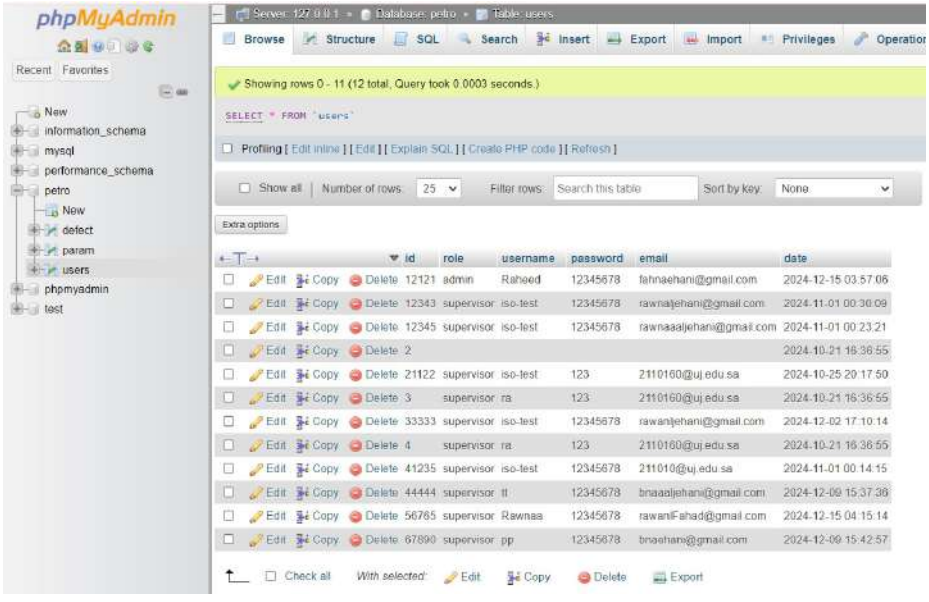


Figure 5.73. Users table after Admin registration

All code related to the signup process and database insertion is provided in Appendix section 8.1.7.

5.8.6.2 User Login

The user login process is designed to provide secure access to the system for both Admins and Supervisors. When users reach the login page, they are required to enter their credentials, which include their id, name,and password ,as shown in Fig. 5.74.

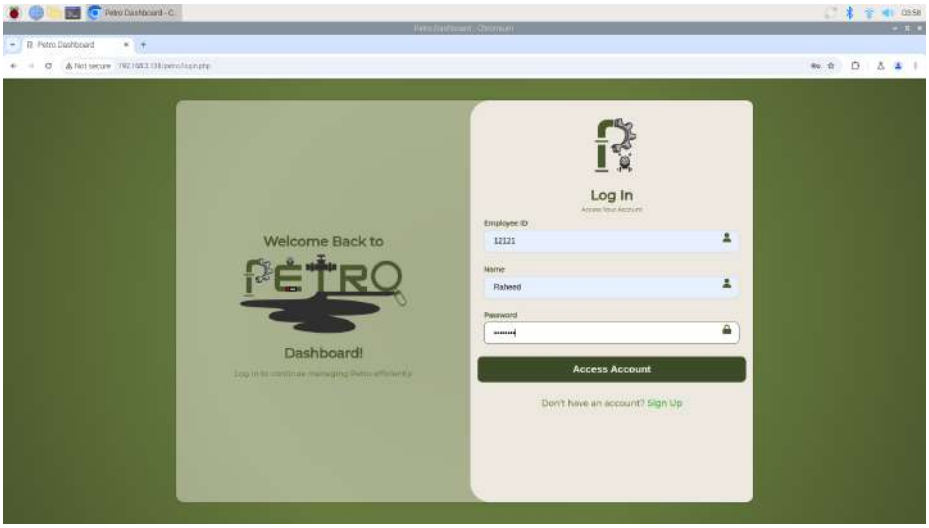


Figure 5.74. Login page

The authentication mechanism checks the submitted information against the database records. If the credentials correspond to an existing account, the system allows access to the user's assigned functionalities. The Admin has full access to all tabs, including setting parameters, viewing the dashboard, and accessing reports, as shown in Fig. 5.75.

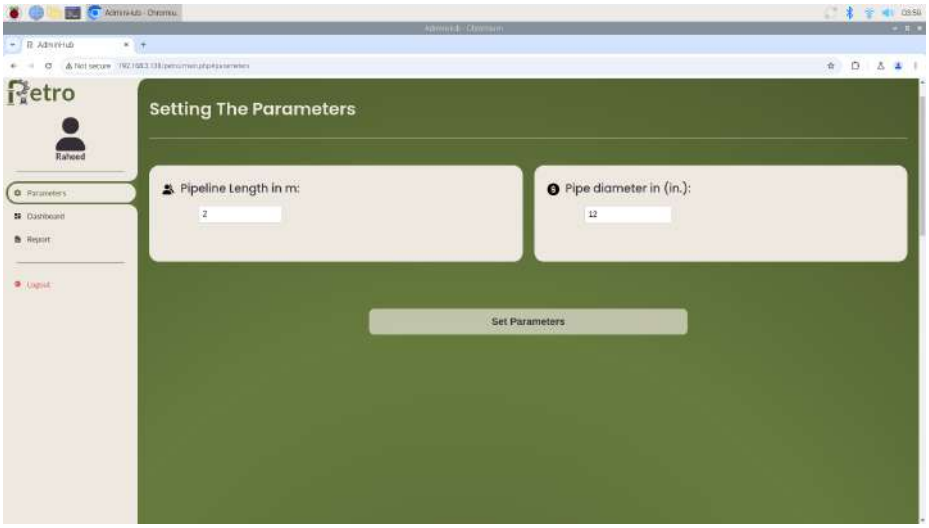


Figure 5.75. admin access to system functionalities

Conversely, Supervisors are limited to the dashboard and reports only, as shown in Fig. 5.76.

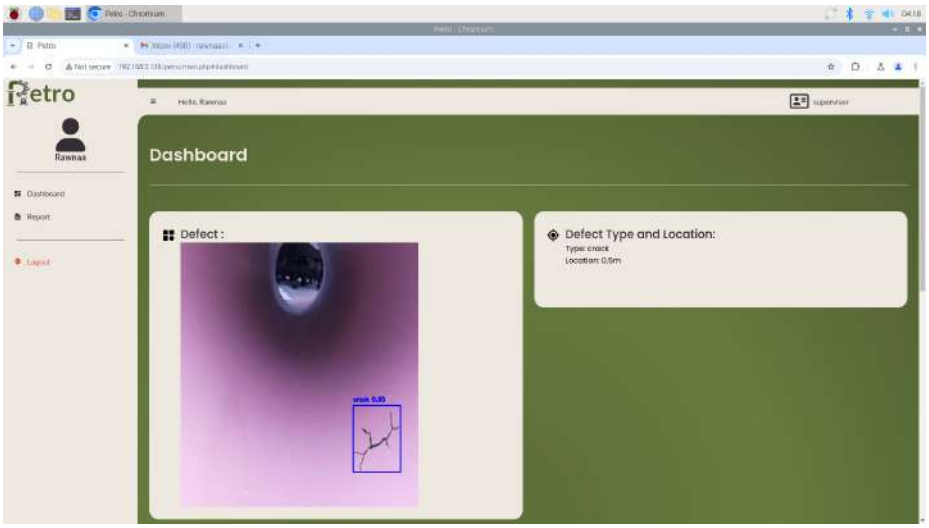


Figure 5.76. Supervisor access to system functionalities

All code related to the log in process and database validation is provided in Appendix section 8.1.8.

5.8.6.3 Main Page

The main page serves as the central hub for users to navigate and access various functionalities within the system. The layout is intuitive, featuring a clean interface that allows users to easily find the tools and information they need based on their roles.

Tabs:

- Set Parameters: This tab allows the Admin to configure essential operational settings for the robot, including the distance to travel and the diameter of the pipe. It is important to note that the pipe diameter must be greater than 10 units to ensure that the robot can fit comfortably within the pipeline.
- Dashboard: The dashboard presents real-time defect detection and localization data performed by the robot during missions. This feature is available to both Admins and Supervisors, providing critical insights into current operations, as illustrated in Fig. 5.77.

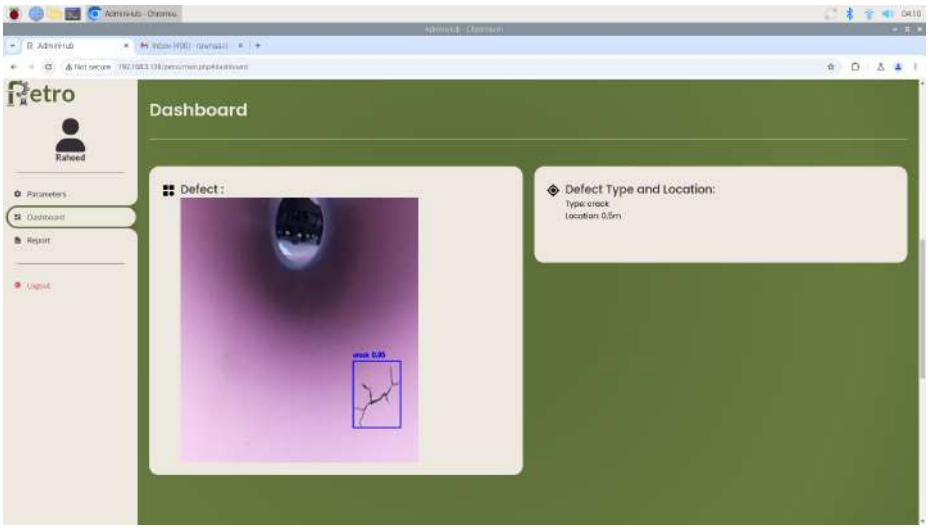


Figure 5.77. Dashboard tab

- Reports: This tab displays all detected defects and their locations from the last mission performed by the robot. Both Admins and Supervisors can access this information to monitor system performance and address any issues promptly, as shown in Fig. 5.78.

Figure 5.78. Report tab

5.9 Conclusion

In conclusion, this chapter offers a comprehensive examination of the hardware and software integral to our project. By detailing the specific hardware components and the software frameworks employed, alongside visual aids such as the block and circuit diagrams, the chapter illustrates the system's architecture and component interactions. The discussion of the prototype further emphasizes our design's practical application, while the implementation details provide insight into the methodologies that guided the project's development. This thorough analysis aims to clarify the technological foundation of our project and the reasoning behind our design decisions, ensuring a solid understanding for all stakeholders involved.

Chapter 6

Testing

6.1 Introduction

In this chapter, we provide a detailed discussion of the testing strategies employed to test and validate the performance, and functionality of the autonomous In-Pipe Inspection Robot(IPIR). We begin with section 6.2 outlining the scenarios for unit testing and focusing on the web pages and hardware components to ensure they operate correctly in isolation, Next with section 6.3 we detail the integration testing scenarios which evaluate how different subsystems such as robot motion, defect detection, localization, and website communication interact and function together. Following this in section 6.4 we present the validation and system testing of the robot's ability to autonomously navigate pipelines, detect and classify defects, and relay real-time data to the web dashboard while ensuring accurate data storage in the database. Finally, we conduct a comparative analysis in section 6.5 of the autonomous In-Pipe Inspection Robot(IPIR)robot with previous systems, focusing on key aspects such as autonomy, localization, real-time detection, and system design. We also perform a performance analysis 6.6 evaluating the system's efficiency, accuracy, and reliability, contributing to an overall assessment of the robot's effectiveness in meeting its objectives.

6.2 Scenario Details for unit testing

Unit testing is a crucial step in ensuring that individual components of a system perform as expected. It involves testing the smallest units of software and hardware in isolation to verify their functionality. In our proposed defect detection system, the key components include hardware elements such as sensors (ultrasonic sensor, camera, encoder), as well as software functionalities for defect classification, reporting, and user interaction. The unit tests outlined in Table VI.I aim to test these components independently to ensure they function correctly.

TABLE VI.I

TEST UNIT SCENARIOS

Test	Input	Output	Expected Result
Case 1	Sign up page: Email and password	pass	Correct email and password
Case 2	Sign up page: Email and password	Fail	Incorrect email or password or both
Case 3	Login page: Email and password	pass	Correct email and password
Case 4	Login page: Email and password	Fail	Incorrect email or password or both
Case 5	Parameters page: Pipeline length	length value (m)	1.5 m, 7 m, 3 m
Case 6	Parameters page: Pipeline diameter	diameter (in)	Greater than 10 inches
Case 7	Dashboard page: OV5647 camera	Defect type	crack, hole, obstacle
Case 8	Dashboard page: Encoder	Defect location	0.5 cm
Case 9	Report page: Inspection Results	Report details	Defect Image, Defect location, Defect type
Case 10	Ultrasonic Sensor	Sense pipe walls and obstacles	Ensuring smooth navigation

6.2.1 Sign up Page Testing

To ensure the integrity of our website's sign up process, we will conduct the following test cases. Each test case will address specific scenarios to verify that our sign up system operates reliably and securely under different conditions.

- **Pass:** if the admin or supervisor writes the email form correctly.
- **Pass:** if the admin or supervisor chooses a password with 8 to 20 characters.
- **Pass:** if the admin or supervisor writes the name under 20 characters.
- **Pass:** if the admin or supervisor writes the ID in 5 numbers.
- **Fail:** if the admin or supervisor writes the email wrong.
- **Fail:** if the admin or supervisor chooses a password out of the range of 8 to 20 characters.
- **Fail:** if the admin or supervisor writes the name more than 20 characters.
- **Fail:** if the admin or supervisor writes the ID using characters.
- **Fail:** if there is an existing admin already.

Our testing cases are summarized in the table VI.II below :

TABLE VI.II

VALID/INVALID CASES IN SIGNUP PAGE

Case	Input Values	Expected Results	Result
1	Petro24@gmail.com	Valid email	Success
2	123456789	Valid password	Success
3	Petro24@gma.com	Invalid email	Failed
4	1234	Invalid password	Failed
5	86330	Valid ID	Success
6	863F	Invalid ID	Failed
7	Raghad	Valid name	Success
8	Raghadmohammedalghadmi	Invalid name	Failed

Fig. 6.1 shows a valid sign-up case where all inputs meet the specified requirements outlined in Table VI.II.

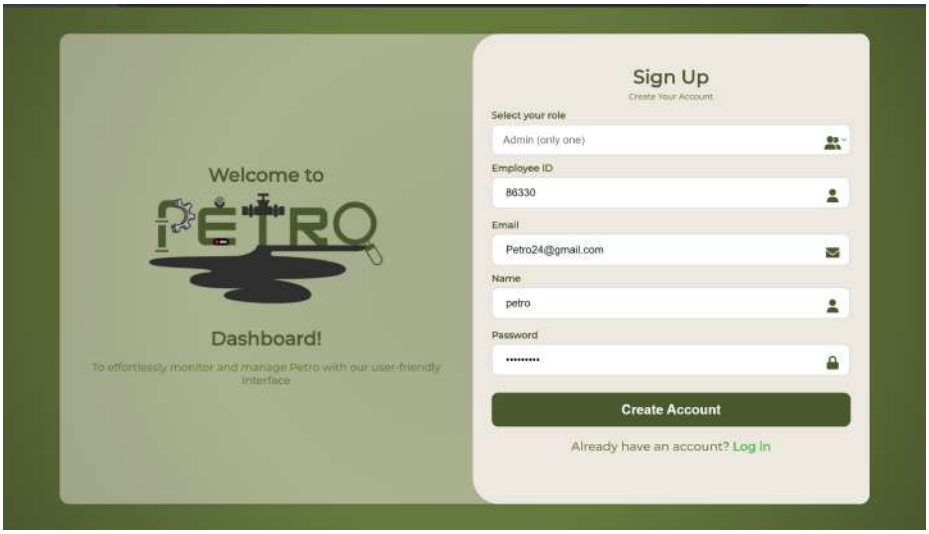


Figure 6.1. Valid Cases in Signup Page

Fig. 6.2 illustrates an invalid signup case, highlighting errors that do not meet the specified criteria, as specified in Table VI.II.

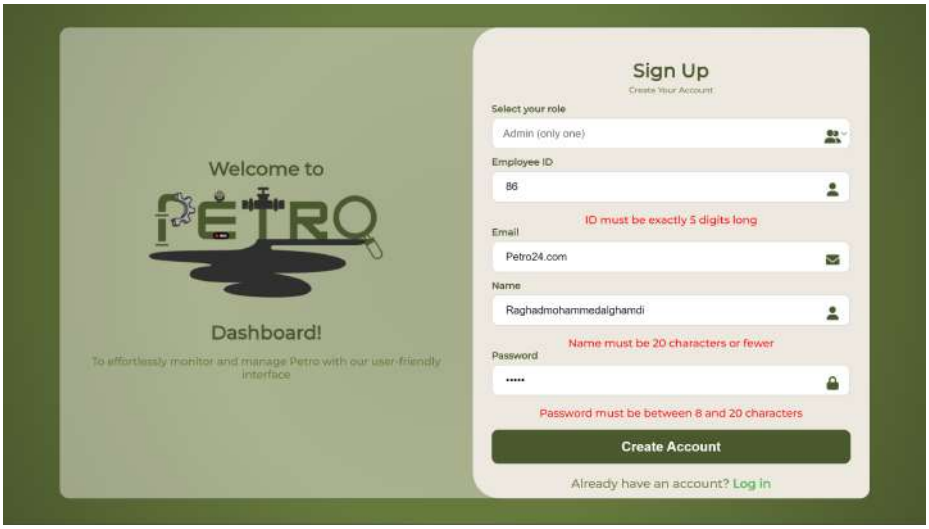


Figure 6.2. Invalid Cases in Signup Page

6.2.2 Login Page Testing

To ensure the integrity of our website's login process, we will conduct the following test cases. Each test case will address specific scenarios to verify that our login system operates reliably and securely under different conditions.

- **Pass:** if the admin or supervisor writes the correct name and password.
- **Fail:** if the admin or supervisor writes the name wrong.
- **Fail:** if the admin or supervisor writes the password wrong.

Our testing cases are summarized in the table VI.III below :

TABLE VI.III

VALID/INVALID CASES IN LOGIN PAGE

Case	Input Values	Expected Results	Result
1	Petro	Valid name	Success
2	123456789	Valid password	Success
3	Raghadmohammedalghadmi	Invalid name	Failed
4	1234	Invalid password	Failed

Fig. 6.3 showcases a successful login attempt with accurate credentials, as outlined in TABLE VI.III.

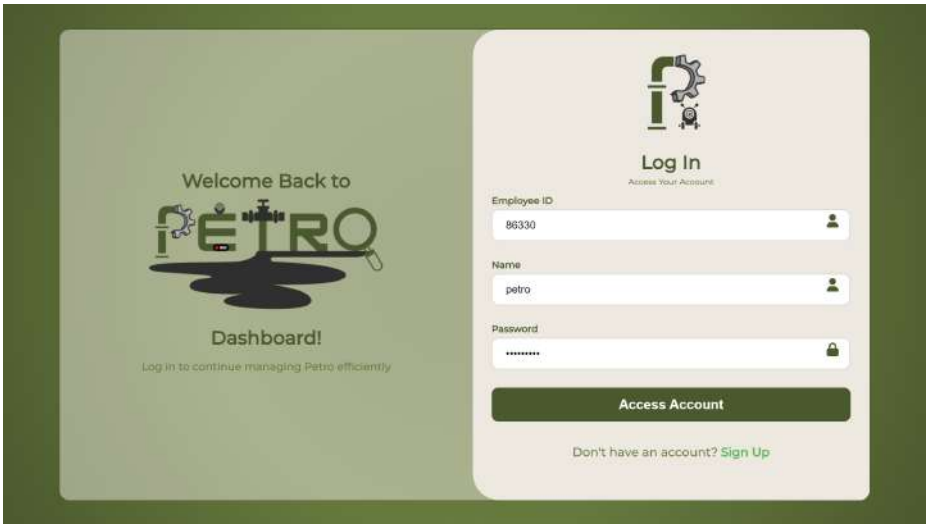


Figure 6.3. Valid Cases in Login Page

Fig. 6.4 highlights failed login attempts caused by incorrect username or password, as specified in TABLE VI.III.

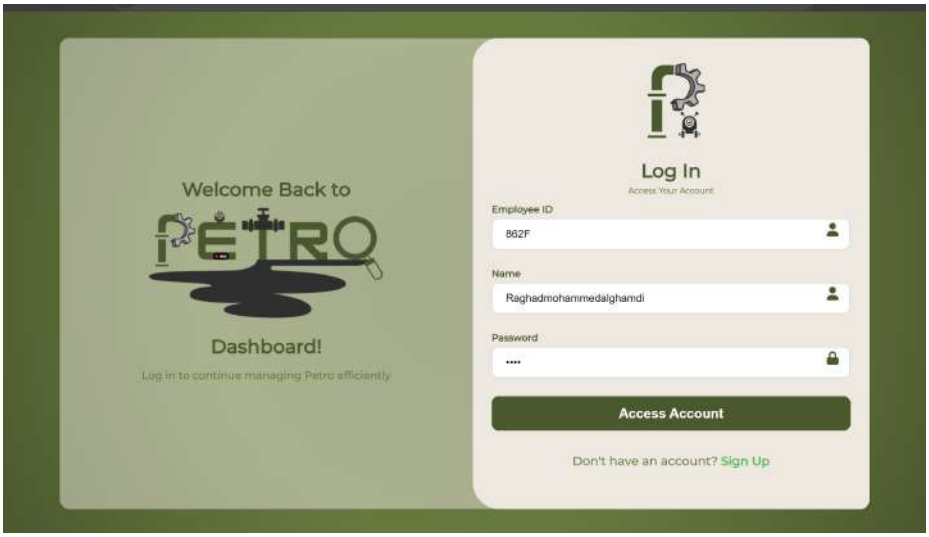


Figure 6.4. Invalid Cases in Login Page

6.2.3 Parameter Page Testing

To ensure the integrity of our website's parameter process, we will conduct the following test cases. Each test case will address specific scenarios to verify that our parameter system operates reliably and securely under different conditions.

- **Pass:** if the pipe length is a positive number (greater than zero).
- **Pass:** if the pipe diameter greater than 10.
- **Fail:** if the pipe length is zero or negative.
- **Fail:** if the diameter is less than 11 inches.

Our testing cases are summarized in the table VI.IV below :

TABLE VI.IV

VALID/INVALID CASES IN PARAMETER PAGE

Case	Input Values	Expected Results	Results
1	Pipe Length: 1.5 m	Valid Length Input	Success
2	Pipe Length: 0 m	Invalid Length (Fail: Not Greater Than Zero)	Failed
3	Pipe Length: -3 m	Invalid Length (Fail: Negative Value)	Failed
4	Pipe Diameter: 11 in	Valid Diameter Input	Success
5	Pipe Diameter: 7 in	Invalid Diameter (Fail: not greater than 10)	Failed

Fig. 6.5 displays valid parameter inputs as described in Table VI.IV.

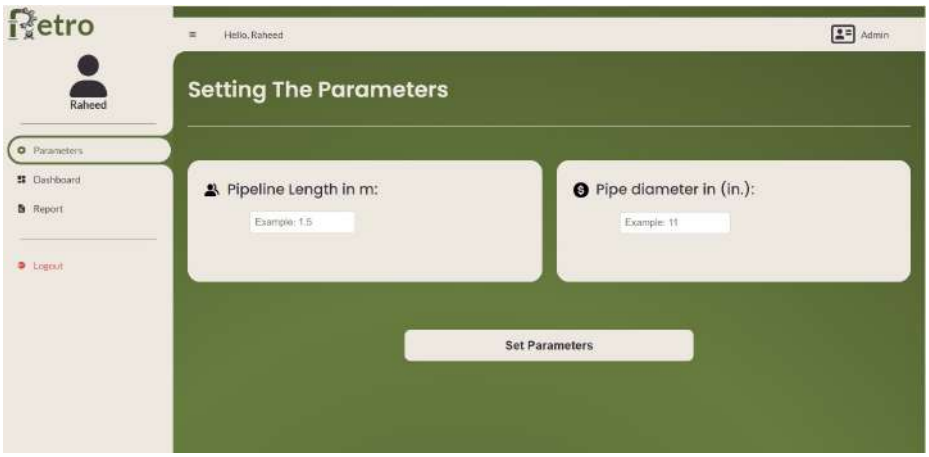


Figure 6.5. Valid Cases in Parameter Page

Fig. 6.6 illustrates invalid parameter inputs as outlined in Table VI.IV.

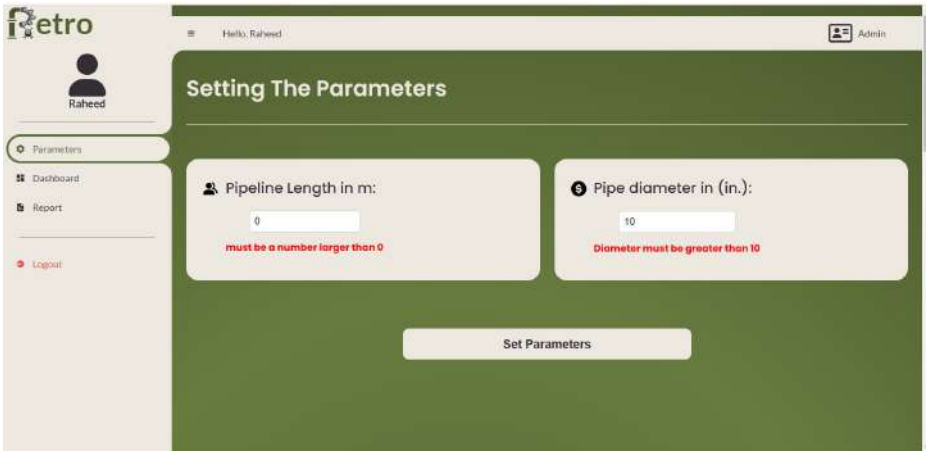


Figure 6.6. Invalid Cases in Parameter Page

6.2.4 Dashboard Page Testing

The Dashboard page is designed to display essential information in real-time during the defect detection process. It provides immediate feedback on the robot's performance of detected defects. The real-time data includes:

- **Defect Image:** Displays the captured image of the defect in real-time.
- **Defect Localization:** Displays the captured image of the defect in real-time.
- **Defect Type:** Identifies the type of defect, such as a crack, hole, or obstacle.

Fig. 6.7 showcases the Dashboard page, which provides real-time defect detection and displays key information such as the defect image, defect localization, and defect type. These features ensure immediate feedback on the robot's performance during the defect detection process, as described in the testing framework.

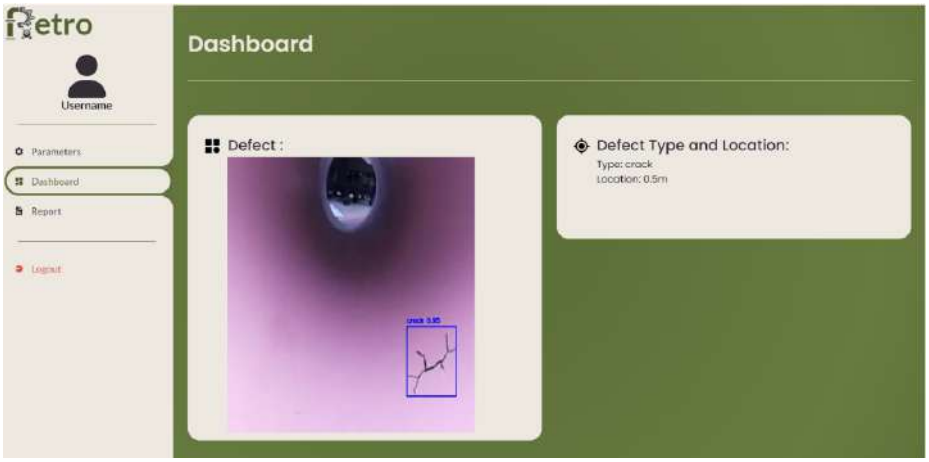


Figure 6.7. Dashboard Page

6.2.5 Report Page Testing

The Report page presents defect data retrieved from the database, offering a historical overview rather than real-time updates. It provides the same essential defect information as the dashboard but focuses on archived data for further analysis and documentation. The displayed data includes:

- **Defect Image:** Shows the stored image of the defect captured by the robot.
- **Defect Localization:** Provides the exact location of the defect as recorded in the database.
- **Defect Type:** Classifies the defect, such as a crack, hole, or obstacle.

Fig. 6.8 showcases the report page, which provides all the defect images of a mission.

Figure 6.8. Report Page

6.2.6 Camera Testing

This test verifies that the OV5647 camera module works as expected with the Raspberry Pi 4, ensuring it powers on, captures images, and transmits data correctly. The camera was connected using a ribbon cable and powered by a portable power bank, as shown in Fig. 6.9.

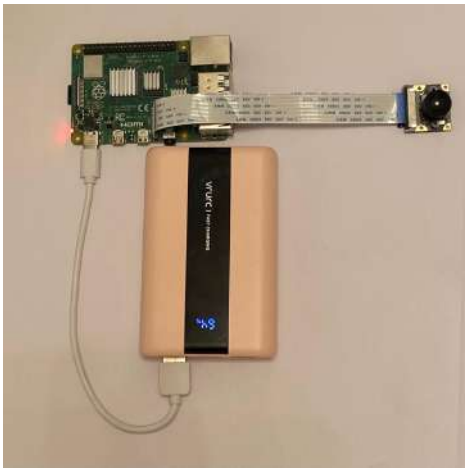


Figure 6.9. Camera testing

The test confirmed the successful operation of the OV5647 camera module. A sample output image in Fig. 6.10 shows the ability to detect pipeline defects, such as cracks and holes, with bounding boxes and labels displayed in real-time.

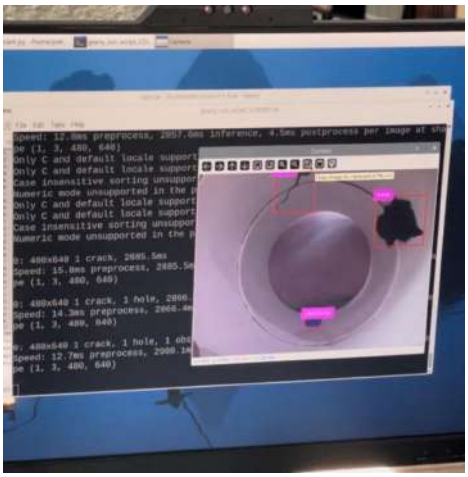


Figure 6.10. Sample Output of OV5647 Camera Module Testing

6.2.7 Ultrasonic testing

This test verifies the ultrasonic sensor's functionality when connected to the Arduino. The sensor was powered and tested to ensure it provides accurate distance measurements for navigation. Fig. 6.11 shows the sensor connected to the Arduino, confirming its successful operation.

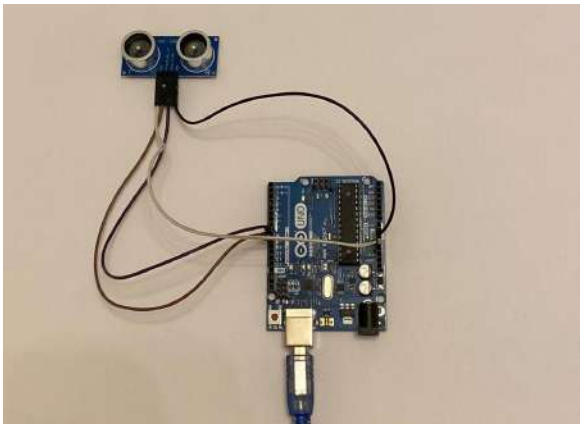


Figure 6.11. Ultrasonic Sensor Testing

This test was performed to validate that the ultrasonic sensor is working properly as shown in Fig. 6.12.

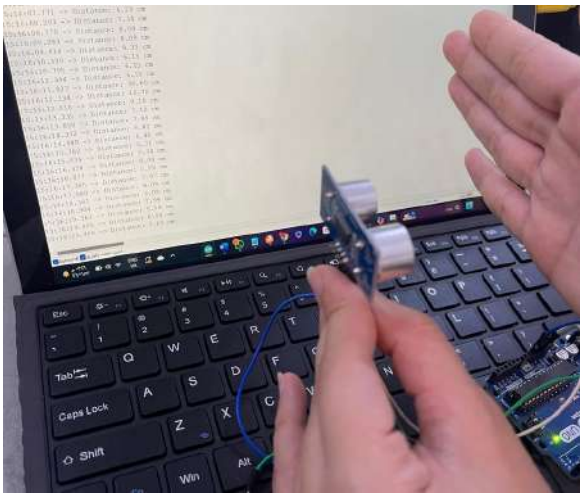


Figure 6.12. Ultrasonic Sensor Readings

This test is intended to verify that all three sensors are properly connected and functioning effectively together. Fig. 6.13 illustrates the wiring connections.

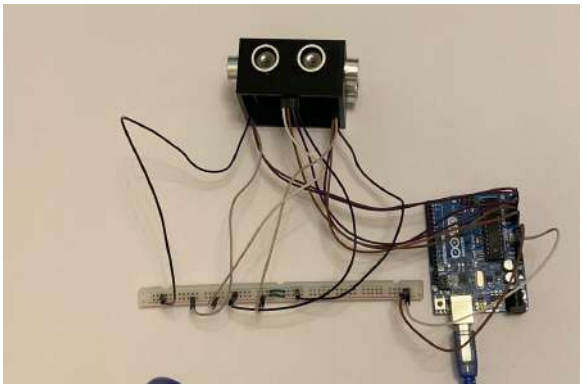


Figure 6.13. 3 Ultrasonic Sensors Testing

Figure 6.14 illustrates the results of testing the sensors in relation to one another. The data presented shows how each sensor responds under various conditions, highlighting their individual performance as well as their interactions.

Figure 6.14. Readings from the Ultrasonic Sensors

6.2.8 Encoder testing

This test verifies the functionality of the encoder when connected to the Raspberry Pi. It is connected to the motors via Arduino and a power supply as shown in Fig. 6.15 to ensure accurate distance measurements.

Figure 6.15. Encoder Testing

After verifying the correct connections of the circuit, Fig. 6.16 illustrates the encoder's readings, confirming their accuracy and proper functionality.

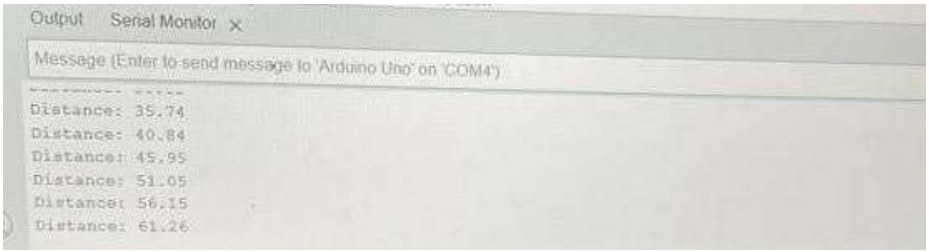


Figure 6.16. Readings from the encoder

6.3 Scenarios for Integration Testing

Integration testing relies heavily on the foundational elements established during unit testing. Its primary focus is to evaluate how individual components work together, ensuring their seamless integration. This process involves validating the interactions between these units. In our system, which comprises four main components, we will specifically test the connections and interactions between each of them. The goal of integration testing is to confirm that these components function correctly as a cohesive unit.

6.3.1 Robot Motion Integration Testing

In this section, the focus will be on testing the robot's ability to control its motion autonomously. Figure 6.17 demonstrates the setup, which contains wheels linked to motors connected to H bridge motor driver for control, connected to Arduino with a power supply. This setup validates the robot's capacity for efficient movement within the pipeline.

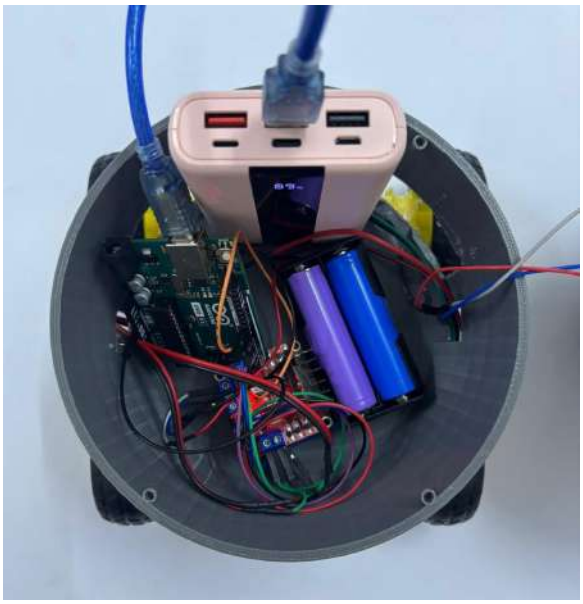


Figure 6.17. Motion Control Setup of The Robot

The first test is designed to verify the robot's ability to move forward within the pipe without any issues. Fig. 6.18 successfully validates this test case.

Figure 6.18. The Robot is Moving forward Inside the Pipe

The second test is designed to verify the robot's ability to make a turn whether it is right or left within the pipe without any issues. Figure 6.19 successfully validates this test case by turning left.

Figure 6.19. The Robot is Turning left Inside the Pipe

6.3.2 Defect detection and classification integration testing

This section evaluates the system's capability to utilize the camera for real-time detection and classification of defects, ensuring seamless integration within the overall defect detection system. The following real-time detection scenarios are tested:

- Case 1: Real-time Detection of a Hole — Ensures the camera immediately captures and identifies a hole as it appears in 1.7 s.

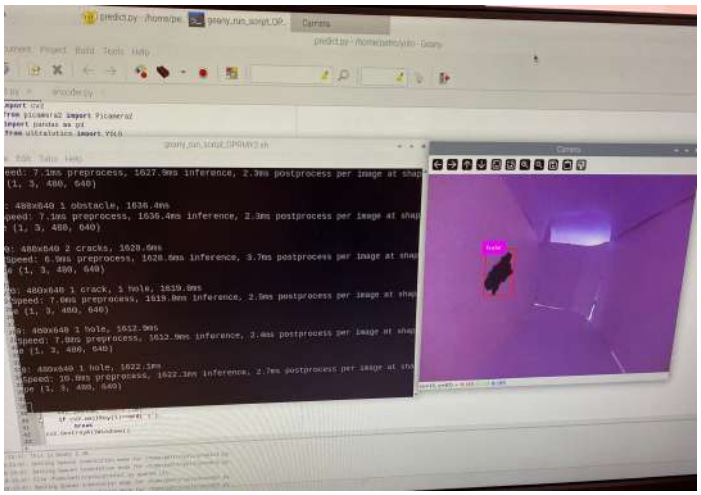


Figure 6.20. hole captured by the OV5647 camera during defect inspection

- Case 2: Real-time Detection of an Obstacle — ensures the camera immediately captures and identifies an Obstacle as it appears in 1.7 s.

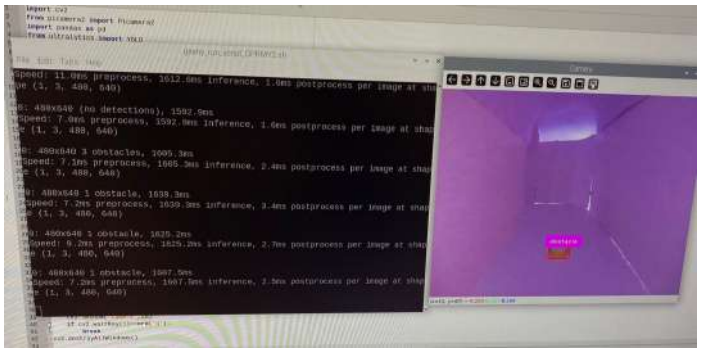


Figure 6.21. Obstacle captured by the OV5647 camera during defect inspection

- Case 3: Real-time Detection of a Crack — ensures the camera immediately captures and identifies a Crack as it appears in 1.7 s.

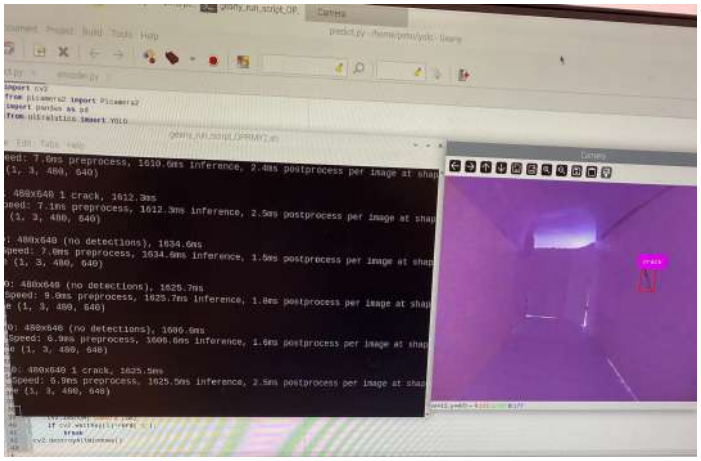


Figure 6.22. Crack captured by the OV5647 camera during defect inspection

These cases validate the camera's functionality in capturing real-time images of various defect types, contributing to effective defect detection and classification.

6.3.3 Localization integration testing

The location of the defect is measured by the encoder connected to the motors during the movement of the robot. After capturing the defect image and calculating the distance and saving it in the database, it will be displayed in the dashboard. Fig. 6.23 illustrates the location of the defect identified by the robot during its inspection process.

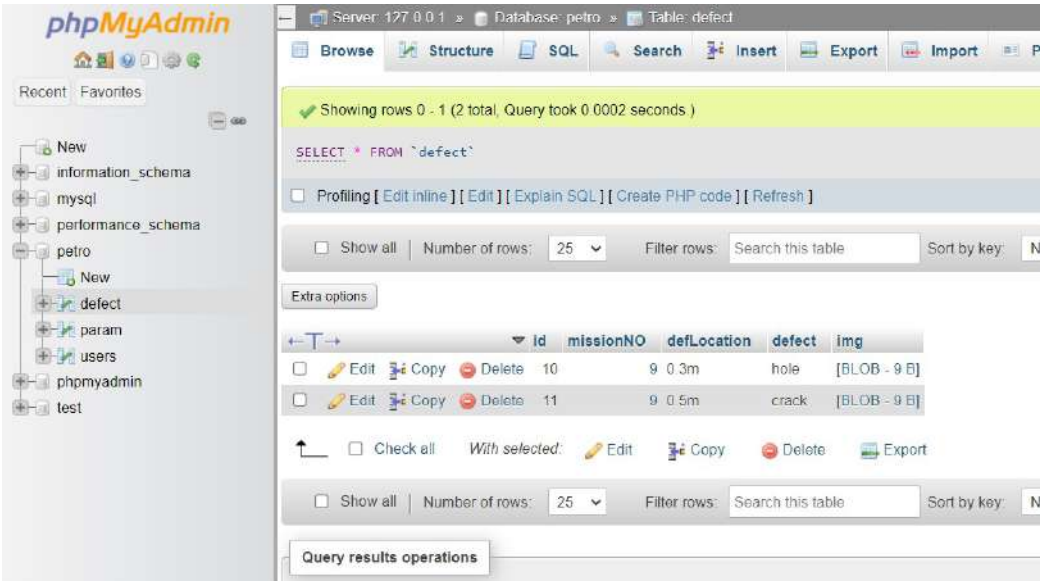


Figure 6.23. Encoder Distance Saved in The Database

6.3.4 Website Communication integration testing

As mentioned in Chapter 5, the database in our system stores images of defects captured by the camera, the locations of these defects recorded by the encoder, information about registered users, and details of the parameters.

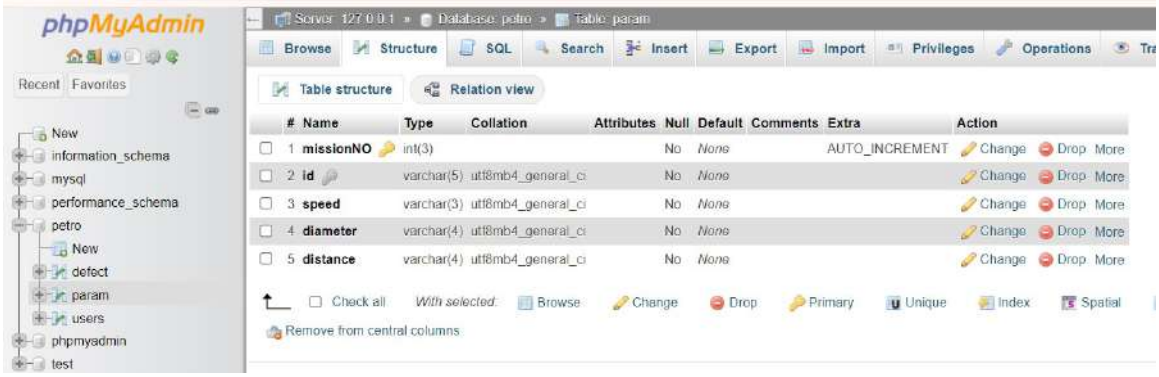


Figure 6.24. Database Table

The website features a sign-up page and a login page, enabling users to access the main dashboard. From the dashboard, users can view mission details and configure various parameters. After connecting the database to our website, we will verify the connectivity by saving an image of the defect and displaying it on the dashboard as shown in Fig. 6.25.

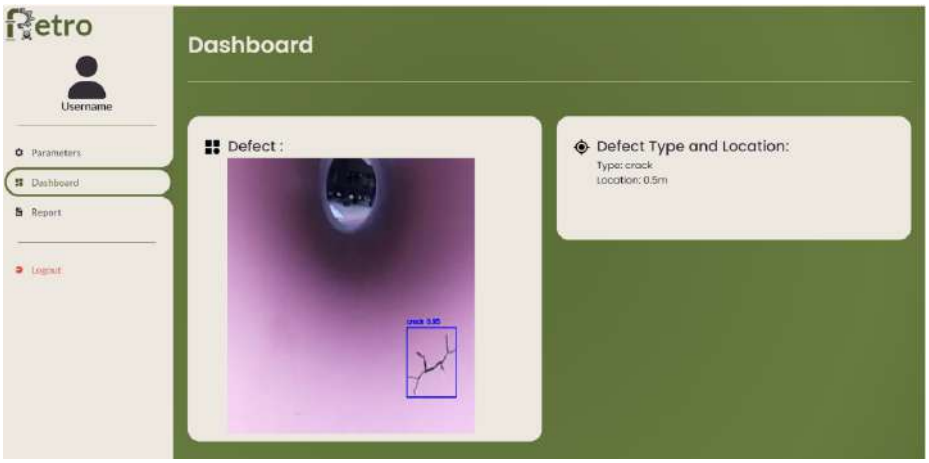


Figure 6.25. Dashboard Page

6.4 Scenarios for Validation and System Testing

System testing ensures the robot operates seamlessly, evaluating its ability to autonomously navigate pipelines, detect and classify defects, and relay real-time data. Test scenarios cover defect detection conditions, with the system displaying defect type, location, and images on the web dashboard while storing data in the database. Administrators can access the report page to review detection missions, including images and data. The robot's obstacle avoidance is also tested to ensure reliable performance in complex environments.

6.4.1 System Testing

To test the system as a whole, we will begin from the start. Initially, the admin accesses the sign-up page as shown in Fig. 6.26, where they register by selecting their role as admin and providing their name, email, ID, and password to complete a successful registration.

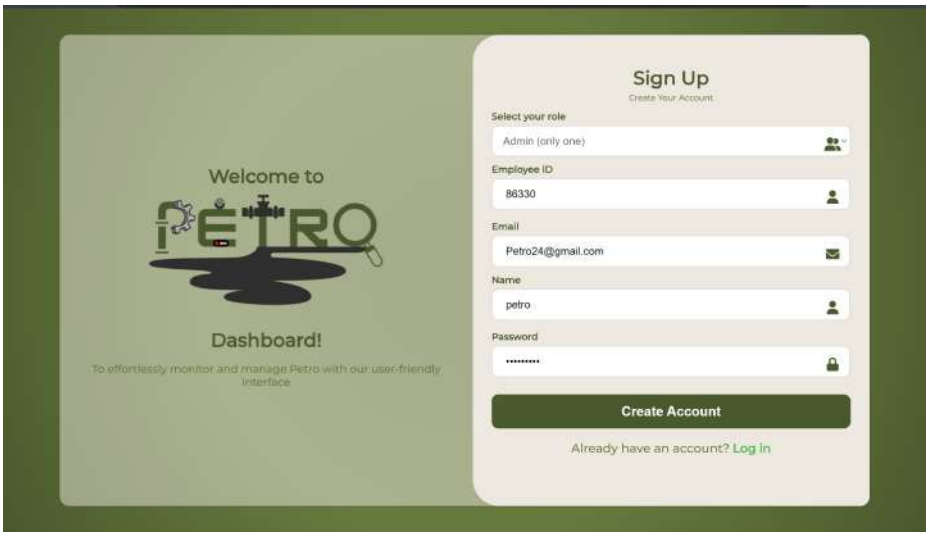


Figure 6.26. Sign up Page Registration

After creating an account as shown in Fig. 6.27 the admin will enter their name, ID, and password on the login page to access the main dashboard of the system.

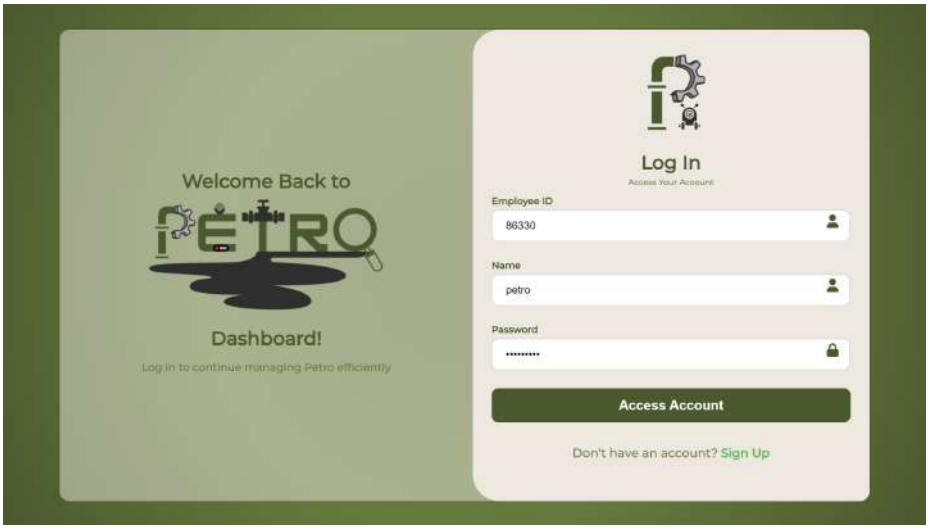


Figure 6.27. Login Page Registration

The main dashboard will appear as in Fig. 6.28, the admin can access the parameter section and enter a valid value of the pipe diameter and pipeline length to start the mission.

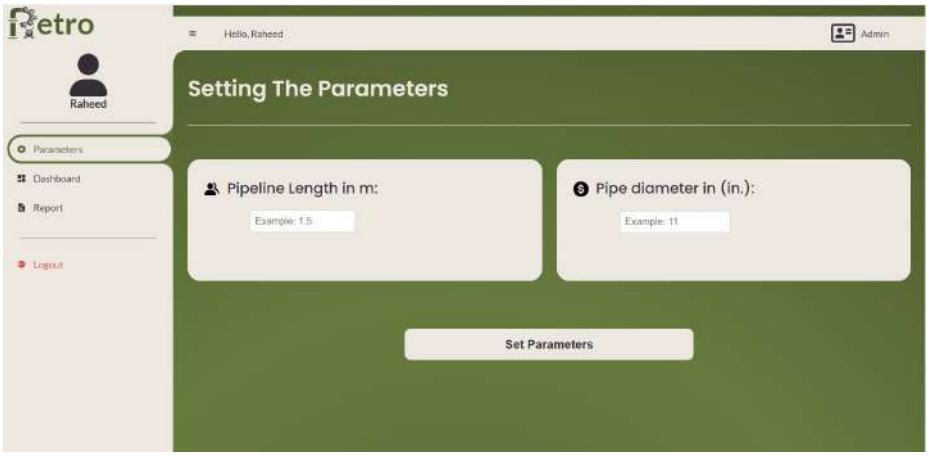


Figure 6.28. Parameter Page Information

The robot will prepare and initiate its mission within the pipe, taking small steps for precise detection. If a defect is detected, the camera will capture it as shown in Fig. 6.29

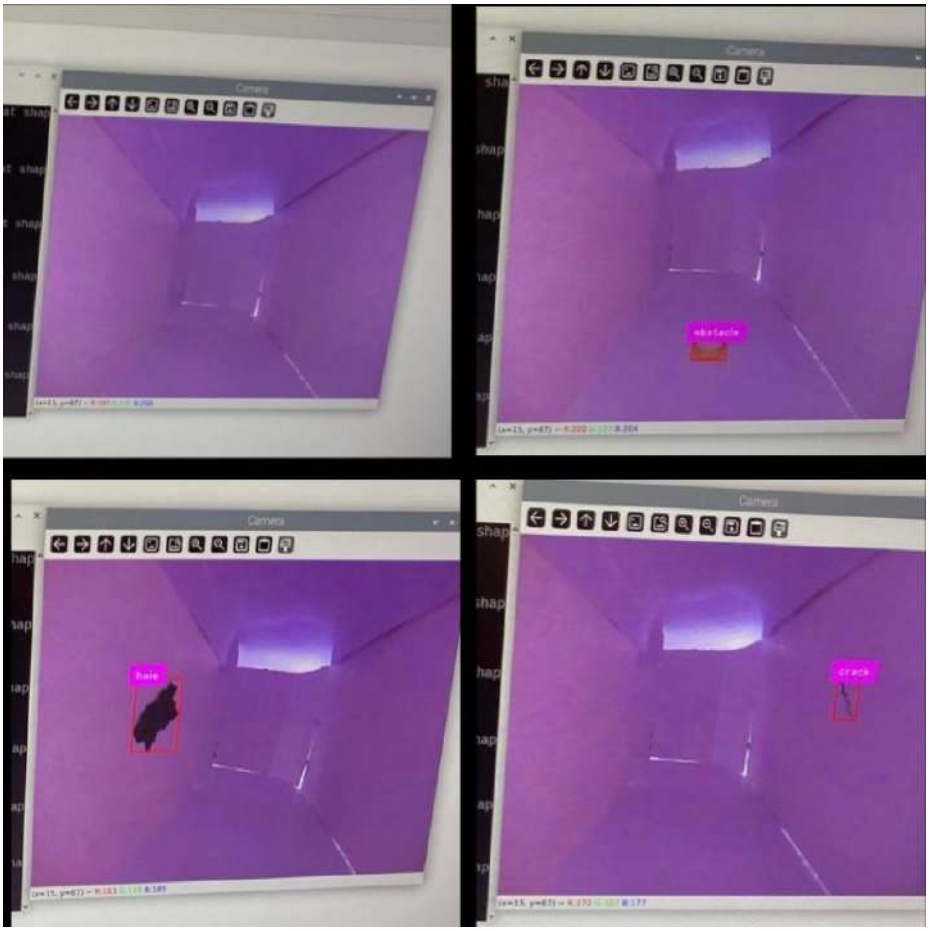


Figure 6.29. All possible defects inside the pipe

While moving the robot will always check for pipe walls or obstacles if it detects something blocking the view of the front ultrasonic sensor, if that happens the robot will stop.

Figure 6.30. The Robot Stops in Case of an Obstacle

If there are any left or right turns, the robot can decide by evaluating the readings from both the left and right ultrasonic sensors. If the reading from the right sensor is greater, the robot will turn right.

Figure 6.31. The robot is turning right

In case of defect detection, it will record the location from the encoder and display it in the dashboard as shown in Fig. 6.32.

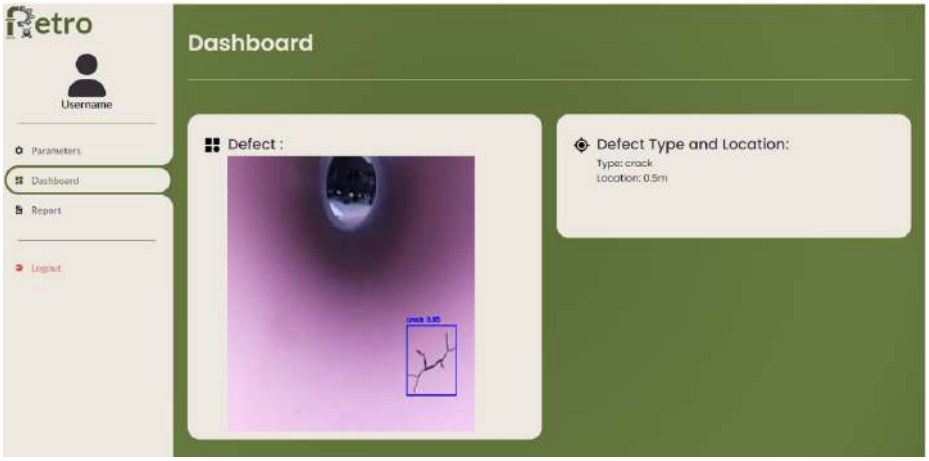


Figure 6.32. Dashboard Page Information

Upon completing the required length of the path, the system will save all defect images from the mission in the database and display them on the report page as shown in Fig. 6.33.

Figure 6.33. Report Page Information

6.5 Comparison Analysis

This section provides a comparative analysis of the Petro Autonomous In-Pipe Inspection Robot against previous research systems. The comparison focuses on five critical aspects: autonomy, localization, real-time detection, dashboard availability, and structural design. The summarized findings are presented in Table VI.V.

TABLE VI.V

PETRO COMPARISON ANALYSIS

Feature	Previous Research	Petro	Research REF
Autonomy	Limited Autonomy; robots often require significant human supervision or rely on predefined paths.	Petro is Semi-autonomous navigation with ultrasonic sensors with adaptive motion control.	• [36] • [34] • [33] • [37] • [32]
Localization	Some systems use encoders or manual methods but struggle in complex environments like T-joints or vertical sections.	Petro uses accurate localization with encoders and ultrasonic sensors, handling diverse pipeline configurations.	• [37] • [33]
Real-Time Detection	Most research focuses on detection with limited localization features	Petro uses YOLOv9t, providing both real-time detection and precise localization for immediate defect reporting.	• [34] • [33]
Dashboard Availability	Most research typically lacks a dashboard for real-time monitoring or visualization.	Petro features a real-time dashboard that provides immediate feedback on detected defects, including images, localization, and defect types (e.g., crack, hole, or obstacle).	• [34] • [33] • [32] • [18] • [19]
Structure Shape	Varying designs, often based on rectangular or cylindrical shapes, may not offer the same level of smooth movement in harsh pipeline environments	Petro robot structure features a spherical body shape, which helps facilitate smooth and easy movement within the pipeline	• [36] • [37]

6.6 Performance Analysis

Performance analysis evaluates a system's efficiency, effectiveness, and reliability. Our autonomous robot with a defect detection system measures speed, cost, accuracy, and complexity to assess how well the system meets its objectives.

- **Accuracy:** As discussed in Chapter 5, we evaluated the accuracy results for all the models we tested. YOLOv9t emerged as the most accurate model, achieving a peak accuracy of 0.69. After tuning and optimizing the performance parameters, the training process improved the accuracy to 0.736, and with the addition of a multi-scale approach, it further increased to 0.768 achieving the best results.

```
100 epochs completed in 8.837 hours.
Optimizer stripped from runs\detect\train6\weights\last.pt, 4.6MB
Optimizer stripped from runs\detect\train6\weights\best.pt, 4.6MB

Validating runs\detect\train6\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLOv9t summary (fused): 486 layers, 1,971,369 parameters, 0 gradients, 7.6 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:04<0
                   all        123        197       0.86      0.727      0.768      0.573
                 crack         46         99      0.742      0.384      0.403      0.247
                  hole         14         14      0.939      0.929      0.972       0.78
              obstacle         47         84      0.897      0.869       0.93      0.692
Speed: 0.8ms preprocess, 13.1ms inference, 0.0ms loss, 6.3ms postprocess per image
Results saved to runs\detect\train6
```

Figure 6.34. Yolov9t with accuracy details

- **Speed:** Speed is a crucial factor in enhancing performance; in our system, the YOLOv9t convolutional neural network (CNN) can perform detection and classification in approximately 1.7 seconds.

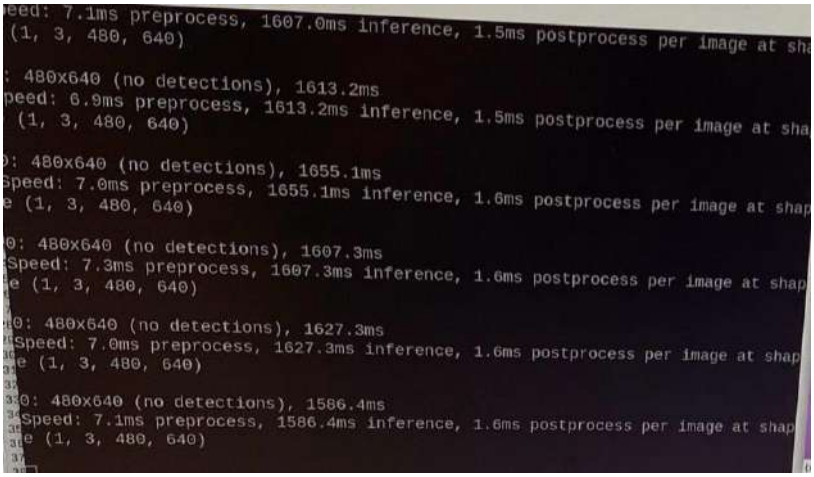


Figure 6.35. Speed of detection and classification

- **Cost:** The total cost of the hardware project was approximately 2,600 riyals, along with an additional 200 riyals for software subscriptions, you will find more detailed information in Table V.I in Chapter 5.
- **Complexity:** The system's complexity performance is influenced by its hardware and software components. On the hardware side, the integration of various sensors and actuators, along with efficient serial communication between the Raspberry Pi and Arduino, is crucial for seamless operation. These components work together to facilitate real-time data processing and control. On the software side, the implementation of the YOLOv9t convolutional neural network (CNN) enhances the system's computer vision capabilities, allowing for accurate defect detection. This advanced software not only improves the speed of processing but also elevates the overall performance of the robot in in-pipe inspections.

6.7 Conclusion

In conclusion, this chapter has examined the various testing methodologies applied to the IPIR robot to ensure its performance, accuracy, and reliability. Through unit testing, integration testing, and system validation, we have demonstrated the robot's ability to autonomously navigate pipelines, detect and classify defects, and transmit real-time data effectively. The comparison analysis highlighted key advancements over previous systems, emphasizing improvements in autonomy, localization, real-time detection, and overall system design. Additionally, the performance analysis provided insights into the efficiency, accuracy, and cost-effectiveness of the IPIR robot, reinforcing its capability to meet the desired objectives. Collectively, these results validate the robustness and effectiveness of the IPIR robot as a reliable solution for pipeline inspection.

Chapter 7

Conclusion and Future work

7.1 Introduction

This chapter will undertake an in-depth examination of the project summary detailed in Section 7.2. We will provide a comprehensive analysis of the key findings and their implications, as presented in Section 7.3. Furthermore, we will critically evaluate the limitations encountered during the research process, as outlined in Section 7.4. Finally, we will explore potential avenues for future research and development, building upon the insights and conclusions derived from this study, as discussed in Section 7.5.

7.2 Project Summary

This project presents the development of an Autonomous In-Pipe Inspection Robot (IPIR) designed to address critical challenges in underground pipeline maintenance. Traditional pipeline inspection methods, such as smart Pipeline Inspection Gauges (PIGs), are often expensive, operationally complex, and limited in real-time defect analysis capabilities. The proposed solution leverages modern advancements in artificial intelligence (AI) and autonomous navigation to offer a cost-effective and efficient alternative.

The robot features a Raspberry Pi 4 as the central processing unit, responsible for running an AI-based defect detection model built on YOLOv9t. It incorporates an OV5647 camera for visual inspection, three ultrasonic sensors, and an Arduino microcontroller equipped with an H bridge motor driver to control four DC motors for movement. Additionally, the encoder optical sensor provides accurate feedback for navigation. These components work together to ensure autonomous functionality and efficient defect detection.

The system is capable of detecting and classifying common pipeline defects, such as cracks, holes, and obstacles, in real time. A web-based dashboard enhances usability by visualizing defect type, location, and captured images, allowing the maintenance team to make informed decisions. After extensive testing, the robot demonstrated reliable defect detection and navigation, proving its effectiveness in different pipeline conditions.

Future work will focus on enhancing wireless communication for reliable data transmission, developing advanced control algorithms for better navigation, creating a system for classifying defects by severity, and upgrading the robot's processor and power capacity for improved performance.

In conclusion, this project provides a scalable and cost-effective solution for pipeline inspections, with significant potential for future improvements to increase its capabilities and applicability in broader infrastructure monitoring.

7.3 Findings

In this section, we highlight the key findings of this project, which demonstrate the effectiveness of the Autonomous In-Pipe Inspection Robot (IPIR) :

- The YOLOv9t convolutional neural network (CNN) model was identified as the optimal choice for this application, achieving a balance of precision and robustness with an accuracy of 0.768. This result is now recognized as the approved solution for the project's future development.
- The robot semi-automation system, operates with minimal human intervention beyond mission initiation. This design enables the robot to navigate pipelines independently, enhancing operational efficiency and reducing the need for manual oversight during inspection processes.
- The robot's capability to accurately identify defect locations within the pipe can significantly reduce both time and resource utilization.
- The careful selection of dataset images, reflecting expected real-world scenarios, enhanced the model's accuracy and performance in practical applications.

7.4 Limitations

In this section, we outline the limitations encountered during the development and testing of the Autonomous In-Pipe Inspection Robot (IPIR). Acknowledging these constraints is essential for understanding the scope of the project and identifying areas for future improvement. Below are the specific limitations identified:

- **Environmental Factors:**
  - Our project requires pipes exceeding 10 inches in diameter to execute the process. While we found suitable 16-inch pipes, they are only available in bulk quantities. Unfortunately, we lack the resources to purchase and transport such a large volume to the university, and individual purchases are not permitted. This limitation poses a significant obstacle to our project's implementation.
- **Data Collection:**
  - The lack of a dedicated dataset for oil and gas pipe defects hindered our project. As a workaround, we utilized a general pipe defect dataset, which required extensive cleaning, augmentation, and re-imaging to adapt it to our specific needs.
  - The restricted access to real oil and gas defect images from major companies like Aramco and SAPIC limits our ability to expand our dataset.
- **Operational limitations:**
  - The semi-automated mode necessitates initial user interaction to position the robot correctly at the center, which is essential for ensuring proper orientation during movement.

7.5 Future work

In this section, we outline potential directions for future work to enhance the capabilities of the Autonomous In-Pipe Inspection Robot (IPIR). Building on the findings and limitations identified in this project, these improvements aim to increase the robot's efficiency, adaptability, and overall performance in various pipeline inspection scenarios.

- Investigate and implement advanced wireless communication technologies, such as long-range networks for robust and reliable data transmission in challenging pipe environments.
- Develop and implement sophisticated control algorithms, such as advanced PID controllers or model predictive control, to enhance the robot's navigation within complex pipe geometries.
- Development of a robust and standardized system for classifying pipe defects based on their severity. This system will provide a consistent framework for assessing the criticality of defects, enabling more informed decisions regarding maintenance and repair priorities.
- enhance the robot's capabilities by integrating a more powerful processor and a high-capacity power bank. This combination will enable the simultaneous execution of wall-following and defect-detection algorithms.
- Incorporate additional defect categories into the system and dataset, including corrosion, erosion, and pipe displacement.

7.6 Conclusion

In conclusion, the Autonomous In-Pipe Inspection Robot (IPIR) has the potential to revolutionize pipeline maintenance by providing efficient and accurate defect detection in real-time. However, there are challenges and limitations that must be addressed to further improve its functionality and adaptability. Addressing these concerns is crucial as we move forward with future development. We have outlined a clear direction for enhancing the robot's performance, including advancements in wireless communication, navigation algorithms, and defect classification systems. This chapter serves as a summary of our current achievements and a road map for future improvements, highlighting the significant impact of this work while maintaining a focus on scalability and practical application.

Chapter 8

Appendix

8.1 Model Selection Codes

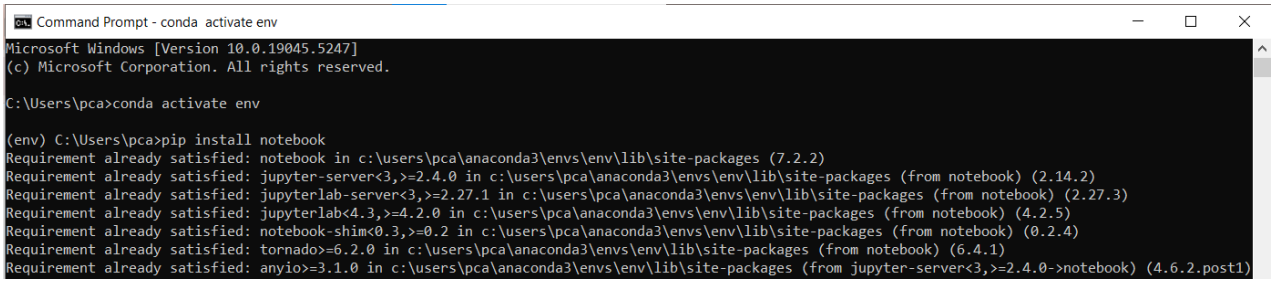


Figure 8.1. pip install notebook

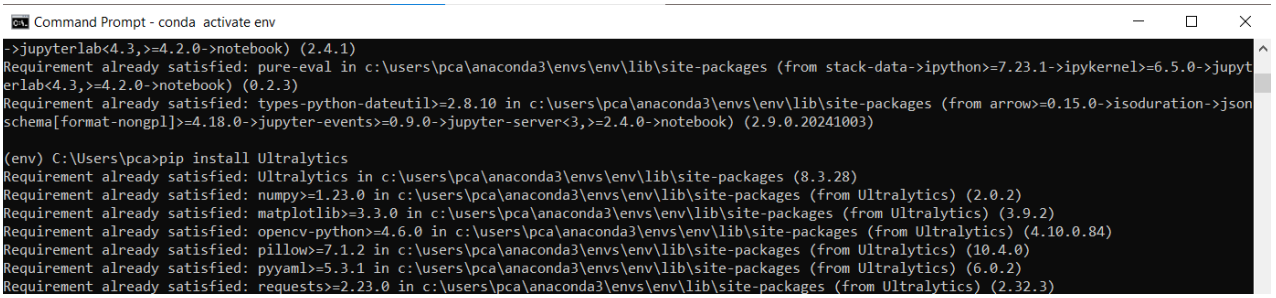


Figure 8.2. pip install ultralytics

8.1.1 YOLOv5n

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov5n.pt")

# Train the model
train_results = model.train(
    data="C:/Users/pca/Desktop/yolo5/yolov5Dataset/data.yaml",  # path to dataset YAML
    epochs=5,  # number of training epochs
    imgsz=640,  # training image size
    device=0,  # device to run on
)
```

Figure 8.3. yolov5n

```
5 epochs completed in 0.072 hours.
Optimizer stripped from runs\detect\train\weights\last.pt, 5.3MB
Optimizer stripped from runs\detect\train\weights\best.pt, 5.3MB

Validating runs\detect\train\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLOv5n summary (fused): 193 layers, 2,503,529 parameters, 0 gradients, 7.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<0
                   all        114        186      0.739      0.657      0.699       0.47
                 crack         44         97      0.522      0.278       0.32      0.187
                  hole         14         14          1      0.906      0.952      0.675
              obstacle         38         75      0.695      0.787      0.824      0.547
Speed: 0.7ms preprocess, 10.0ms inference, 0.0ms loss, 3.1ms postprocess per image
Results saved to runs\detect\train
```

Figure 8.4. yolov5n-map

8.1.2 YOLOv8s

```
# @title Custom Training
%cd {HOME}

!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml
epochs=5
batch=16
imgsz=640 plots=True
```

Figure 8.5. yolov8s

```
Validating runs/detect/train2/weights/best.pt...
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11126745 parameters, 0 gradients, 28.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:02<00:00,  1.69it/s]
                   all        114        186      0.747       0.55      0.643      0.432
                 crack        114         97       0.55      0.247      0.262      0.143
                  hole        114         14      0.902      0.643       0.86      0.616
              obstacle        114         75      0.789       0.76      0.807      0.538
Speed: 0.2ms preprocess, 5.6ms inference, 0.0ms loss, 4.2ms postprocess per image
Results saved to runs/detect/train2
💡 Learn more at https://docs.ultralytics.com/modes/train
```

Figure 8.6. yolov8s-map

8.1.3 YOLOv9t

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov9t.pt")

# Train the model
train_results = model.train(
    data="C:/Users/pca/Desktop/yolo9/yolov9Dataset/data.yaml",  # path to dataset YAML
    epochs=5,  # number of training epochs
    imgsz=640,  # training image size
    device=0,  # device to run on
)
```

Figure 8.7. yolov9t

```
Validating runs\detect\train3\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLOv9t summary (fused): 486 layers, 1,971,369 parameters, 0 gradients, 7.6 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:03<0
                   all        123        197      0.676      0.671       0.69      0.495
                 crack         46         99      0.668      0.303      0.364       0.22
                  hole         14         14      0.625      0.857      0.845      0.659
              obstacle         47         84      0.734      0.853      0.861      0.606
Speed: 1.2ms preprocess, 16.6ms inference, 0.0ms loss, 2.3ms postprocess per image
Results saved to runs\detect\train3
```

Figure 8.8. yolov9t-map

8.1.4 YOLOv10n

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov10n.pt")

# Train the model
train_results = model.train(
    data="C:/Users/pca/Desktop/yolo10/yolo10Dataset/data.yaml",  # path to dataset YAML
    epochs=5,  # number of training epochs
    imgsz=640,  # training image size
    device=0,  # device to run on, i.e. device=0 or device=0,1,2,3 or device=cpu
)
```

Figure 8.9. yolov10n

```
5 epochs completed in 0.092 hours.
Optimizer stripped from runs\detect\train2\weights\last.pt, 5.7MB
Optimizer stripped from runs\detect\train2\weights\best.pt, 5.7MB

Validating runs\detect\train2\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLOv10n summary (fused): 285 layers, 2,695,586 parameters, 0 gradients, 8.2 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<0
                   all        114        186      0.521      0.525      0.524      0.351
                 crack         44         97       0.38      0.151      0.191      0.123
                  hole         14         14      0.531      0.729       0.69      0.515
              obstacle         38         75      0.653      0.693      0.691      0.413
Speed: 0.5ms preprocess, 11.7ms inference, 0.0ms loss, 0.4ms postprocess per image
Results saved to runs\detect\train2
```

Figure 8.10. yolov10n-map

8.1.5 YOLOv11n

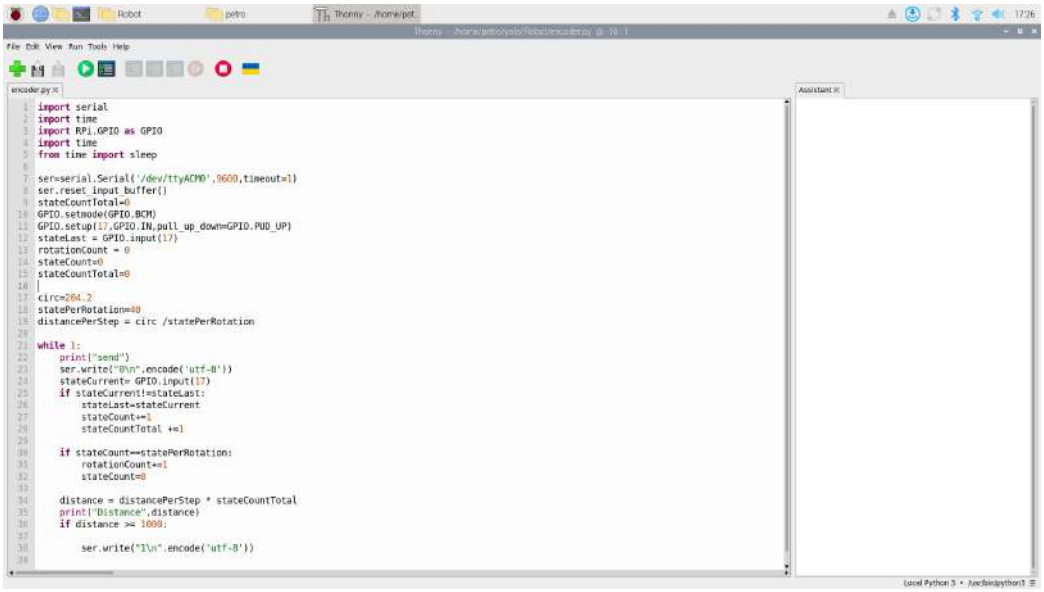
```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolo11n.pt")

# Train the model
train_results = model.train(
    data="C:/Users/pca/Desktop/yolo11/yolo11Dataset/data.yaml",  # path to dataset YAML
    epochs=5,  # number of training epochs
    imgsz=640,  # training image size
    device=0,  # device to run on, i.e. device=0 or device=0,1,2,3 or device=cpu
)
```

Figure 8.11. yolov11n

```
100 epochs completed in 1.465 hours.
Optimizer stripped from runs\detect\train\weights\last.pt, 5.5MB
Optimizer stripped from runs\detect\train\weights\best.pt, 5.5MB

Validating runs\detect\train\weights\best.pt...
Ultralytics 8.3.28  Python-3.12.7 torch-2.5.1 CUDA:0 (NVIDIA GeForce GTX 1060, 6144MiB)
YOLO11n summary (fused): 238 layers, 2,582,737 parameters, 0 gradients, 6.3 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<0
                   all        114        186      0.816      0.707      0.728      0.526
                 crack         44         97      0.628      0.366      0.375      0.216
                  hole         14         14      0.974      0.929      0.934      0.739
              obstacle         38         75      0.845      0.827      0.874      0.624
Speed: 0.9ms preprocess, 9.7ms inference, 0.0ms loss, 2.4ms postprocess per image
Results saved to runs\detect\train
```

Figure 8.12. yolov11n-map

8.1.6 Robot Navigation and Localization



```
import serial
import time
import RPi.GPIO as GPIO
import time
from time import sleep

ser=serial.Serial('/dev/ttyACM0',9600,timeout=1)
ser.reset_input_buffer()
stateCountTotal=0
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
stateLast = GPIO.input(17)
rotationCount = 0
stateCount=0
stateCountTotal=0

circ=264.2
statePerRotation=40
distancePerStep = circ /statePerRotation

while 1:
    print("send")
    ser.write("0\n".encode('utf-8'))
    stateCurrent= GPIO.input(17)
    if stateCurrent!=stateLast:
        stateLast=stateCurrent
        stateCount+=1
        stateCountTotal +=1

    if stateCount==statePerRotation:
        rotationCount+=1
        stateCount=0

    distance = distancePerStep * stateCountTotal
    print("Distance",distance)
    if distance >= 1000:

        ser.write("1\n".encode('utf-8'))
```

Figure 8.13. Code for encoder in RPI

```
81  void left() {
82  analogWrite(enA, 255);
83  analogWrite(enB, 255);
84    digitalWrite(in1, LOW);
85    digitalWrite(in2, HIGH);
86    digitalWrite(in3, HIGH);
87    digitalWrite(in4, LOW);
88  }
89  void right() {
90  analogWrite(enA, 255);
91  analogWrite(enB, 255);
92    digitalWrite(in1, HIGH);
93    digitalWrite(in2, LOW);
94    digitalWrite(in3, LOW);
95    digitalWrite(in4, HIGH);
96  }
97  void Stop() {
98  analogWrite(enA, 0);
99  analogWrite(enB, 0);
100   digitalWrite(in1, LOW);
101   digitalWrite(in2, LOW);
102   digitalWrite(in3, LOW);
103   digitalWrite(in4, LOW);
104 }
105 void forward(){
106 analogWrite(enA, 150);
107 analogWrite(enB, 150);
108   digitalWrite(in1, HIGH);
109   digitalWrite(in2, LOW);
110   digitalWrite(in3, HIGH);
111   digitalWrite(in4, LOW);
112 }
113 void backward(){
114 analogWrite(enA, 200);
115 analogWrite(enB, 200);
116   digitalWrite(in1, LOW);
```

Figure 8.14. Functions to control motors speed and direction

```
void loop() {
 if  (Serial.available()>0){
    String a = Serial.readStringUntil('\n');
    if( a == "0")
    {
int frontSensor = sensorFront();
if(frontSensor <= threshold){
int leftSensor = sensorLeft();
int rightSensor = sensorRight();
delay(1000);
if(rightSensor>leftSensor){
    right();
    delay(690);
  }
else if(leftSensor>rightSensor){
  left();
  delay(780);
  }
  else{
    Stop();
  }
}
 else{
   frontSensor = sensorFront();
  forward();
  delay(300);
  Stop();
if(frontSensor <= threshold){
  return;
}
else{
  Stop();
  delay(3000);
}
 }
    }else if ( a == "1"){
  Stop();
  delay(5000);
 }
  }
    else {
    Stop();
    }
}
```

Figure 8.15. Code for navigation with localization

8.1.7 Sign up page

signup.php

```
petro > signup.php
  3   <head>
  8       <style>
235       </style>
236   </head>
237   <body>
238       <div class="container">
239           <div class="left-panel">
240               <h1>Welcome to</h1>
241               <img src="petro.png" width="350" height="150">
242               <h1>Dashboard!</h1>
243               <p>To effortlessly monitor and manage Petro with our user-friendly interface</p>
244           </div>
245           <div class="right-panel">
246               <div class="form-container">
247                   <h1>Sign Up</h1>
248                   <p class="create">Create Your Account</p>
249                   <?php if(isset($_GET['error'])) { ?>
250                       <p class="error"><?php echo $_GET['error']; ?> </p>
251                   <?php } ?>
252                   <form action="insert.php" method="post">
253                       <div class="form-group">
254                           <label for="role">Select your role</label>
255                           <select id="role" name="role" required>
256                               <option value="" disabled selected>Select your role</option>
257                               <option value="admin">Admin (only one)</option>
258                               <option value="supervisor">Supervisor</option>
259                           </select>
260                           <i class="fas fa-user-group" style="color: #4b592e;"></i>
261                       </div>
262                       <div class="form-group">
263                           <label for="id">Employee ID</label>
264                           <input type="text" name="id" id="id" placeholder="Enter your employee ID" required>
265                           <i class="fas fa-user" style="color: #4b592e;"></i>
266                       </div>
267                       <p class="id-error" id="id-error"></p>
268                       <div class="form-group">
269                           <label for="email">Email</label>
```

Figure 8.16. HTML Code sign up page

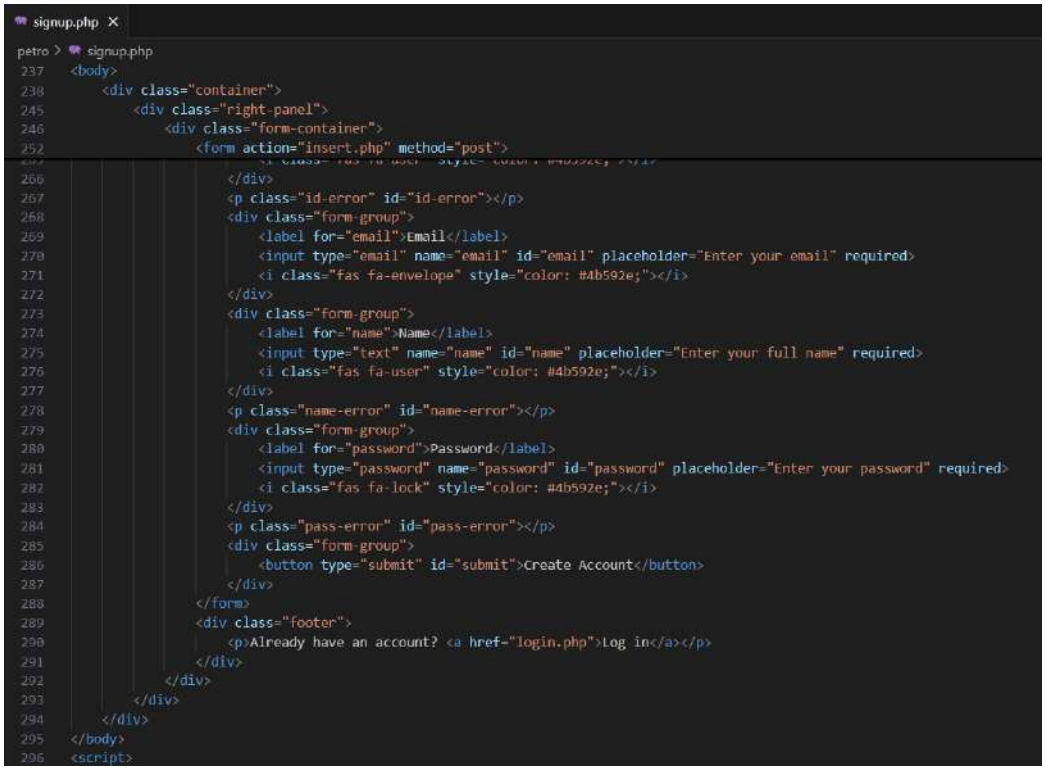


Figure 8.17. HTML Code sign up page

signup.php

petro > signup.php

```
295  </body>
296  <script>
297      // Get references to form elements
298      const pass = document.getElementById("password");
299      const btn = document.getElementById("submit");
300      const id = document.getElementById("id");
301      const name = document.getElementById("name");
302
303      // Function to check if a string contains only digits
304      function containsOnlyDigits(str) {
305          return /^\d+$/.test(str);
306      }
307
308      // Password validation on blur event
309      pass.addEventListener('blur', () => {
310          const error = document.getElementById('pass-error');
311          if (pass.value?.length < 8 || pass.value?.length > 20) {
312              btn.disabled = true;
313              btn.style.cursor = "not-allowed";
314              error.innerHTML = "Password must be between 8 and 20 characters";
315              return; // Corrected from returnb
316          }
317          error.innerHTML = "";
318          btn.disabled = false;
319          btn.style.cursor = "auto";
320      });
321
```

Figure 8.18. JavaScript Code sign up page

signup.php

petro > signup.php

```
296   <script>
309       pass.addEventListener( blur , () => {
320       });
321
322       // Employee ID validation on blur event
323       id.addEventListener('blur', () => {
324           const error = document.getElementById('id-error');
325           if (id.value?.length !== 5 || !containsOnlyDigits(id.value)) {
326               btn.disabled = true;
327               btn.style.cursor = "not-allowed";
328               error.innerHTML = "ID must be exactly 5 digits long";
329               return; // Corrected from returnb
330           }
331           error.innerHTML = "";
332           btn.disabled = false;
333           btn.style.cursor = "auto";
334       });
335
336       // Name validation on blur event
337       name.addEventListener('blur', () => {
338           const error = document.getElementById('name-error');
339           if (name.value?.length > 20) {
340               btn.disabled = true;
341               btn.style.cursor = "not-allowed";
342               error.innerHTML = "Name must be 20 characters or fewer";
343               return; // Corrected from returnb
344           }
345           error.innerHTML = "";
346           btn.disabled = false;
347           btn.style.cursor = "auto";
348       });
349   </script>
350   </html>
```

Figure 8.19. JavaScript Code sign up page

signup.php insert.php

petro > insert.php

```php
<?php
// Check if the request method is POST
if($_SERVER["REQUEST_METHOD"] == "POST"){

  try {
    // Get form data from POST request
    $id = $_POST["id"];
    $role = $_POST["role"];
    $username = $_POST["name"];
    $password = $_POST["password"];
    $email = $_POST["email"];
    $errorMsg = "";

    // Validate Employee ID: must be exactly 5 digits
    if(strlen($id) != 5 || !ctype_digit($id)){
      header("Location:signup.php?error=ID must be exactly 5 digits long");
      exit(); // Stop script execution
    }

    // Validate Name: must be 20 characters or fewer
    if(strlen($username) > 20){
      header("Location:signup.php?error=Name must be 20 characters or fewer");
      exit();
    }

    // Validate Password: must be at least 8 characters long
    if(strlen($password) < 8){
      header("Location:signup.php?error=Password must be at least 8 characters long");
      exit();
    }
    // Validate Password: must be 20 characters or fewer
    elseif(strlen($password) > 20){
      header("Location:signup.php?error=Password must be 20 characters or fewer");
      exit();
    }

```

Figure 8.20. PHP Code sign up page

signup.php insert.php ×

petro > insert.php

```php
if($_SERVER["REQUEST_METHOD"] == "POST"){
  try {
    // Connect to the database
    require_once "connect.php";

    // Check if an admin user already exists
    $queryselect = "SELECT * FROM users WHERE role = 'admin';";
    $stmtone = $pdo->prepare($queryselect);
    $stmtone->execute();
    $results = $stmtone->fetchAll(PDO::FETCH_ASSOC);

    // Allow only one admin; if none exists or if the role is 'supervisor'
    if(empty($results) || $role == 'supervisor'){
      // Check if the provided ID or email already exists
      $queryselectID = "SELECT id, email FROM users WHERE id = :id OR email = :email;";
      $stmtTWO = $pdo->prepare($queryselectID);
      $stmtTWO->bindParam(":id", $id);
      $stmtTWO->bindParam(":email", $email);
      $stmtTWO->execute();
      $resultsid = $stmtTWO->fetchAll(PDO::FETCH_ASSOC);

      // If the ID and email do not exist, insert new user data
      if(empty($resultsid)){
        // Query to insert user data
        $queryinsert = "INSERT INTO users (id, role, username, password, email) VALUES (:id, :role, :username, :password, :email);";
        // Prevent SQL injection
        $stmt = $pdo->prepare($queryinsert);
        $stmt->bindParam(":id", $id);
        $stmt->bindParam(":role", $role);
        $stmt->bindParam(":username", $username);
        $stmt->bindParam(":password", $password);
        $stmt->bindParam(":email", $email);
        $stmt->execute();
```

Figure 8.21. PHP Code sign up page

signup.php insert.php

petro > insert.php

```
3   if($_SERVER["REQUEST_METHOD"] == "POST"){
5     try {
47      if(empty($results) || $role == 'supervisor'){
57        if(empty($resultsid)){
67          $stmt->execute();
68
69          // Redirect to login page upon successful registration
70          header("Location:login.php");
71        } else {
72          // Redirect back to signup with an error if user already exists
73          header("Location:signup.php?error=This user already exists");
74        }
75      } else {
76        // Redirect back to signup if an admin already exists
77        header("Location:signup.php?error=An admin already exists");
78      }
79
80      // Close the database connection
81      $pdo = null;
82      $stmtone = null;
83
84      // Terminate the script
85      die();
86    } catch(PDOException $e) {
87      // Handle any database errors
88      die("Query failed: " . $e->getMessage());
89    }
90  } else {
91    // Redirect to signup if request method is not POST
92    header("Location:signup.php");
93  }
```

Figure 8.22. PHP Code sign up page

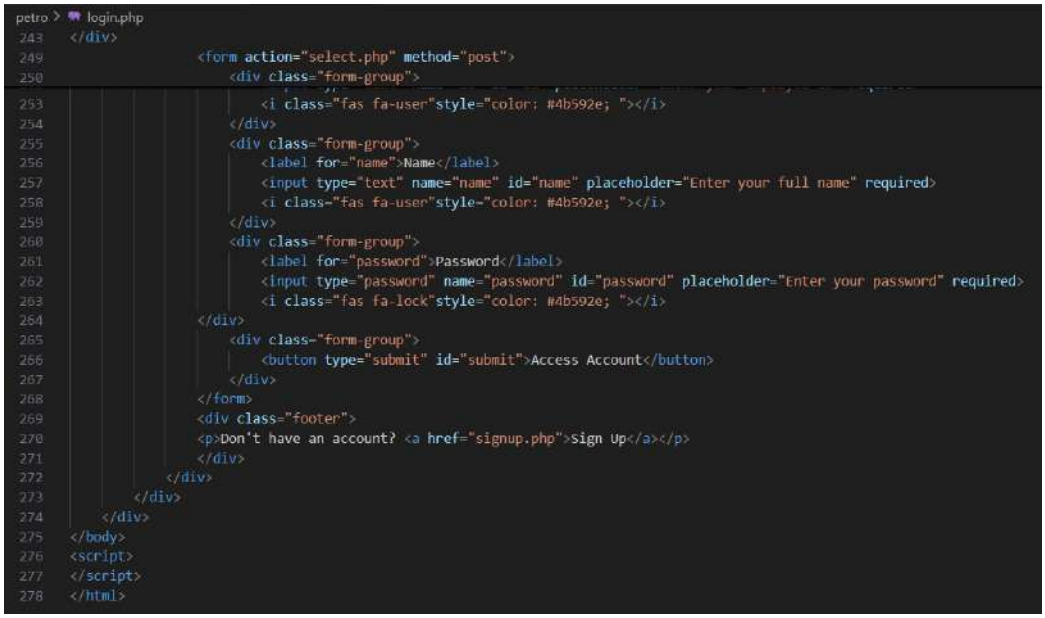
8.1.8 Log in page

signup.php insert.php login.php

petro > login.php

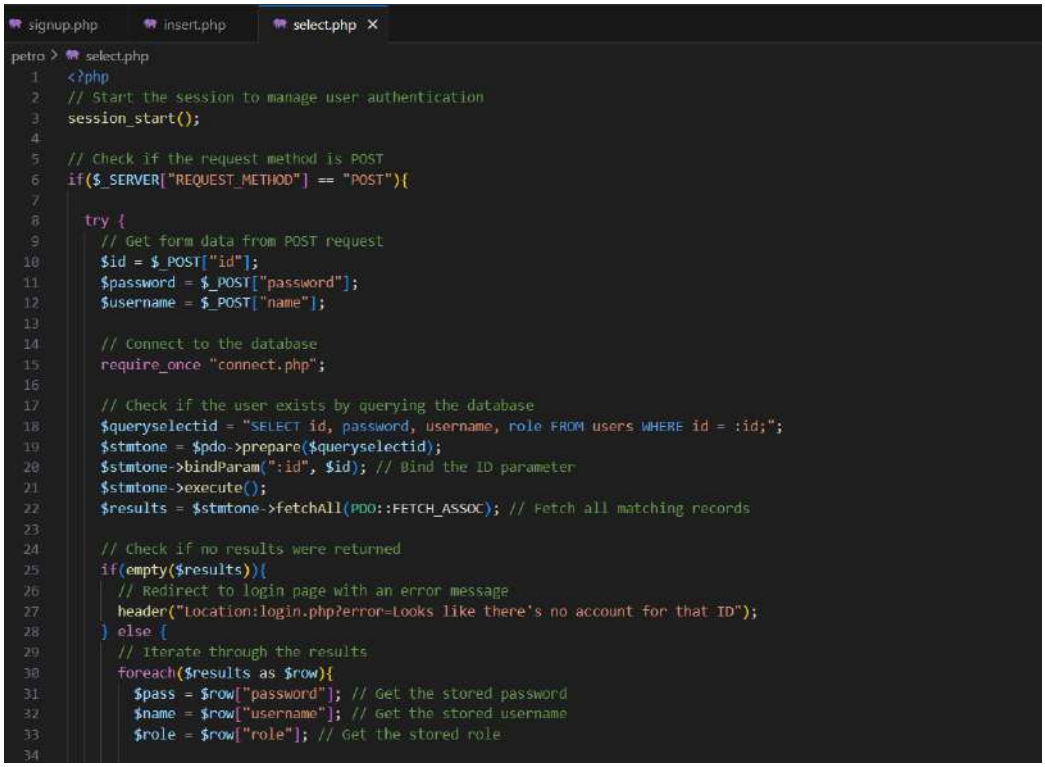
```
228     }
229         </style>
230     </head>
231     <body>
232         <div class="container">
233             <div class="left-panel">
234                 <h1>Welcome Back to</h1>
235                 <img src="petro.png" width="350" height="150">
236                 <h1>Dashboard!</h1>
237                 <p>Log in to continue managing Petro efficiently</p>
238             </div>
239             <div class="right-panel">
240                 <div class="form-container">
241                     <div class="icon-container">
242                     <img src="P-logo.png" width="100" height="50">
243     </div>
244                     <h1>Log In</h1>
245                     <p class ="create">Access Your Account</p>
246                     <?php if(isset($_GET['error'])) { ?>
247                         <p class="error" ><?php echo $_GET['error']; ?> </p>
248                         <?php } ?>
249                     <form action="select.php" method="post">
250                         <div class="form-group">
251                             <label for="id">Employee ID</label>
252                             <input type="text" name="id" id="id" placeholder="Enter your employee ID" required>
253                             <i class="fas fa-user"style="color: #4b592e; "></i>
254                         </div>
255                         <div class="form-group">
256                             <label for="name">Name</label>
257                             <input type="text" name="name" id="name" placeholder="Enter your full name" required>
258                             <i class="fas fa-user"style="color: #4b592e; "></i>
259                         </div>
260                         <div class="form-group">
261                             <label for="password">Password</label>
262                             <input type="password" name="password" id="password" placeholder="Enter your password" required>
263                             <i class="fas fa-lock"style="color: #4b592e; "></i>
264                     </div>
```

Figure 8.23. HTML Code log in page



```
petro > login.php
243   </div>
249                 <form action="select.php" method="post">
250                     <div class="form-group">
253                         <i class="fas fa-user"style="color: #4b592e; "></i>
254                     </div>
255                     <div class="form-group">
256                         <label for="name">Name</label>
257                         <input type="text" name="name" id="name" placeholder="Enter your full name" required>
258                         <i class="fas fa-user"style="color: #4b592e; "></i>
259                     </div>
260                     <div class="form-group">
261                         <label for="password">Password</label>
262                         <input type="password" name="password" id="password" placeholder="Enter your password" required>
263                         <i class="fas fa-lock"style="color: #4b592e; "></i>
264                 </div>
265                     <div class="form-group">
266                         <button type="submit" id="submit">Access Account</button>
267                     </div>
268                 </form>
269                 <div class="footer">
270                 <p>Don't have an account? <a href="signup.php">Sign Up</a></p>
271                 </div>
272             </div>
273         </div>
274     </div>
275 </body>
276 <script>
277 </script>
278 </html>
```

Figure 8.24. HTML Code log in page



```
signup.php   insert.php   select.php ×
petro > select.php
1   <?php
2   // Start the session to manage user authentication
3   session_start();
4
5   // Check if the request method is POST
6   if($_SERVER["REQUEST_METHOD"] == "POST"){
7
8     try {
9       // Get form data from POST request
10      $id = $_POST["id"];
11      $password = $_POST["password"];
12      $username = $_POST["name"];
13
14      // Connect to the database
15      require_once "connect.php";
16
17      // Check if the user exists by querying the database
18      $queryselectid = "SELECT id, password, username, role FROM users WHERE id = :id;";
19      $stmtone = $pdo->prepare($queryselectid);
20      $stmtone->bindParam(":id", $id); // Bind the ID parameter
21      $stmtone->execute();
22      $results = $stmtone->fetchAll(PDO::FETCH_ASSOC); // Fetch all matching records
23
24      // Check if no results were returned
25      if(empty($results)){
26        // Redirect to login page with an error message
27        header("Location:login.php?error=Looks like there's no account for that ID");
28      } else {
29        // Iterate through the results
30        foreach($results as $row){
31          $pass = $row["password"]; // Get the stored password
32          $name = $row["username"]; // Get the stored username
33          $role = $row["role"]; // Get the stored role
34
```

Figure 8.25. PHP Code log in page

signup.php insert.php select.php ×

petro > select.php

```php
if($_SERVER["REQUEST_METHOD"] == "POST"){
  try {
    } else {
      // Iterate through the results
      foreach($results as $row){
        $pass = $row["password"]; // Get the stored password
        $name = $row["username"]; // Get the stored username
        $role = $row["role"]; // Get the stored role
        // Check if the provided password and username match the stored values
        if($pass == $password && $username == $name){
          // Set session variables for logged-in user
          $_SESSION["loggedin"] = true;
          $_SESSION["username"] = $name;
          $_SESSION["role"] = $role;
          $_SESSION["id"] = $id;
          // Redirect to the main page upon successful login
          header("Location:main.php");
        } else {
          // Redirect back to login with an error if credentials are incorrect
          header("Location:login.php?error=Oops! Wrong password or name");
        }
      }
    }
    // Close the database connection
    $pdo = null;
    $stmtone = null;

    // Terminate the script
    die();
  } catch(PDOException $e) {
    // Handle any database errors
    die("Query failed: " . $e->getMessage());
  }
} else {
  // Redirect to signup page if request method is not POST
  header("Location:signup.php");
}
```

Figure 8.26. PHP Code log in page

8.1.9 Main page

signup.php insert.php select.php main.php ×

petro > main.php

```php
<?php
// Start the session to manage user authentication
session_start();

// Output session variables for debugging purposes
echo $_SESSION["loggedin"];
echo $_SESSION["username"];

// Store the user's role in a variable
$role = $_SESSION["role"];

// Check if the user is logged in; if not, redirect to the login page
if(!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] != true){
    header("location: login.php");
    exit; // Stop script execution
}

?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Link to Boxicons for icons -->
    <link href='https://unpkg.com/boxicons@2.0.9/css/boxicons.min.css' rel='stylesheet'>
    <!-- Link to custom CSS -->
    <link rel="stylesheet" href="style.css">
    <!-- Link to Font Awesome for additional icons -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
    <!-- Link to Google Fonts for Material Symbols -->
    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@40,200,0,200
    <title>Petro</title>
</head>
<body>
```

Figure 8.27. HTML and PHP Code main page

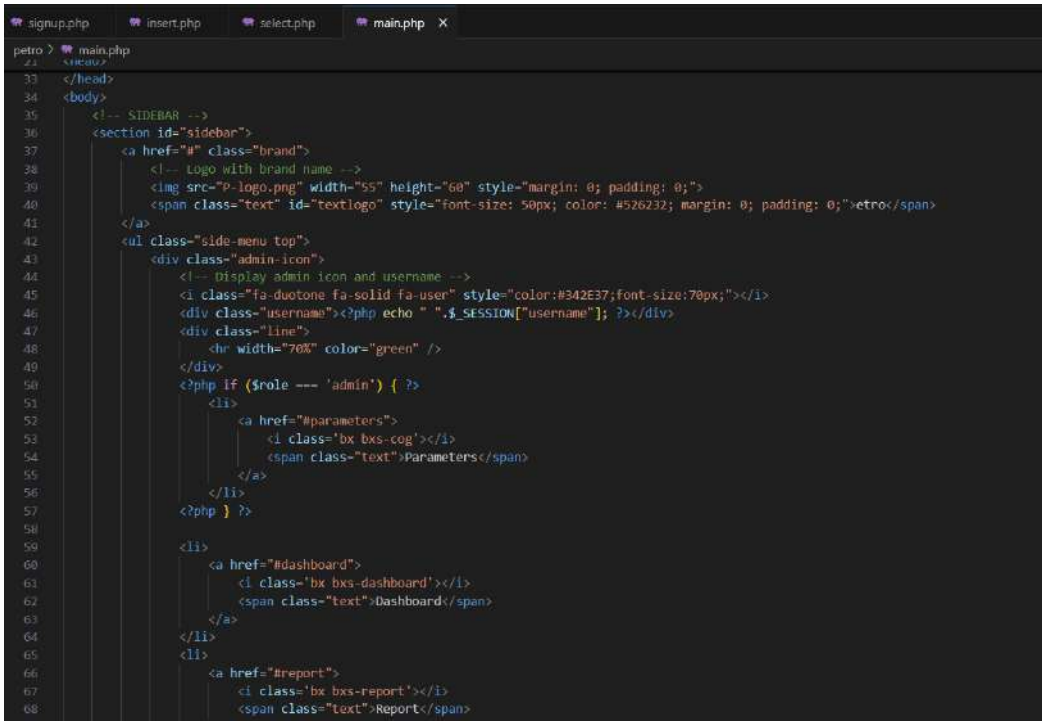


Figure 8.28. HTML and PHP Code main page

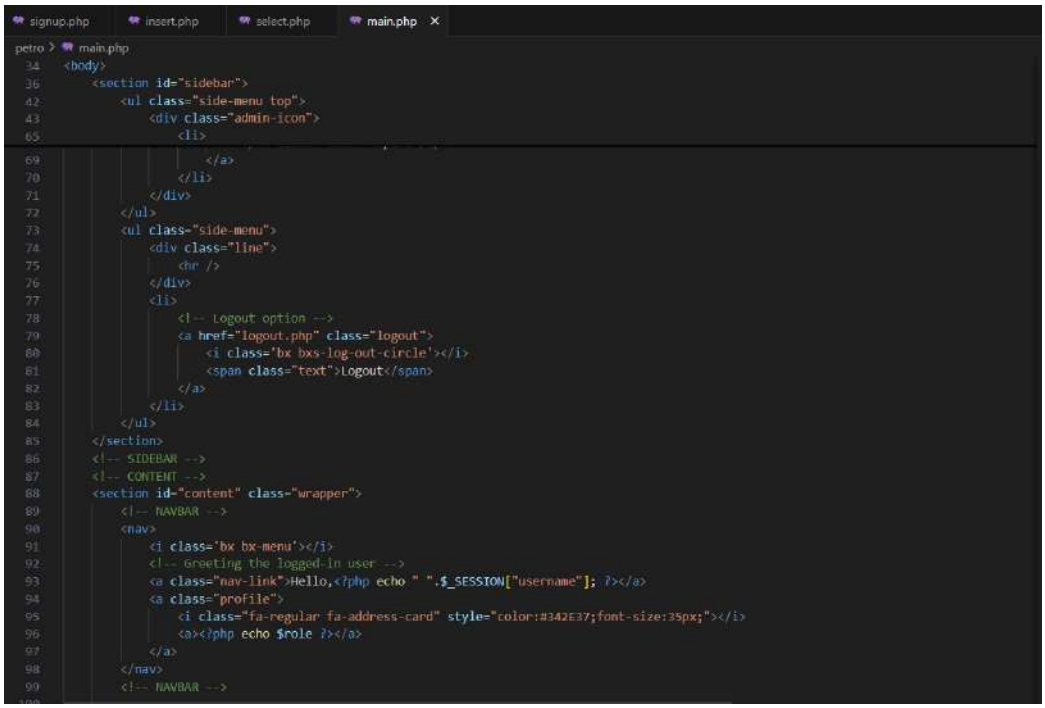


Figure 8.29. HTML and PHP Code main page

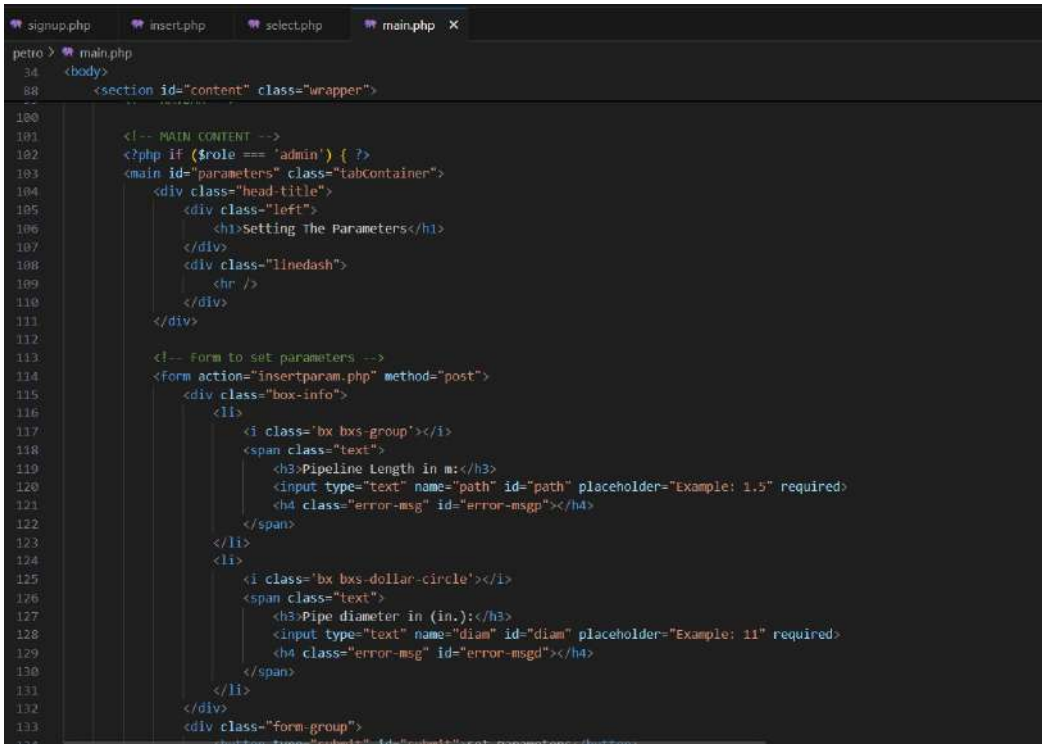
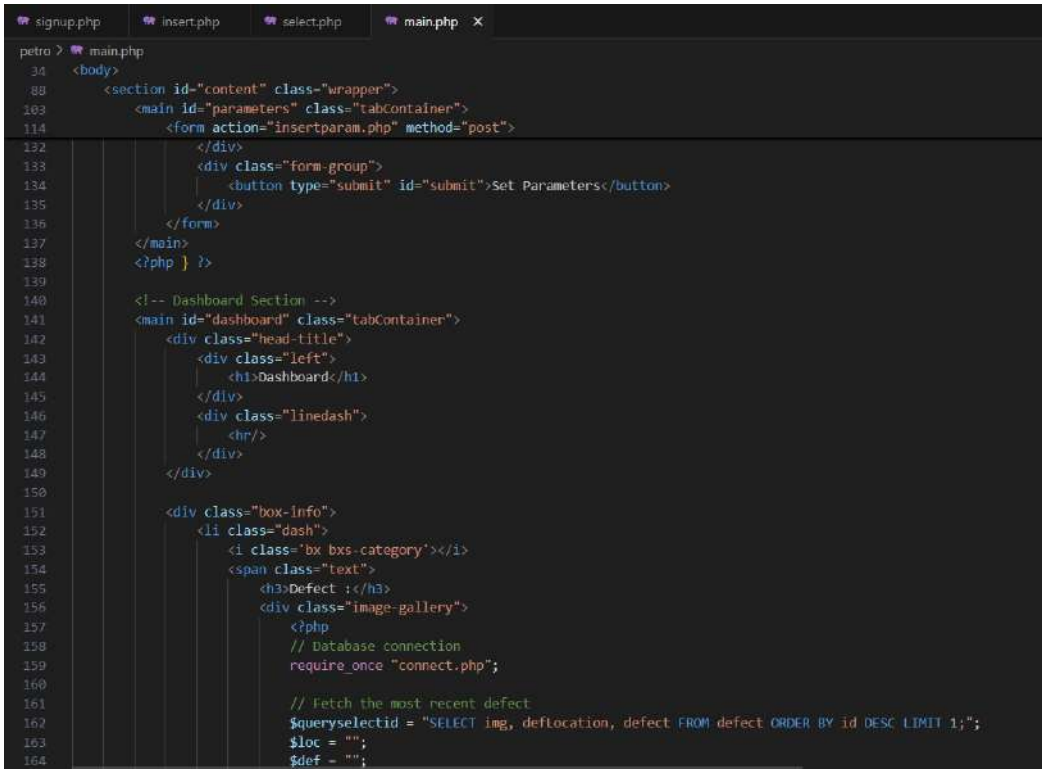
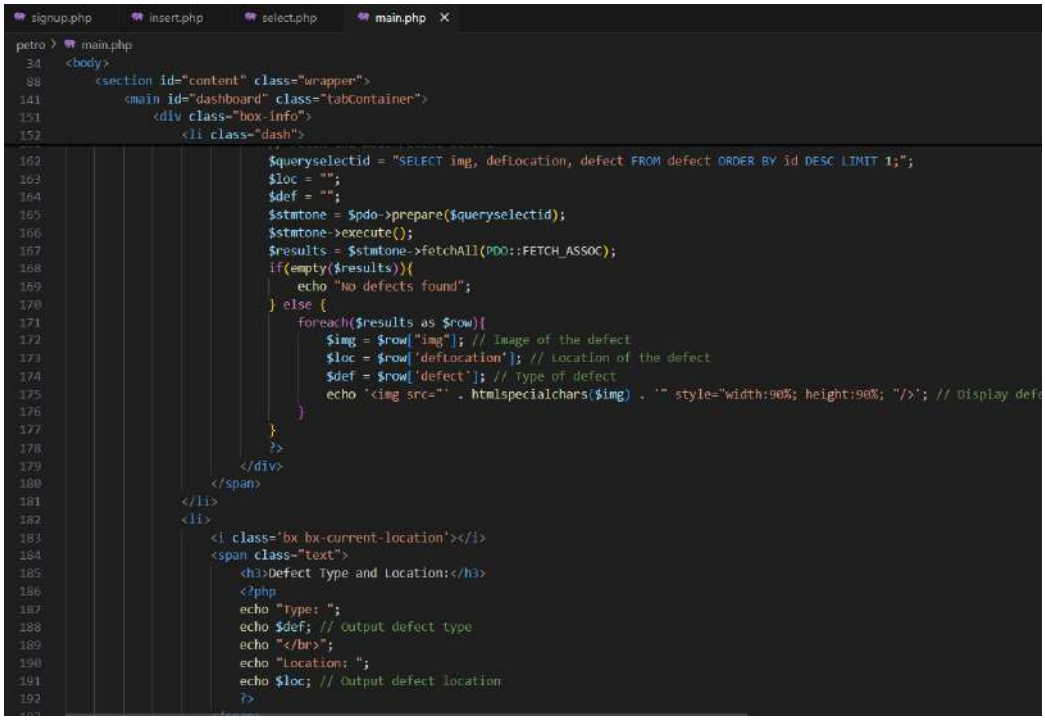


Figure 8.30. HTML and PHP Code main page



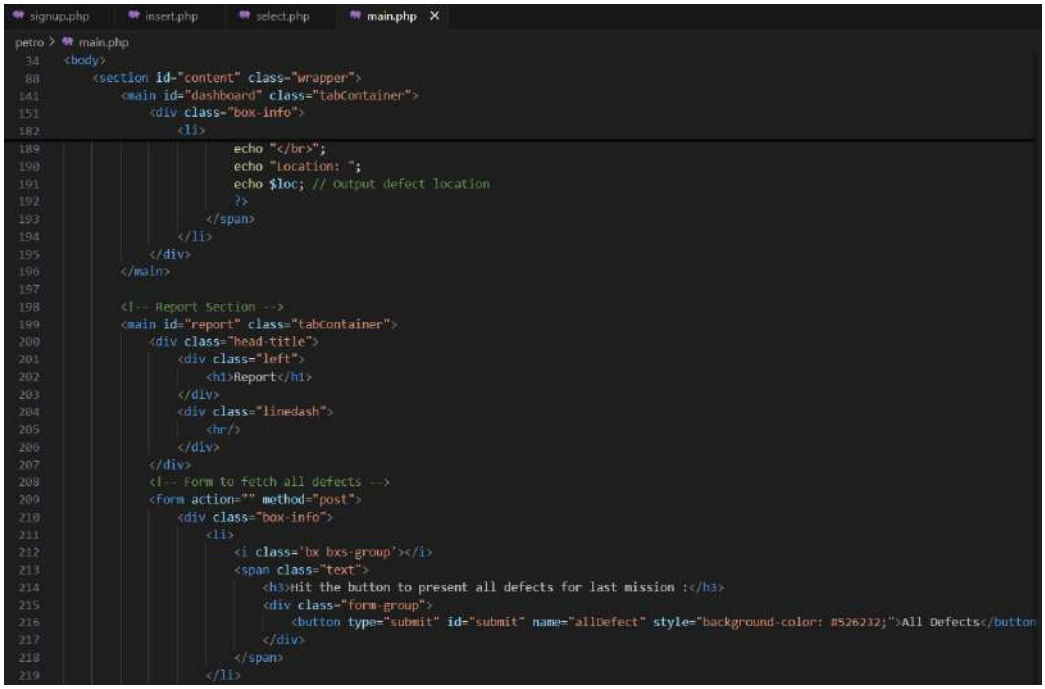
```
signup.php  insert.php  select.php  main.php ×
petro > main.php
34    <body>
88        <section id="content" class="wrapper">
103           <main id="parameters" class="tabContainer">
114               <form action="insertparam.php" method="post">
132                   </div>
133                   <div class="form-group">
134                       <button type="submit" id="submit">Set Parameters</button>
135                   </div>
136               </form>
137           </main>
138           <?php } ?>
139
140           <!-- Dashboard Section -->
141           <main id="dashboard" class="tabContainer">
142               <div class="head-title">
143                   <div class="left">
144                       <h1>Dashboard</h1>
145                   </div>
146                   <div class="linedash">
147                       <hr/>
148                   </div>
149               </div>
150
151               <div class="box-info">
152                   <li class="dash">
153                       <i class='bx bxs-category'></i>
154                       <span class="text">
155                           <h3>Defect :</h3>
156                           <div class="image-gallery">
157                               <?php
158                               // Database connection
159                               require_once "connect.php";
160
161                               // Fetch the most recent defect
162                               $queryselectid = "SELECT img, deflocation, defect FROM defect ORDER BY id DESC LIMIT 1;";
163                               $loc = "";
164                               $def = "";
```

Figure 8.31. HTML and PHP Code main page



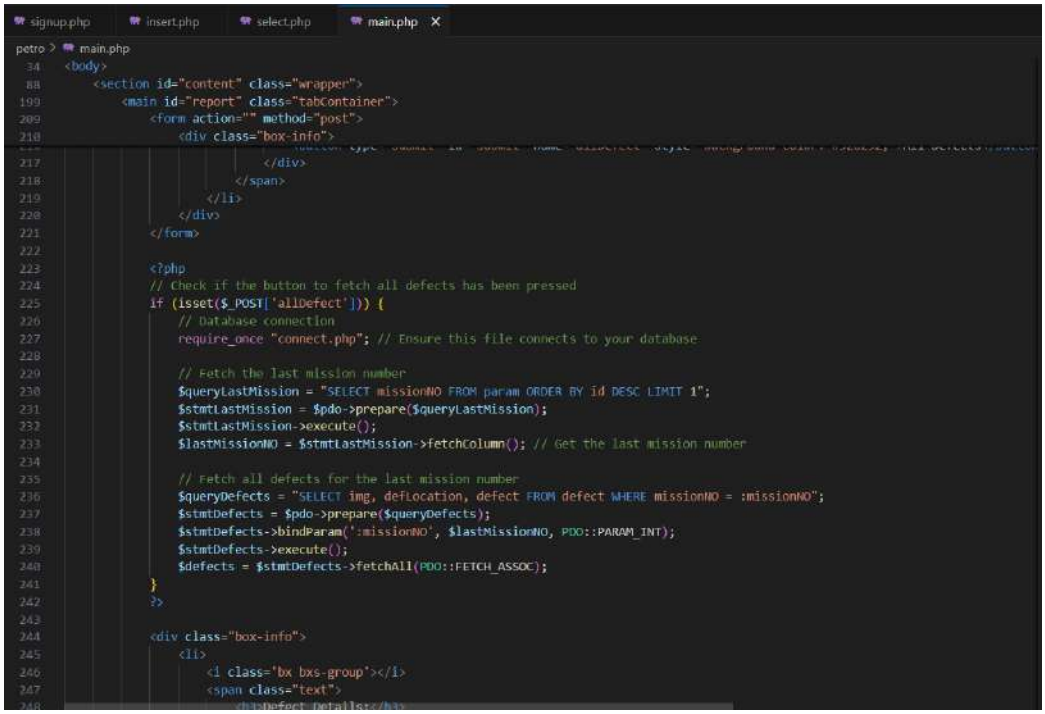
```
signup.php  insert.php  select.php  main.php ×
petro > main.php
 34   <body>
 88       <section id="content" class="wrapper">
141           <main id="dashboard" class="tabContainer">
151               <div class="box-info">
152                   <li class="dash">
162                           $queryselectid = "SELECT img, deflocation, defect FROM defect ORDER BY id DESC LIMIT 1;";
163                           $loc = "";
164                           $def = "";
165                           $stmtone = $pdo->prepare($queryselectid);
166                           $stmtone->execute();
167                           $results = $stmtone->fetchAll(PDO::FETCH_ASSOC);
168                           if(empty($results)){
169                               echo "No defects found";
170                           } else {
171                               foreach($results as $row){
172                                   $img = $row["img"]; // Image of the defect
173                                   $loc = $row['deflocation']; // Location of the defect
174                                   $def = $row['defect']; // Type of defect
175                                   echo '<img src="' . htmlspecialchars($img) . '" style="width:90%; height:90%; "/>'; // Display defe
176                               }
177                           }
178                           ?>
179                       </div>
180                   </span>
181               </li>
182               <li>
183                   <i class='bx bx-current-location'></i>
184                   <span class="text">
185                       <h3>Defect Type and Location:</h3>
186                       <?php
187                       echo "Type: ";
188                       echo $def; // Output defect type
189                       echo "</br>";
190                       echo "Location: ";
191                       echo $loc; // Output defect location
192                       ?>
```

Figure 8.32. HTML and PHP Code main page



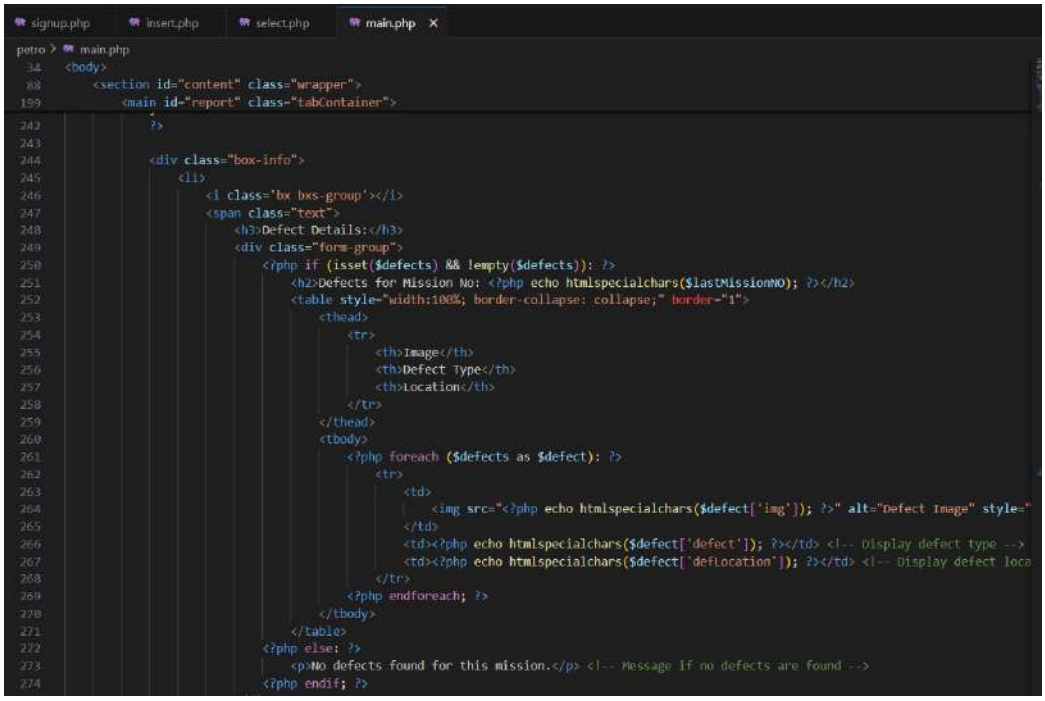
```
signup.php  insert.php  select.php  main.php ×
petro > main.php
 34   <body>
 88       <section id="content" class="wrapper">
141           <main id="dashboard" class="tabContainer">
151               <div class="box-info">
182                   <li>
189                           echo "</br>";
190                           echo "Location: ";
191                           echo $loc; // Output defect location
192                           ?>
193                       </span>
194                   </li>
195               </div>
196           </main>
197
198           <!-- Report Section -->
199           <main id="report" class="tabContainer">
200               <div class="head-title">
201                   <div class="left">
202                       <h1>Report</h1>
203                   </div>
204                   <div class="linedash">
205                       <hr/>
206                   </div>
207               </div>
208               <!-- Form to fetch all defects -->
209               <form action="" method="post">
210                   <div class="box-info">
211                       <li>
212                           <i class='bx bxs-group'></i>
213                           <span class="text">
214                               <h3>Hit the button to present all defects for last mission :</h3>
215                               <div class="form-group">
216                                   <button type="submit" id="submit" name="allDefect" style="background-color: #526232;">All Defects</button
217                               </div>
218                           </span>
219                       </li>
```

Figure 8.33. HTML and PHP Code main page



```
signup.php  insert.php  select.php  main.php
petro > main.php
  <body>
      <section id="content" class="wrapper">
          <main id="report" class="tabContainer">
              <form action="" method="post">
                  <div class="box-info">
                                  </div>
                              </span>
                          </li>
                  </div>
              </form>

              <?php
              // Check if the button to fetch all defects has been pressed
              if (isset($_POST['allDefect'])) {
                  // Database connection
                  require_once "connect.php"; // Ensure this file connects to your database

                  // Fetch the last mission number
                  $queryLastMission = "SELECT missionNO FROM param ORDER BY id DESC LIMIT 1";
                  $stmtLastMission = $pdo->prepare($queryLastMission);
                  $stmtLastMission->execute();
                  $lastMissionNO = $stmtLastMission->fetchColumn(); // Get the last mission number

                  // Fetch all defects for the last mission number
                  $queryDefects = "SELECT img, defLocation, defect FROM defect WHERE missionNO = :missionNO";
                  $stmtDefects = $pdo->prepare($queryDefects);
                  $stmtDefects->bindParam(':missionNO', $lastMissionNO, PDO::PARAM_INT);
                  $stmtDefects->execute();
                  $defects = $stmtDefects->fetchAll(PDO::FETCH_ASSOC);
              }
              ?>

              <div class="box-info">
                  <li>
                      <i class='bx bxs-group'></i>
                      <span class="text">
```

Figure 8.34. HTML and PHP Code main page



```
signup.php  insert.php  select.php  main.php
petro > main.php
  <body>
      <section id="content" class="wrapper">
          <main id="report" class="tabContainer">
              ?>

              <div class="box-info">
                  <li>
                      <i class='bx bxs-group'></i>
                      <span class="text">
                          <h3>Defect Details:</h3>
                          <div class="form-group">
                              <?php if (isset($defects) && !empty($defects)): ?>
                                  <h2>Defects for Mission No: <?php echo htmlspecialchars($lastMissionNO); ?></h2>
                                  <table style="width:100%; border-collapse: collapse;" border="1">
                                      <thead>
                                          <tr>
                                              <th>Image</th>
                                              <th>Defect Type</th>
                                              <th>Location</th>
                                          </tr>
                                      </thead>
                                      <tbody>
                                          <?php foreach ($defects as $defect): ?>
                                              <tr>
                                                  <td>
                                                      <img src="<?php echo htmlspecialchars($defect['img']); ?>" alt="Defect Image" style="
                                                  </td>
                                                  <td><?php echo htmlspecialchars($defect['defect']); ?></td> <!-- Display defect type -->
                                                  <td><?php echo htmlspecialchars($defect['defLocation']); ?></td> <!-- Display defect loca
                                              </tr>
                                          <?php endforeach; ?>
                                      </tbody>
                                  </table>
                              <?php else: ?>
                                  <p>No defects found for this mission.</p> <!-- Message if no defects are found -->
                              <?php endif; ?>
```

Figure 8.35. HTML and PHP Code main page

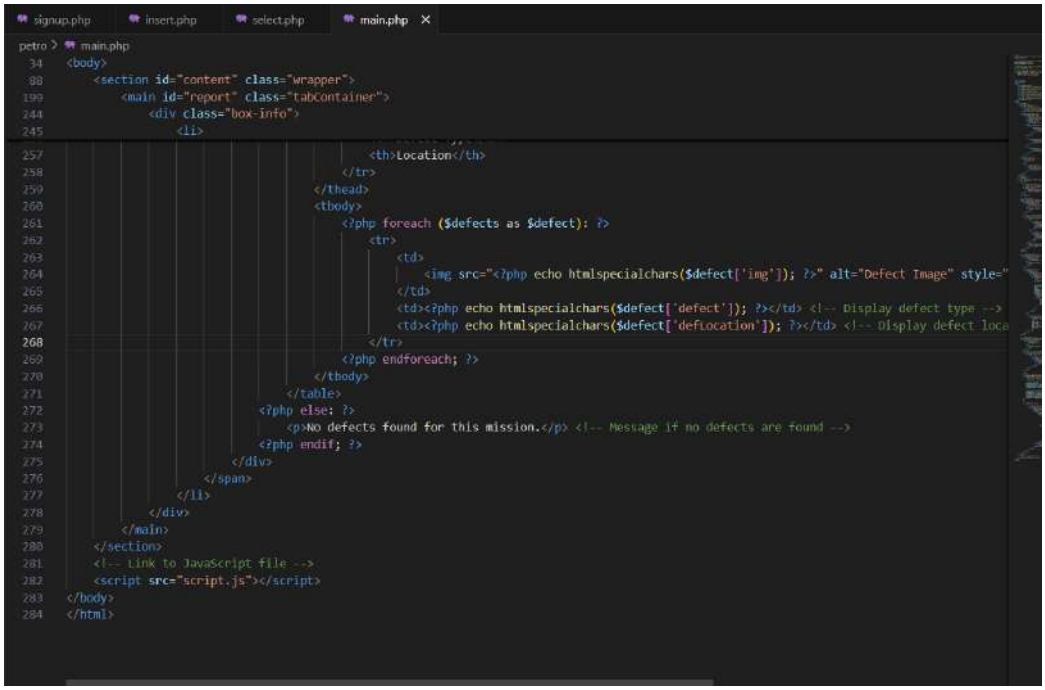


Figure 8.36. HTML and PHP Code main page

8.1.9.1 Parameters Insertion

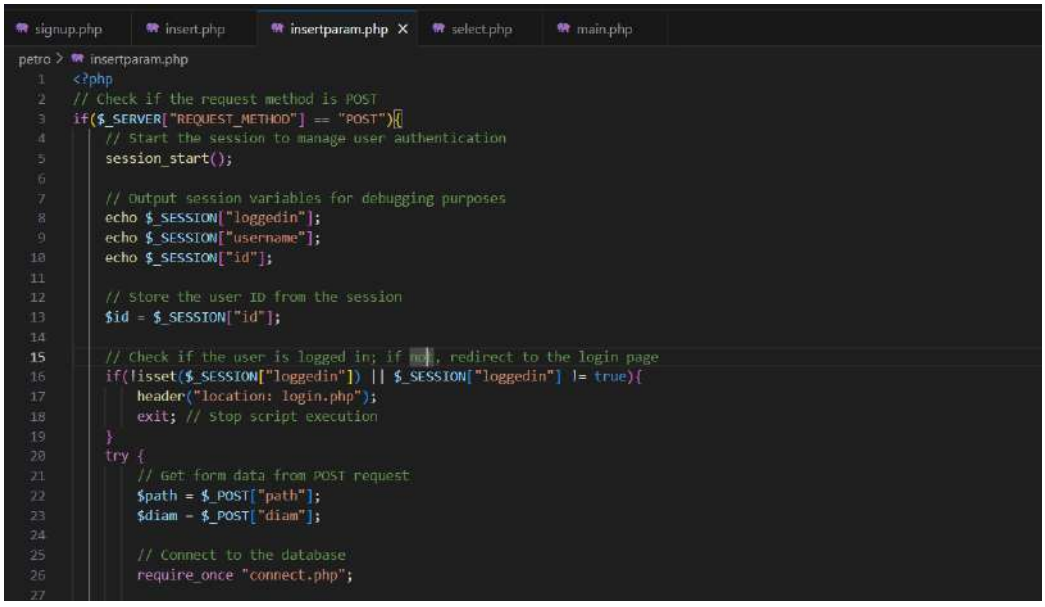
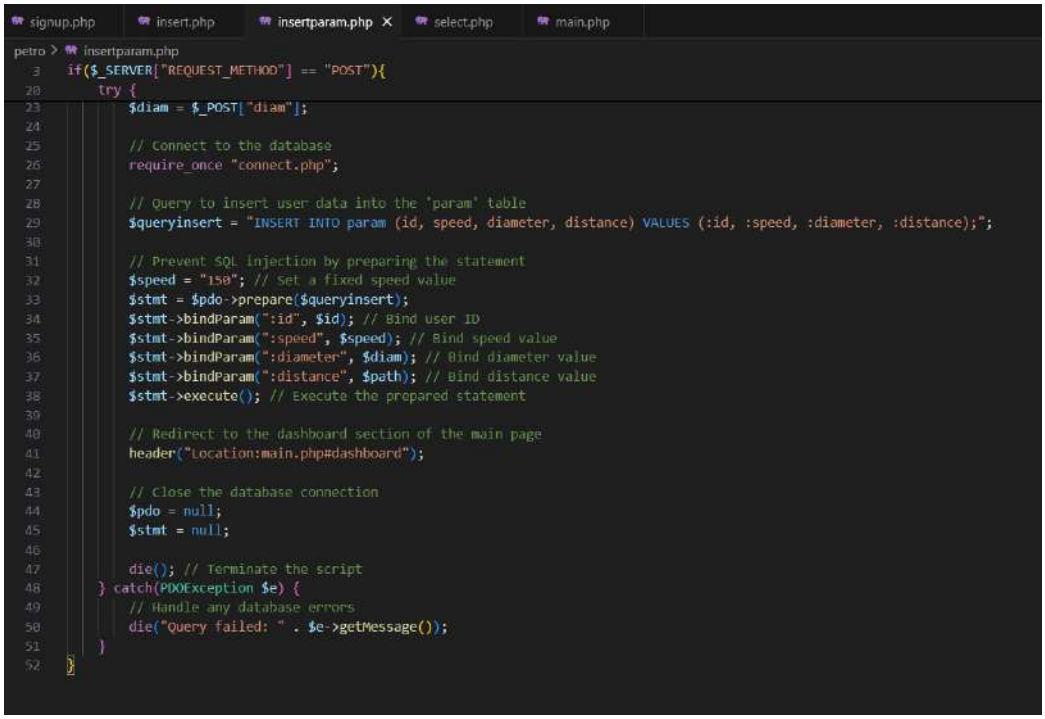


Figure 8.37. PHP Code to insert parameters in database



```php
if($_SERVER["REQUEST_METHOD"] == "POST"){
    try {
        $diam = $_POST["diam"];

        // Connect to the database
        require_once "connect.php";

        // Query to insert user data into the 'param' table
        $queryinsert = "INSERT INTO param (id, speed, diameter, distance) VALUES (:id, :speed, :diameter, :distance);";

        // Prevent SQL injection by preparing the statement
        $speed = "150"; // Set a fixed speed value
        $stmt = $pdo->prepare($queryinsert);
        $stmt->bindParam(":id", $id); // Bind user ID
        $stmt->bindParam(":speed", $speed); // Bind speed value
        $stmt->bindParam(":diameter", $diam); // Bind diameter value
        $stmt->bindParam(":distance", $path); // Bind distance value
        $stmt->execute(); // Execute the prepared statement

        // Redirect to the dashboard section of the main page
        header("Location:main.php#dashboard");

        // Close the database connection
        $pdo = null;
        $stmt = null;

        die(); // Terminate the script
    } catch(PDOException $e) {
        // Handle any database errors
        die("Query failed: " . $e->getMessage());
    }
}
```

Figure 8.38. PHP Code to insert parameters in database

Bibliography

[1] J.-H. Kim, "DESIGN OF a FULLY AUTONOMOUS MOBILE PIPELINE EXPLORATION ROBOT (FAMPER)," *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/268175443_DESIGN_OF_A_FULLY_AUTONOMOUS_MOBILE_PIPELINE_EXPLORATION_ROBOT_FAMPER

[2] dfrobot.com, "Raspberry Pi 4 Model B - 2GB." [Online]. Available: https://www.dfrobot.com/product-1876.html

[3] "Arduino Uno R3 ATmega328P SMD Micro (Micro USB)," 5 2022. [Online]. Available: https://www.cybertice.com/product/1030/arduino-uno-r3-atmega328p-%E0%B9%81%E0%B8%9A%E0%B8%9A-smd-micro-%E0%B9%80%E0%B8%9E%E0%B8%B4%E0%B9%88%E0%B8%A1%E0%B8%9E%E0%B8%AD%E0%B8%A3%E0%B9%8C%E0%B8%97%E0%B8%82%E0%B8%A2%E0%B8%B2%E0%B8%A2%E0%B8%82%E0%B8%B2-%E0%B9%84%E0%B8%A1%E0%B9%88%E0%B8%A1%E0%B8%B5%E0%B8%AA%E0%B8%B2%E0%B8%A2-micro-usb

[4] "Raspberry Pi Camera Module (G) OV5647 – ielectrony." [Online]. Available: https://ielectrony.com/en/product/%D9%83%D8%A7%D9%85%D9%8A%D8%B1%D8%A7-%D8%B1%D8%A7%D8%B3%D8%A8%D8%B1%D9%8A-%D8%A8%D8%A7%D9%8A-%D9%81%D8%A7%D8%A6%D9%82%D8%A9-%D8%A7%D9%84%D8%A7%D8%AF%D8%A7%D8%A1-ov5647/

[5] "Micro bit Lesson — Using the Ultrasonic Module « osoyoo.com," 9 2018. [Online]. Available: https://osoyoo.com/2018/09/18/micro-bit-lesson-using-the-ultrasonic-module/

[6] "HC-020K Wheel Speed Counting Sensor Encoder Kit – Robolabs.lk." [Online]. Available: https://robolabs.lk/product/hc-020k-wheel-speed-counting-sensor-encoder-kit/

[7] "TT DC Gearbox Motor." [Online]. Available: https://www.filotronix.com/dc-motor/616--tt-dc-gearbox-motor-.html

[8] Kuriosity, "Motor Driver Dual H Bridge L298N - Kuriosity," 4 2023. [Online]. Available: https://kuriosity.sg/products/motor-driver-dual-h-bridge-l298n

[9] "Oil, Gas Petroleum Industry." [Online]. Available: https://www.indx.com/en/industries/oil-gas#:~:text=The%20Oil%20%26%20Gas%20industry%20is,to%20industrial%20production%20and%20manufacturing.

[10] "Saudi Arabia - Oil Gas Petrochemicals," 1 2024. [Online]. Available: https://www.trade.gov/country-commercial-guides/saudi-arabia-oil-gas-petrochemicals

[11] L. R. Willey, "Pipeline construction safety practices," Ph.D. dissertation, National University.

[12] B. M. Norzuhri, "A study on Pipeline Inspection Gauge (PIG) Flow Parameters in Small Size Pipe," *A STUDY ON PIPELINE INSPECTION GAUGE (PIG) FLOW PARAMETERS IN SMALL SIZE PIPE*, 2 2013. [Online]. Available: http://umpir.ump.edu.my/7011/

[13] "Smart pigging Solutions." [Online]. Available: https://www.sgs.com/en/services/smart-pigging-solutions

[14] C. S. Copilot, "Intelligent Pigging procedure - How does it work? - Pigtek," 1 2024. [Online]. Available: https://www.pigtek.com/blog/intelligent-pigging-procedure-how-does-it-work/

[15] V. T. J. S. C. A. Hoang, "Effective time constraint management techniques for successful projects," 11 2023. [Online]. Available: https://viindoo.com/blog/business-management-3/time-constraint-1847

[16] B. Lutkevich and S. Lewis, "waterfall model," 11 2022. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model

[17] R. S. Elankavi, D. Dinakaran, A. S. A. Doss, R. K. Chetty, and M. Ramya, "Design and motion planning of a wheeled type pipeline inspection robot," *Journal of Robotics and Control/Journal of Robotics and Control (JRC)*, vol. 3, no. 4, pp. 415–430, 7 2022. [Online]. Available: https://doi.org/10.18196/jrc.v3i4.14742

[18] "Design of reconfigurable in-pipe exploration robots," 3 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8551187

[19] P. Oyekola, D. Somade, S. Syed, and O. Apis, "Application of computer vision in pipeline inspection robot," 03 2021.

[20] G. Bernasconi and G. Giunta, "Acoustic detection and tracking of a pipeline inspection gauge," *Journal of Petroleum Science and Engineering*, vol. 194, p. 107549, 11 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920410520306203

[21] "Robotic System with Power Line Communication for In-pipe Inspection of Underground Urban Gas Pipeline," 11 2019. [Online]. Available: https://ieeexplore.ieee.org/document/9043827

[22] H. Choi and S. Ryew, "Robotic system with active steering capability for internal inspection of urban gas pipelines," *Mechatronics*, vol. 12, no. 5, pp. 713–736, 6 2002. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0957415801000228

[23] A. Shukla and H. Karki, "A review of robotics in onshore oil-gas industry," in *2013 IEEE International Conference on Mechatronics and Automation*, 2013, pp. 1153–1160.

[24] "Towards Internet of Underground Things in smart lighting: A statistical model of wireless underground channel," 5 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8000155

[25] C. Cariou, L. Moiroux-Arvis, F. Pinet, and J.-P. Chanet, "Internet of Underground Things in Agriculture 4.0: Challenges, applications and Perspectives," *Sensors*, vol. 23, no. 8, p. 4058, 4 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/8/4058

[26] "Evaluation of LoRa LPWAN technology for remote health and wellbeing monitoring," 3 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7498898

[27] R. Singh, M. Baz, C. L. Narayana, M. Rashid, A. Gehlot, S. V. Akram, S. S. Alshamrani, D. Prashar, and A. S. AlGhamdi, "Zigbee and Long-Range Architecture Based Monitoring System for Oil Pipeline Monitoring with the Internet of Things," *Sustainability*, vol. 13, no. 18, p. 10226, 9 2021. [Online]. Available: https://www.mdpi.com/2071-1050/13/18/10226

[28] H. Li, R. Li, J. Zhang, and P. Zhang, "Development of a pipeline inspection robot for the Standard oil Pipeline of China National Petroleum Corporation," *Applied Sciences*, vol. 10, no. 8, p. 2853, 4 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/8/2853

[29] "Industrial Robot Automation | Rockwell Automation." [Online]. Available: https://www.rockwellautomation.com/en-us/capabilities/industrial-automation-control/robot-automation.html

[30] P. K. Panigrahi and S. K. Bisoy, "Localization strategies for autonomous mobile robots: A review," *Journal of King Saud University. Computer and information sciences/Mağala ğama al-malīk Saud : ùlm al-ḥasib wa al-malumat*, vol. 34, no. 8, pp. 6019–6039, 9 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821000550

[31] M. Durai, C.-W. Lan, and H. Chang, "In-line detection of defects in steel pipes using flexible gmr sensor array," *Journal of King Saud University - Science*, vol. 34, no. 2, p. 101761, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1018364721004237

[32] S. Kazeminasab and M. Banks, "Towards long-distance inspection for in-pipe robots in water distribution systems with smart navigation," 05 2021.

[33] G. Chhotaray and A. Kulshreshtha, *Defect detection in oil and gas pipeline: a machine learning application*, 9 2018. [Online]. Available: https://doi.org/10.1007/978-981-13-1274-8_14

[34] "IEEE Xplore Full-Text PDF:." [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9759370

[35] S. Kazeminasab and M. Banks, "Towards long-distance inspection for in-pipe robots in water distribution systems with smart motion facilitated by particle filter and multi-phase motion controller," *Intelligent Service Robotics*, vol. 15, 07 2022.

[36] H. Imran, M. Z. H. Zim, and M. Ahmmed, *PIRATE: Design and Implementation of Pipe Inspection Robot*, 05 2021, pp. 77–88.

[37] A. Kenzhekhan, A. Bakytzhanova, S. Omirbayev, Y. Tuieubayev, M. Daniyalov, and A. Yeshmukhametov, "Design and development of an In-Pipe mobile robot for pipeline inspection with AI defect detection System," *Design and development of an in-pipe mobile robot for pipeline inspection with AI defect detection system*, 10 2023. [Online]. Available: https://doi.org/10.23919/iccas59377.2023.10316817

[38] S. S. H. Hajjaj and I. B. Khalid, "Design and development of an inspection robot for oil and gas applications," *International Journal of Engineering & Technology*, vol. 7, no. 4.35, pp. 5–10, 2018.

[39] J. Butts, "What is the Raspberry Pi?" 9 2021. [Online]. Available: https://www.howtogeek.com/754492/what-is-raspberry-pi/

[40] Admin, "How encoders work in Robotics," 6 2023. [Online]. Available: https://roboticsmeta.com/how-encoders-work-in-robotics/

[41] "OV5647-75 FOV IR camera module for Raspberry Pi 3B+4B, suitable for large or night landscape surveillance." [Online]. Available: https://shorturl.at/TnQVu

[42] Q. Wang, X. Du, D. Jin, and L. Zhang, "Real-Time ultrasound doppler tracking and autonomous navigation of a miniature Helical robot for accelerating thrombolysis in dynamic blood flow," *ACS nano*, vol. 16, no. 1, pp. 604–616, 1 2022. [Online]. Available: https://doi.org/10.1021/acsnano.1c07830

[43] "DC Motor: What is it? how does it work? types, uses." [Online]. Available: https://www.iqsdirectory.com/articles/electric-motor/dc-motors.html

[44] Dejan, “L298N Motor Driver 8211; Arduino Interface, how it works, codes, schematics,” 2 2022. [Online]. Available: https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

[45] “Wire - Energy Education.” [Online]. Available: https://energyeducation.ca/encyclopedia/Wire

[46] N. Daly, “What is a use case?” 4 2024. [Online]. Available: https://www.wrike.com/blog/what-is-a-use-case/

[47] Nishadha and Creately, “Use Case Diagram Tutorial (Guide with Examples),” 12 2022. [Online]. Available: https://creately.com/guides/use-case-diagram-tutorial/

[48] GeeksforGeeks, “Use case diagrams | Unified Modeling Language (UML),” 7 2024. [Online]. Available: https://www.geeksforgeeks.org/use-case-diagram/

[49] Nishadha and Creately, “UML Class Diagram Relationships Explained with Examples,” 11 2022. [Online]. Available: https://creately.com/guides/class-diagram-relationships/

[50] omniVision, “Color CMOS QSXGA (5 megapixel) image sensor with OmniBSI technology,” Tech. Rep., 11 2009.

[51] “Simultaneous localization and mapping for inspection robots in water and sewer pipe networks: a review,” 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9548901

[52] J. Tang, “A Comparison of Encoder Technologies (and Selection Tips),” 5 2024. [Online]. Available: https://blog.orientalmotor.com/an-overview-of-optical-magnetic-and-capacitive-encoder-technologies-and-selection-tips

[53] P. 1, “DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC - Boutique Semageek.” [Online]. Available: https://boutique.semageek.com/en/1281-dc-gearbox-motor-tt-motor-200rpm-3-to-6vdc-3006152778672.html#:~:text=plastic%20gear%20motors%20(also%20known,a%20breadboard%20or%20terminal%20blocks.

[54] K. McAleer, “H-Bridges.” [Online]. Available: https://www.kevsrobots.com/resources/how_it_works/h-bridges.html

[55] W. contributors, "Command-line interface," 11 2024. [Online]. Available: https://en.wikipedia.org/wiki/Command-line_interface

[56] "Why Visual Studio code?" 11 2021. [Online]. Available: https://code.visualstudio.com/docs/editor/whyvscode

[57] David, "WHAT IS FRITZING, HOW DOES IT WORK AND HOW TO USE IT?" 6 2023. [Online]. Available: https://soldered.com/learn/what-is-fritzing-how-does-it-work-and-how-to-use-it/?srsltid=AfmBOoo_kWQ9gDTMGIqSmHGYb3FHaop9sxgxkLGSTxeF9nj8jiiNHSkm

[58] YoungWonks, "Raspberry Pi Imager," 5 2022. [Online]. Available: https://www.youngwonks.com/blog/Raspberry-Pi-Imager

[59] A. Van Den Boogaard, "Balancer: everything you need to know about BAL." [Online]. Available: https://weareblox.com/en-eu/balancer?utm_medium=search&utm_source=google

[60] f. given i=A, given=Alexey. Labeled vs unlabeled data: everything you need to know - trainingdata.pro. [Online]. Available: https://trainingdata.pro/labeled-vs-unlabeled-data

[61] J. Murel, PhD and E. Kavlakoglu, "Object Detection," 8 2024. [Online]. Available: https://www.ibm.com/topics/object-detection

[62] f. given i=X, given=Xiangqian and f. given i=X, given=Xing, "Research on surface defect detection algorithm of pipeline weld based on yolov7," vol. 14, no. 1. [Online]. Available: https://www.researchgate.net/publication/377597556_Research_on_surface_defect_detection_algorithm_of_pipeline_weld_based_on_YOLOv7

[63] f. given=Joseph, f. given=Santosh, f. given=Ross, f. given=Ali, University of Washington, Allen Institute for AI, and Facebook AI Research, "You only look once: Unified, real-time object detection." [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

[64] "How LoRa Tree enhances underground communications - Worldsensing," 11 2023. [Online]. Available: https://www.worldsensing.com/success-story/underground-monitoring/